Ritter's Crypto Glossary *and* Dictionary of Technical Cryptography

Technical Cryptographic Terms Explained

Laugh at deceptive claims!
Learn why cryptography cannot be guaranteed!

See Cryptography's Hall of Shame!: Crypto Controversies

A *Ciphers By Ritter* Page

Terry Ritter

2007 August 16

For a basic introduction to cryptography, see "Learning About Cryptography" @: http://www.ciphersbyritter.com/LEARNING.HTM. Please feel free to send comments and suggestions for improvement to: ritter@ciphersbyritter.com (you may need to copy and paste the address into a web email reader). You may wish to help support this work by patronizing "Ritter's Crypto Bookshop" at: http://www.ciphersbyritter.com/BOOKSHOP.HTM.

Index

Or use the browser facility "Edit / Find on this page" to search for particular terms.

Major Topics

problem, cryptanalysis, data compression, distinguisher, Dynamic Transposition, entropy, huge block cipher advantages, Kerckhoffs' requirements, known plaintext, multiple encryption, old wives' tale, one time pad, proof, randomness testing, really random, risk, scalable, snake oil, software patent, strength, term of art, threat model, trust, unpredictable

- **Cryptography:** additive RNG, Algebra of Secrecy Systems, balance, BB&S, bent function, block cipher and stream cipher models, block cipher, Boolean function nonlinearity, break, cascade ciphering, cipher, CBC, a cipher taxonomy, cipher testing, cipher system, Cloak2, cryptanalysis, Dynamic Substitution, Dynamic Transposition, entropy, jitterizer, known plaintext, homophonic substitution, huge block cipher advantages, Kerckhoffs' Requirements, known plaintext, letter frequencies, man-in-the-middle attack, Message, Mixing Cipher design strategy, multiple encryption, old wives' tale, one time pad, opponent, Perfect Secrecy, public-key cipher, risk analysis, stream cipher, s-box, secret-key cipher, standard cipher, Variable Size Block Cipher
- **Electronics:** amplifier, bypass, capacitor, clock, conductor, crystal oscillator, finite state machine, flip-flop, ground loop, hardware, inductor, jitter, logic function, logic level, metastability, negative resistance, noise, operational amplifier, oscillator, quality management, RC filter, resistor excess noise, resonance, semiconductor, shot noise, software, system design, thermal noise, transformer, transistor, transistor self-bias
- **Math:** associative, commutative, distributive, fast Walsh transform, field, finite field, group, Latin square, math notation, mod 2 polys, orthogonal Latin squares, primitive polynomial, proof, ring
- **Noise:** 1/f Noise, avalanche multiplication, Johnson noise, pink noise, really random, resistor excess noise, shot noise, thermal noise, white noise, zener breakdown
- **Patents:** intellectual property, patent changes, patent claims, patent complaints, patent consequences, patent infringement, patenting cryptography, patenting software, patent reading, patent valuation, prior art, software patent
- **Random:** additive RNG, finite state machine, linear feedback shift register, pseudorandom, random number generator, randomness testing, really random, RNG, unpredictable
- **Risk:** quality management, risk, risk analysis, risk management, single point of failure
- **Software:** risk management, Software Engineering, Structured Programming, system design, software patent
- **Statistics:** augmented repetitions, distribution, Kolmogorov-Smirnov, null hypothesis, one-sided test, population, population estimation, random, randomness testing, really random, universe

---

Contents

Introduction

Hadamard, Hamming Distance, Hardware, Hash, Hazzard, Heuristic, Hex, Hexadecimal, Hidden Markov Model, Hold Time, Homomorphism, Homophonic, Homophonic Substitution, HTTP Status Codes, Huge Block Cipher Advantages, Hybrid, Hypothesis

I

IDEA, Ideal Mixing, Ideal Secrecy, Identity, Identity Element, ignoratio elenchi, i.i.d., Imaginary Number, Impedance, Impedance Matching, Impossible, Improbable, Independent, Independent Variable, Inductive Reasoning, Inductor, Inference, Informal Proof, Information, Injective, Innuendo, Insulator, Integer, Integrity, Intellectual Property, Intermediate Block, Interval, Into, Invention, Inverse, Inverter, Invertible, Involution, Irreducible, Iterated Block Cipher, IV

J

Jitter, Jitterizer, Johnson Noise, Just Semantics

K

Karnough Map, KB, Kb, Kerckhoffs' Requirements, Key, Key Archives, Key Authentication, Key Distribution Problem, Key Loading, Key Loss, Key Management, Key Problems, Key Reuse, Key Selection, Key Storage, Key Transport, Keyed Substitution, Keyphrase, Key Schedule, Keyspace, Keystream, Keystroke, Known Plaintext, Known Plaintext Attack, Kolmogorov-Chaitin Complexity, Kolmogorov-Smirnov

L

Latency, Latin Square, Latin Square Combiner, Law, Layer, LC, Lemma, Letter Frequencies, LFSR, Lie, LIFO, Linear, Linear Complexity, Linear Cryptanalysis, Linear Equation, Linear Factor, Linear Feedback Shift Register, Linear Logic Function, Log$_2$, Logic, Logic Fallacy, Logic Function, Logic Level, lsb, lsB

M

MAC, M-Sequence, Machine Language, Magnetic Field, Man-in-the-Middle Attack, Mapping, Markov Chain, Markov Process, Mathematical Cryptography, Math Notation, Maximal Length, MB, Mb, MDS Codes, Mean, Mechanism, Mechanistic Cryptography, Median, Mere Semantics, Mersenne Prime, Message, Message Archives, Message Authentication, Message Authentication Code, Message Digest, Message Integrity, Message Key, Metastability, Method of Proof and Refutations, Military Grade, Misdirection, MITM, Mixing, Mixing Cipher, Mixing Cipher Design Strategy, Mod 2, Mod 2 Polynomial, Mode, Modulo, Monadic, Monoalphabetic Substitution, Monographic, Monoid, Monomial, msb, msB, M-Sequence, Multipermutation, Multiple Anagramming, Multiple Ciphering, Multiple Encryption

N

N, National Cryptologic Museum, Natural Number, Negative Resistance, NIST, Noise, Nomenclator, Nominal, non causa pro causa, Nonce, Nonlinearity, Nonrepudiaiton, non sequitur, Normal Distribution, NOT, Novelty, NSA, Null, Null Distribution, Null Hypothesis

O

OAEP, Object Code, Objective, Occam's Razor, Octal, Octave, OFB, Ohm's Law, Old Wives' Tale, oLs, One-Sided Test, One-Tailed Test, One Time Pad, One-To-One, One Way Diffusion, One Way Hash, Onto, Op Amp, Opcode, Operating Mode, Operational Amplifier, Opponent, Option, OR, Order, Ordinal, Orthogonal, Orthogonal Latin Squares, Oscillator, OTP, Output Feedback, Overall Diffusion

P

P-Value, Padding, Paradox, Parallel, Parity, Password, Patent, Patent Changes, Patent Claims, Patent Complaints, Patent Consequences, Patenting Cryptography, Patenting Software, Patent Infringement, Patent Reading, Patent Valuation, Path, Pedantic, Penknife, Perfect Secrecy, Period, Permutation, petitio principii, PGP, Phase, Phase Locked Loop, Phase Noise, Physically Random, Piezoelectric, Pink Noise, Pipeline, PKCS, PKI, Plagiarism, Plaintext, PLL, PN Sequence, Poisson Distribution, Polyalphabetic Combiner, Polyalphabetic Substitution, Polygram Substitution, Polygraphic, Polynomial, Polyphonic, Population, Population Estimation, post hoc ergo propter hoc, Power, Practice, PRBS, Precision, Predictable, Premise, Primitive, Primitive Polynomial, Prime, Prior Art, Privacy, PRNG, Probability, Process, Product Cipher, Proof, Propaganda, Proposition, Pseudorandom, PTO, Public Key Cipher, Public Key Infrastructure, Pure Cipher

Q

Qbit, Quality, Quality Management, Quantum Computer, Quantum Cryptography, Quartz, Quoting Out Of Context

R

R, R[x], R/I, Random, Randomness Testing, Random Number, Random Number Generator, Random Sampling, Random Variable, Random Walk, Range, Rationalization, Ratio Transformer, RC Filter, Reactance, Really Random, Real Number, Reasoning, Red, Red Herring, reductio ad absurdum, Redundancy, Relation, Relative Entropy, Relatively Prime, Relay, Reliability, Re-Originating Stream Cipher, Research Hypothesis, Resistor, Resistor Excess Noise, Resonance, Reverse CRC, Rhetoric, Ring, Ringing, Risk, Risk Analysis, Risk Management, Root, RMS, Root Mean Square, RNG, Round, RSA, Rule of Thumb, Running Key

S

SAC, Safe Prime, Salt, Sample, S-Box, Scalable, Scalar, Schematic Diagram, Science, Scientific Method, Scientific Model, Scientific Paper, Scientific Publication, Scrambler, Secrecy, Secret Code, Secret Key Cipher, Security, Security Through Obscurity, Seed, Self Inverse, Self-Synchronizing Stream Cipher, Semiconductor, Semigroup, Series, Session, Session Key, Set, Setup Time, Shannon, Shielding, Shift Register, Shot Noise, Shuffle, Sieve of Eratosthenes, Significance, Simple Substitution, Sine Wave, Single Point of Failure, Smooth Number, Snake Oil, Socrates, Socratic Method, Software, Software Engineering, Software Patent, Sophist, Sophistry, Source Code, S-P, Special Prime, Special Pleading, Specification, Speculation, Spin, SPN, Squares, Square Wave, SSL, Stability, Stake, Standard Cipher, Standard Deviation, State, State Machine, Static Electricity, Stationary Process, Statistic, Statistics, Steganography, Stochastic, Stochastic Process, Straw Man, Stream, Stream Cipher, Strength, Strict Avalanche Criterion (SAC), Structured Programming, Subjective, Substitution, Substitution Cipher, Substitution-Permutation, Substitution Table, Superencipherment, Superencryption, Superposition, Surjective, Switch, Switching Function, Symmetric Cipher, Symmetric Group, Synchronization, Synchronous, Synchronous Stream Cipher, Synthesis, System, System Design

T

Table Selection Combiner, Tautology, Taxonomy, Temperature, TEMPEST, Temporal Average, Term, Term of Art, Test, Theorem, Theory, Thermal Noise, Thesis, Threat, Threat Model, Time Constant, Trademark, Trade Secrecy, Traffic Analysis, Trajectory, Transformer, Transistor, Transistor Saturation, Transistor Self-Bias, Transposition, Transposition Cipher, Trap Door, Tree Analysis, Trinomial, Triple DES, TRNG, Trojan Horse, Truly Random, Trust, Truth Table, tu quoque, Tunneling, Two-Sided Test, Two-Tailed Test, Type I Error, Type II Error

U

UFN, Unary, Unbalanced Feistel Network, Uncertainty, Unexpected Distance, Unicity Distance, Uniform Distribution, Universe, Unknowable, Unpredictable, User Authentication

V

Variable, Variable Size Block Cipher, Variance, VCO, Vector, VENONA, Vernam Cipher, Vet, Voltage, Voltage Controlled Oscillator, Voltage Divider, Vulnerability

W

Walsh Functions, Walsh-Hadamard Transform, Weak Key, Weight, Whitening, White Noise, WHT, Wide Trail Strategy, Wire, Work Characteristic

X

XOR, XOR Encryption

Z

Z, Zener Breakdown, Zeroize, Z_n, Z_n*, (Z,+), (Z,+,*), Z/n, Z/p, Z/pZ, (Z/pZ)[x]

Introduction to the Crypto Glossary

This Glossary started as a way to explain the terms on my cryptography web pages describing my:

- **Inventions** (e.g., Dynamic Substitution, Dynamic Transposition, Balanced Block Mixing, Mixing Ciphers and Variable Size Block Ciphers);
- **Discoveries** (e.g., augmented repetitions);
- **Results** (e.g., Boolean function nonlinearity); and
- **Developments** (e.g., "checkerboard construction" of orthogonal Latin squares).

Having a Glossary meant I could *reduce* the text on most pages, while *expanding* background for the definitions, and *relating* the ideas to other similar, contradictory, or more basic ideas.

Why Bother with Definitions?

The value of a definition is insight. But:

- Simple descriptions are not always possible.
- Terms have meaning within particular contexts.
- Tedious examples may be required to expose the full meaning.

Good definitions can expose assumptions and provide a basis for reasoning to larger conclusions.

Consider the idea that cryptography is used to keep secrets: We expect a cipher to win each and every contest brought by anyone who wishes to expose secrets. We call those people opponents, but who are they really, and what can they do? In practice, we cannot know. Opponents operate in secret: We do not know their names, nor how many they are, nor where they work. We do not know what they know, nor their level of experience or resources, nor anything else about them. Because we do not know our opponents, we also do not know what they can do, including whether they can break our ciphers. Unless we know these things that cannot be known, we cannot tell whether a particular cipher design will prevail in battle. We cannot expect to know when our cipher has failed.

Even though the entire reason for using cryptography is to protect secret information, it is *by definition* ***impossible*** to know whether a cipher can do that. *Nobody* can know whether a cipher is strong enough, no matter how well educated they are, or how experienced, or how well connected, because they would have to know the opponents best of all. The definition of cryptography implies a contest between a cipher design and unknown opponents, and that means a successful outcome cannot be guaranteed by anyone.

Sometimes the Significance is Implied

Consider the cryptographer who says: "My cipher is strong," and the cryptanalyst who says: "I think your cipher is weak." Here we have two competing claims with different sets of possibilities: First, the cryptographer has the great disadvantage of not being able to prove cipher strength, nor to even list every possible attack so they can be checked. In contrast, the cryptanalyst might be able to actually *demonstrate* weakness, but only by dint of massive effort which may not succeed, and will not be compensated even if it does. Consequently, most criticisms will be *extrapolations*, possibly based on experience, and also possibly wrong.

The situation is inherently unbalanced, with a bias *against* the cryptographer's detailed and thought-out claims, and *for* mere handwave first-thoughts from anyone who deigns to comment. This is the ultimate conservative bias *against* anything new, and *for* the status quo. Supposedly the bias exists because if the cryptographer's claim is wrong user secrets might be exposed. But the old status-quo ciphers are in that same position. Nothing about an old cipher makes it necessarily strong.

Unfortunately, for users to benefit from cryptography they have to accept *some* strength argument. Even more unfortunately:

- Many years of trusted use do not testify about strength, but do provide both motive and time for opponents to develop secret attacks.
- Many failures to break a cipher do not imply it is strong.
- There can be no expertise on the strength of unbroken ciphers.

So on the one hand we need a cipher, and on the other have no way to know how strong the various ciphers are. For an industry, this is breathtakingly disturbing.

In modern society we purchase things to help us in some way. We go to the store, buy things, and they work. Or we notice the things do not work, and take them back. We know to take things back because we can see the results. Manufactured things work specifically because design and production groups can test which designs work better or worse or not at all. In contrast, if the goal of cryptography is to keep secrets, we generally cannot expect to know whether our cipher has succeeded or failed. Cryptography cannot test the fundamental property of interest: whether or not a secret has been kept.

The inability to test for the property we need is an extraordinary situation; perhaps no other manufactured thing is like that. Because the situation is unique, few understand the consequences. Cryptography is not like other manufactured things: nobody can trust it because nobody can test it. Nobody, anywhere, no matter how well educated or experienced, can test the ability of an unbroken cipher to keep a secret in practice. Thus we see how mere definitions allow us to deduce fundamental limitations on cryptography and cryptanalysis by simple reasoning from a few basic facts.

Relationships Between Ideas

The desire to expose relationships between ideas meant expanding the Glossary beyond cryptography *per se* to cover terms from related areas like electronics, math, statistics, logic and argumentation. Logic and argumentation are especially important in cryptography, where measures are few and math proofs may not apply in practice.

This Crypto Glossary is directed toward anyone who wants a better understanding of what cryptography can and cannot do. It is intended to address basic cryptographic principles in ways that allow them to be related, argued, and deeply understood. It is particularly concerned with fundamental limits on cryptography, and contradictions between rational thought and the current cryptographic wisdom. Some of these results may be controversial.

The Glossary is intended to build the fundamental understandings which lie at the base of all cryptographic reasoning, from novice to professional and beyond. It is particularly intended for users who wish to avoid being taken in by attacker propaganda. (Propaganda is an expected part of cryptography, since it can cause users to take actions which make things vastly easier for opponents.) The Glossary is also for academics who wish to see and avoid the logic errors so casually accepted by previous generations. One goal of the Glossary is to clarify the usual casual claims that confuse both novices and professionals. Another is to provide some of the historical technical background developed before the modern mathematical approach.

Reason in Cryptography

The way we understand reality is to follow logical arguments. All of us can do this, not just professors or math experts. Even new learners can follow a cryptographic argument, provided it is presented clearly. So, in this Glossary, one is occasionally expected to *actually follow* an argument and come to a personal conclusion. That can be scary when the result contradicts the conventional wisdom; then one then starts to question both the argument and the reasoning, as I very well know. But that scary feeling is just an expected consequence of a field which has allowed various unsupported claims and unquestioned beliefs to wrongly persist (see old wives' tales).

Unfortunately, real cryptography is *not* well-modeled by current math (for example, see proof and cryptanalysis). It is normally expected that the link between theory and reality is provided by the assumptions the math requires. (Obviously, proof conclusions only apply in practice when every assumed quality actually occurs in practice.) In math, each of these assumptions has equal value (since the lack of any one will void the conclusion), but in practice some assumptions are more equal than others. Certain assumptions conceivably *can* be guaranteed by the user, but other assumptions may be *impossible* to guarantee. When a model requires assumptions that cannot be verified in practice, that model cannot predict reality.

Current mathematical models almost *never* allow situations where the user can control every necessary assumption, making most proof results meaningless in practice. In my view, mathematical cryptography needs *practical* models. Of course, one might expect more realistic models to be less able to support the current plethora of mathematical results. Due to the use of more realistic models, some results in the Crypto Glossary do contradict well-known math results.

Opposing Philosophies

By carrying the arguments of conventional cryptographic wisdom to their extremes, it is possible to see two opposing groups, which some might call *theoretical* versus *practical.* While this simplistic model is far too coarse to take very seriously, it does have some basis in reality.

The Crypto Theorists supposedly argue that no cryptosystem can be trusted unless it has a mathematical proof, since anything less is mere wishes and hope. Unfortunately, there *is* no such cryptosystem. No cipher can be guaranteed strong in practice, and that is the real meaning of the one time pad. As long as even one unbreakable system existed, there was at least a possibility of others, but now there is no reason for such hope. The OTP is secure only in simplistic theory, and strength cannot be guaranteed in practice for users. This group seems most irritating when they imply that math proofs are most important, even when in practice those proofs provide no benefits to the user.

The Crypto Practitioners supposedly argue that systems should be designed to oppose the most likely reasonable threats, as in physical threat model analysis. In the physical world it is possible to make statements about limitations of opponents and attacks; unfortunately, few such statements can be made in cryptography. In cryptography, we know neither the opponents nor their attacks nor what they can do in combination. Successful attack programs can be reproduced and then applied by the most naive user, who up to that time had posed only the most laughable threat.

***Both* groups are wrong:** There will be no proof in practice, and speculating on the abilities of the opponents is both delusional and hopeless. Moreover, no correct compromise seems possible. Taking a little proof from one side and some threat analysis from the other simply is not a valid recipe for making secure ciphers.

There is a valid recipe for security and that is a growing, competitive industry of cipher development. Society needs more than just a few people developing a handful of ciphers, but actual design groups who continually innovate, design, develop, measure, attack and improve new ciphers in a continuing flow. That is expensive work, as the NSA budget clearly shows. Open society will get such results only if open society will pay for them. Since payment is the issue, it is clear that "free" ciphers act to oppose exactly the sort of open cryptographic development society needs.

Absent an industry of cipher design, perhaps the best we can do is to design systems in ways such that a cipher actually can fail, while the overall system retains security. That is redundancy, and is a major part of engineering most forms of life-critical systems (e.g., airliners), except for cryptography. The obvious start is multiple encryption.

What is the Point?

The practical worth of all this should be a serious regard for cryptographic risk. The possibility of cryptographic failure exists despite all claims and proofs to the contrary. Users who have something to protect must understand that cryptography has risks, and there is a real possibility of failure. If a possibility of information exposure is acceptable, one might well question the use of cryptography in the first place.

Even if users only want their information *probably* to be secure, they still have a problem: Only our opponents know our cipher failures, because they occur in secret. Our opponents do not expose our failures because they want those ciphers to continue in use. Few if any users will know when there is a problem, so we cannot count how many ciphers fail, and so cannot know that probability. Since there can be no expertise about what unknown opponents do, looking for an "expert opinion" on cipher failure probabilities or strength is just nonsense.

Conventional cryptographic expertise is based on the open literature. Unfortunately, unknown attacks can exist, and even the best informed cannot predict strength against them. While defending against *known* attacks may *seem* better than nothing, that actually may *be* nothing to opponents who have another approach. In the end, cipher and cryptosystem designers vigorously defend against attacks from academics who will not be their opponents.

On the other hand, even opponents read the open literature, and may make academic attacks their own. But surprisingly few academic attacks actually recover key or plaintext and so can be said to be real, practical threats. Much of the academic literature is based on strength assumptions which cannot be guaranteed or vulnerability assumptions which need not exist, making the literature less valuable in practice than it may appear.

Math cannot prove that a cipher is strong in practice, so we are forced to accept that any cipher may fail. We do not, and probably *can* not know the likelihood of that. But we do know that a single cipher is a single point of failure which just begs disaster. (Also see standard cipher.)

It is possible to design in ways which reduce risk. Systems can be designed with redundancy to eliminate the single point of failure (see multiple encryption). This is often the done in safety-critical fields, but rarely in cryptography. Why? Presumably, people have been far too credulous in accepting math proofs which rarely apply in practice. Thus we see the background for my emphasis on basics, reasoning, proof, and realistic math models.

Simple Encryption

To protect against fire, flood or other disaster, most software developers should store their current work off-site. The obvious solution is to first encrypt the files and then upload an archive to a web site. The straightforward use of cryptography to protect archives is an example of the pristine technical situation often seen as normal. Then we think of cipher strength and key protection, which seem to be all there is. But most cryptography is not that simple.

Climate of Secrecy. For any sort of cryptography to work, those who use it must not give away the secrets. Most times keeping secrets is as easy, or as hard, as just not talking or writing about them. Issues like minimizing paper output and controlling and destroying copies seem fairly obvious, although hardly business as usual. But secrets are almost always composed in plaintext, and the computers doing that may have plaintext secrets saved in various hidden operating system files. And opponents may introduce programs to compromise computers which handle secrets. It is thus necessary to control all forms of access to equipment which holds secrets, despite that being awkward and difficult. It is especially difficult to control access on the net.

Network Security. Computers only can do what they are told to do. When network designers decide to include features which allow attacks, that decision is as much a part of the problem as an attack itself. It seems a bit much to complain about insecurity when insecurity is part of the design. Design decisions have made the web insecure. Until web systems only implement features which maintain security, there can be none.

It is possible to design computing systems more secure than the ones we have now. If we provide no internal support for external attack, no attacks can prevail. The entire system must be designed to limit and control external web access and prevent surprises that slip by unnoticed. We can decompose the system into relatively small modules, and then test those modules in a much stronger way than trying to test a complex program. A possible improvement might be some form of restricted intermediate or quarantine store between the OS and the net. Better security design may mean that some things now supported insecurely no longer can be supported at all.

Current practice identifies two environments: The local computer, which is "fully" trusted, and the Internet, which is not trusted. This verges on a misuse of the concept of trust, which requires substantial consequences for misuse or betrayal. Absent consequences, trust is mere unsupported belief and provides no basis for reasoning. We do not trust a machine per se, since it only does what the designer made it do. And when there are no consequences for bad design, there really is no reason to trust the designer either.

A better approach would be fine OS control over individual programs, including individual scripts, providing validation and detailed limits on what each program can do, on a per-program basis. This would expand the firewall concept from just net access to every resource, including processor time, memory, all forms of I/O, *plus* the ability to invoke, or be invoked by, other programs. For example, most programs do not need, and so would not be allowed, net access, even if invoked by a program or running under a process which has such access. Programs received from the net would by default start out in quarantine, not have access to normal store, and could run only under strong limitations. A human would have to explicitly elevate them to a selected higher status, with the change logged. Program operation exceeding limitations would be prevented, logged, and accumulated in a control which supported validation, fine tuning, selective responses and serious quarantine.

Security is Off-The-Net. The best way to avoid web insecurity has nothing to do with cryptography. The way to avoid web insecurity is to not connect to the web, ever. Use a separate computer for secrets, and do not connect it to the net, or even a LAN, since computers on the LAN probably will be on the net. Carefully move information to and from the secrets computer with a USB flash drive. Protect access to that equipment.

Glossary Structure and Use

For most users, the Crypto Glossary will have many underlined (or perhaps colored) words. Usually, those are hypertext "links" to other text in the Glossary; just click on the desired link.

Links to my other pages generally offer a choice between a "local" link or a full web link. The user working from a downloaded copy of the Glossary only would normally use the full web links. The user working from a CD or disk-based copy of all my pages would normally use the local links.

Links to my other pages also generally open and use another window. (Hopefully that will avoid the need to reload the Glossary after a reference to another article.) Similarly, links *from my other pages* to terms in the Glossary also generally open a window specifically for the Glossary. (In many cases, that will avoid reloading the Glossary for every new term encountered on those pages.)

In cryptography, as in much of language in general, the exact same word or phrase often is used to describe two or more distinct ideas. Naturally, this leads to confused, irreconcilable argumentation until the distinction is exposed (and often thereafter). Usually I handle this in the Crypto Glossary by having multiple numbered definitions, with the most common usage (not necessarily the best usage) being number 1.

The worth of this Glossary goes beyond mere definitions. Much of the worth is the *relationships between ideas*: Hopefully, looking up one term leads to other ideas which are similar or opposed or which support the first. The Glossary is a big file, but breaking it into many small files would ruin much of the advantage of related ideas, because then most related terms would be in some other part. And although the Glossary could be compressed, that would generally *not* reduce download time, because most modems automatically compress data during transmission anyway. Dial-up users typically should download the Glossary onto local storage, then use it locally, updating periodically.

Value

I have obviously spent a lot of personal time constructing this Crypto Glossary, with the hope that it would be *more* than just background to my work. Hopefully, the Glossary and the associated introduction: "Learning About Cryptography" (see locally, or @: http://www.ciphersbyritter.com/LEARNING.HTM) will be of some wider benefit to the crypto community. So, if you have used this Glossary lately, why not drop me a short email and tell me so?

Feel free to tell me how much it helped or even how it failed you; perhaps I can make it better for the next guy. If you use web email, just copy and paste my email address: ritter@ciphersbyritter.com

---

1/f Noise

In electronics, a random-like analog signal with amplitude proportional to the inverse of frequency. Also called "flicker." Well known both in semiconductor electronics and physics. Not a white noise, but a pink noise.

Resistor excess noise is a 1/f noise generated in non-homogenous resistances, such as the typical thick-film surface-mount (SMT) resistor composed of conductor particles and fused glass. It is thought that DC current forms a preferential path through the conductive grains, a path that varies dynamically at random, thus modulating the resistance and creating noise. Homogenous metal films do not have 1/f noise.

The especially large amount of 1/f noise in MOSFET's could be understood if the glass layer the gate rests on is unexpectedly rough. That could could create islands of conduction (which some literature appears to support), which then act like resistive grains.

In a single-crystal semiconductor, 1/f noise may be related to the organization of atomic bonding at the outside surfaces, which must be different than inside the crystalline bulk material. If the semiconductor surface could be shown to be composed of conductive islands, that would be an enlightening result.

8b10b

A block code which represents 8-bit data as 10-bit values or codewords. This gives 1024 codewords to encode 256 values plus perhaps 12 new control codes; this freedom can be used to approach general bit balance in each codeword. However, since a 10-bit code has only 252 balanced values (whereas 256 data and perhaps 12 control symbols are required), balancing must extend across codewords. The encoding process must maintain a count of the current unbalance and correct that in the next codeword. Also see coding theory.

Abelian

In abstract algebra, a commutative group or ring.

Absolute

In the study of logic, something observed similarly by most observers, or something agreed upon, or which has the same value each time measured. Something not in dispute, unarguable, and independent of other state. As opposed to contextual.

AC

Alternating Current: Electrical power which repeatedly reverses direction of flow. As opposed to DC.

Generally used for power distribution because the changing current supports the use of transformers. Utilities can thus transport power at high voltage and low current, which minimize "ohmic" or I^2R losses. The high voltages are then reduced at power substations and again by pole transformers for delivery to the consumer.

Academic

1. Scholarly.
2. Theoretical.
3. Conventional.
4. Impractical.

Academic Break

A break or technically successful attack which is also impractical. See: academic.

Access
The ability (right or permission) to interact with (approach, enter, speak with, read or use) some one or some thing.

Access Control
A possible goal of cryptography. The idea of restricting documents, equipment or keys to those authorized such access.

Accountability
Nonrepudiation. A goal of cryptography. Responsibility for messages sent.

Accuracy
The ratio of a measured value to the true value. A percentage of the measured value, surrounding the measured value, which is supposed to contain the correct value. Also see: precision.

Acronym
A word constructed from the beginning letters of each word in a phrase or a name or a title.

Active
1. In motion, or in use. In contrast to "passive."
2. An S-box whose input has changed.

Additive Combiner
An additive combiner uses numerical concepts similar to addition to mix multiple values into a single result. This is the basis for conventional stream ciphering. Also see extractor.

One example is byte addition modulo 256, which simply adds two byte values, each in the range 0..255, and produces the remainder after division by 256, again a value in the byte range of 0..255. The modulo is automatic in an addition of two bytes which produces a single byte result. Subtraction is also an "additive" combiner.

Another example is bit-level exclusive-OR which is addition mod 2. A byte-level exclusive-OR is a polynomial addition.

Additive combiners are linear, in contrast to nonlinear combiners such as:

- Latin square combiners, and
- Dynamic Substitution combiners.

Additive RNG
(Additive random number generator.) A LFSR-based RNG typically using multi-bit elements and integer addition (instead of XOR) combining. References include:

Knuth, D. 1981. *The Art of Computer Programming,* Vol. 2, *Seminumerical Algorithms.* 2nd ed. 26-31. Addison-Wesley: Reading, Massachusetts.

Marsaglia, G. and L. Tsay. 1985. Matrices and the Structure of Random Number Sequences. *Linear Algebra and its Applications.* 67:147-156.

Advantages include:

- A long, mathematically proven cycle length.

- Especially efficient software implementations.
- Almost arbitrary initialization (some element must have its least significant bit set).
- A simple design which is easy to get right.

In addition, a vast multiplicity of independent cycles has the potential of confusing even a quantum computer, should such a thing become realistic.

```
For Degree-n Primitive, and Bit Width w
   Total States:       2^nw
   Non-Init States:    2^n(w-1)
   Number of Cycles:   2^(n-1)(w-1)
   Length Each Cycle:  (2^n-1)2^(w-1)
   Period of lsb:      2^n-1
```

The binary addition of two bits with no carry input is just XOR, so the lsb of an Additive RNG has the usual maximal length period.

A degree-127 Additive RNG using 127 elements of 32 bits each has 2^{4064} unique states. Of these, 2^{3937} are disallowed by initialization (the lsb's are all "0") but this is just one unusable state out of 2^{127}. There are still 2^{3906} cycles which *each* have almost 2^{158} steps. (The Cloak2 stream cipher uses an Additive RNG with 9689 elements of 32 bits, and so has 2^{310048} unique states. These are mainly distributed among 2^{300328} different cycles with almost 2^{9720} steps each.)

Like any other LFSR, and like any other RNG, and like any other FSM, an Additive RNG is very weak when standing alone. But when steps are taken to hide the sequence (such as using a jitterizer nonlinear filter and Dynamic Substitution combining), the resulting cipher can have significant strength.

Additive Stream Cipher

The conventional stream cipher, based on simple additive combining.

Adduction

In argumentation, the process of synthesizing a deep understanding of meaning, based on the analysis of multiple examples. Often used in the Socratic Method.

ad hoc

Something established for a particular one-time purpose.

AES

Advanced Encryption Standard. A 128-bit or 256-bit conventional block cipher replacement for DES. The new block cipher chosen by NIST for general use by the U.S. Government.

The mechanics of AES are widely available elsewhere. Here I note how one particular issue common to modern block ciphers is reflected in the realized AES design. That issue is the size of the implemented keyspace compared to the size of the potential keyspace for blocks of a given size.

A Block Cipher Model

A common academic model for conventional block ciphers is a "family of permutations." The "permutation" part of this means that every plaintext block value is found as ciphertext, but generally in a different position. The "family" part of this can mean every possible permutation. However, modern block ciphers key-select only an infinitesimal fraction of those possibilities.

Suppose we have a block which may take on any of *n* different values. How many ways can those *n* block values be rearranged as in a block cipher? Well, the first value can be placed in any of the *n* possible positions, but that fills one position so the second value has only *n*-1 positions available. Continuing on, the third has *n*-2 possibilities and so on for *n* different factors. Thus we find that the number of options is the same as the definition of factorial. The number of distinct permutations of *n* different values is *n*-factorial.

The Corresponding AES Model

A 128-bit key can select 2^{128} emulated tables. However, a 128-bit block has an alphabet of about 3.4×10^{38} different values, and so could have 3.4×10^{38} *factorial* emulated tables. That value is BIG, BIG, BIG, but still within range of my JavaScript page:

- "JavaScript Powers and Factorials Computation," locally, or @: http://www.ciphersbyritter.com/JAVASCRP/PERMCOMB.HTM#Powers

There we find that 3.4×10^{38} different values have on the order of $2^{(1040)}$ distinct permutations. That value would take 10^{40} bits to represent, and can be directly compared to the 256 bits needed to represent the larger keys used in AES.

A 128-bit block can be any one of 2^{128} or 3.4×10^{38} different values. To form a particular permutation, the first value can be placed in any of 3.4×10^{38} places, the second in 3.4×10^{38}-1 places, and so on for 3.4×10^{38} different factors. As a ballpark calculation, we might expect 3.4×10^{38}-factorial to be similar to $(10^{38})^{1038}$. That would be the same as 2 to the power 10^{38} log2 10^{38}, which is 2 to the power 10^{38} * 128, and that is about 2 to the power 10^{40}, nicely confirming the JavaScript results.

AES Reality

For 128-bit blocks and 256-bit keys, AES provides:

- 2^{256} keyed or emulated tables, out of about
- $2^{10,000,000,000,000,000,000,000,000,000,000,000,000,000}$ possibilities.

The obvious conclusion is that *almost none* of the keyspace implicit in the theoretical model of a conventional block cipher is actually implemented in AES, and that is consistent with other modern designs. Is that important? Apparently not, but nobody really knows. It does seem to imply that just a few known plaintext blocks should be sufficient to identify the correct key from a set of possibilities, which might make known plaintext more of an issue than normally claimed. Does it lead to a known break? No, or at least not yet. But having only a tiny set of keyed permutations should lead to questions about patterns and relationships within the selected set.

The real issue here is not the exposure of a particular weakness in AES, since no such weakness is shown. Instead, the issue is that conventional cryptographic wisdom does not force models to correspond to reality, and poor models lead to errors in reasoning. The distinction between theory and practice is pronounced in cryptography. For other examples of failure in the current cryptographic wisdom, see one time pad, BB&S, DES, and, of course, old wives' tale.

Is AES Enough for Government Secrets?

AES is said to be certified for SECRET and TOP SECRET classified material. That might have us believe that AES is trusted by NSA, but it may mean less than it seems.

No cipher, by itself, can guarantee security. Any cryptographic system will have to be certified by NSA before protecting classified information. In practice, cryptosystems will be provided by NSA to contractors, those systems may or may not use AES, and they may not use AES in the expected form. That does not imply that AES is bad, it just means that we cannot really know what NSA will allow, despite general claims.

Affine

Generally speaking, linear.

Technically, function *f* : G -> G of the form:

```
f(x) = ax + b
```

with non-zero constant "b".

Affine Boolean Function

A Boolean function which can be represented in the form:

$$a_n x_n + a_{n-1} x_{n-1} + \ldots + a_1 x_1 + a_0$$

where the operations are mod 2: addition is Exclusive-OR, and multiplication is AND.

Note that all of the variables x_i are to the first power only, and each coefficient a_i simply enables or disables its associated variable. The result is a single Boolean value, but the constant term a_0 can produce either possible output polarity.

Here are all possible 3-variable affine Boolean functions (each of which may be inverted by complementing the constant term):

affine	truth table							
c	0	0	0	0	0	0	0	0
x0	0	1	0	1	0	1	0	1
x1	0	0	1	1	0	0	1	1
x1+x0	0	1	1	0	0	1	1	0
x2	0	0	0	0	1	1	1	1
x2+ x0	0	1	0	1	1	0	1	0
x2+x1	0	0	1	1	1	1	0	0
x2+x1+x0	0	1	1	0	1	0	0	1

See also: Boolean function nonlinearity.

Affine Cipher

One of the classic hand-ciphers, described mathematically as

```
F(x) = ax + b (mod n)
```

where the non-zero term makes the equation affine.

Most of the classic hand-ciphers can be seen as simple substitution stream ciphers. Each plaintext letter selects an entry in the substitution table (for that cipher), and the contents of that entry becomes the ciphertext letter. The affine equation thus represents one way to set up the table, as a particular simple permutation of the letters in the table. (Of course, by using the equation we need no explicit table, but we also constrain ourselves to the simplicity of the equation.) To assure that we have a permutation, we require that a and n be relatively prime, that is, the gcd(a,n) = 1, or in number theory notation, just (a,n) = 1.

In modern terms, the strength of the classic substitution ciphers is essentially nil. In modern cryptanalysis, we generally assume that the opponent has a substantial amount of known plaintext. Since the table does not change, every known-plaintext character has the potential to fill in another entry in the table. Very soon the table is almost completely exposed, which ends all strength. These simple substitution ciphers with small, fixed tables (or even just equations for such tables) are also extremely vulnerable to attacks using ciphertext only.

Algebra

The use of variables and the valid manipulation of expressions in the study of numbers.

Algebraic Normal Form

(ANF). Typically, the symbolic representation of a mapping in the usual sum-of-products form.

- For Boolean functions in symbolic form, each term is an input variable combination for which the output is '1'.
- For Boolean functions in explicit form, basically a truth table: simply a list of the output value as it will occur when stepping through all possible input variable combinations, one-by-one. This is just the bit sequence of the output value as it would occur in input-variable order.

Algebra of Secrecy Systems

An oft-overlooked proposal by Shannon, describing both the construction of a multiple encryption cipher (called the "product") *and* the keyed selection of one from among many ciphers (called the "weighted sum").

> "The first combining operation is called the product operation and corresponds to enciphering the message with the first secrecy system *R* and enciphering the resulting cryptogram with the second system *S*, the keys for *R* and *S* being chosen independently."
>
> "The second combining operation is 'weighted addition.'
>
> $S = pR + qS \quad p + q = 1.$
>
> It corresponds to making a preliminary choice as to whether system *R* or *S* is to be used with probabilities *p* and *q*, respectively. When this is done or *R* or *S* is used as originally defined." [p.658]

More specifically (and with a change of notation):

> "If we have two secrecy systems *T* and *R* we can often combine them in various ways to form a new secrecy system *S*. If *T* and *R* have the same domain (message space) we may form a kind of 'weighted sum,'
>
> $S = pT + qR$
>
> where $p + q = 1$. This operation consists of first making a preliminary choice with probabilities *p* and *q* determining which of *T* and *R* is used. This choice is part of the key of *S*. After this is determined *T* or *R* is used as originally defined. The total key of *S* must specify which of *T* and *R* is used, and which key of *T* (or *R*) is used."
>
> "More generally we can form the sum of a number of systems.
>
> $S = p_1T + p_2R + . . . + p_mU \quad \mathrm{Sum}(p_i) = 1$
>
> We note that any system *T* can be written as a sum of fixed operations
>
> $T = p_1T_1 + p_2T_2 + . . . + p_mT_m$
>
> T_i being a definite enciphering operation of *T* corresponding to key choice *i*, which has probability *p*."
>
> "A second way of combining two secrecy systems is taking the 'product,' . . . . Suppose *T* and *R* are

two systems and the domain (language space) of *R* can be identified with the range (cryptogram space) of *T*. Then we can apply first *T* to our language and then *R* to the result of this enciphering process. This gives a resultant operation *S* which we write as a product

$$S = RT$$

The key for *S* consists of both keys of *T* and *R* which are assumed chosen according to their original probabilities and independently. Thus if the *m* keys of *T* are chosen with probabilities

$$p_1 p_2 \; . \; . \; . \; p_m$$

and the *n* keys of *R* have probabilities

$$p'_1 p'_2 \; . \; . \; . \; p'_n \; ,$$

then *S* has at most *mn* keys with probabilities $p_i\,p'_j$. In many cases some of the product transformations $R_i\,T_j$ will be the same and can be grouped together, adding their probabilities.

"Product encipherment is often used; for example, one follows a substitution by a transposition or a transposition by a Vigenere, or applies a code to the text and enciphers the result by substitution, transposition, fractionation, etc."

"It should be emphasized that these combining operations of addition and multiplication apply to secrecy systems as a whole. The product of two systems *TR* should not be confused with the product of the transformations in secrecy systems $T_i\,R_j$ . . . ."

-- Shannon, C. E. 1949. Communication Theory of Secrecy Systems. *Bell System Technical Journal.* 28:656-715.

It is easy to dismiss this as being of historical interest only, but there are advantages here which are well beyond our current usage.

For the keyed selection among ciphers, there would be some sort of simple protocol (i.e., not cryptographic *per se*), for communicating cipher selections to the deciphering end. (Perhaps there would be some sort of simple handshake for email use.) The result would be to have (potentially) a new selection from a set of ciphers on a message-by-message basis.

- Having frequent cipher changes guarantees that we *can* change ciphers, immediately and easily, if any cipher we use is found weak.
- A cipher change terminates any existing break of a particular cipher which has been exposing our information. Since we cannot expect to know when a break exists, changing to a different cipher can minimize the effect of a cipher fault even though we know nothing about that fault.
- Using different ciphers at different times prevents information from being concentrated under a single cipher. This prevents the opposing attack budget from concentrating on one target.
- The ability to easily change ciphers supports the continued creation and use of new ciphers, which the opponents must then identify, obtain, analyze and break. Although single new cipher design costs can be distributed among users simply by selling product, each opponent must bear the full cost of analysis, since most attackers cannot cooperate. And as the set of ciphers continues to grow, the opponents may never catch up to the complete set of ciphers actually in use.
- Cipher selection has minimal execution cost.

With respect to multiple encryption or ciphering "stacks" (as in "protocol stacks"), there are various security advantages:

- A cipher stack prevents a single broken cipher from exposing our information. Since any particular cipher may be broken and we will not know (the opponents do not tell us), this protects against a dangerous single point of failure.
- A three-cipher stack hides the known-plaintext (and "defined-plaintext") information for each individual cipher. Such information simply is not exposed to the opponents, which thus prevents known-plaintext attacks (and defined-plaintext attacks) on the individual ciphers. The construction thus eliminates whole classes of attack on the component ciphers.
- A three-cipher stack gives us exponentially many different ciphering stack possibilities. The intent here is *not* to add keyspace, since reasonable ciphers already have enough keyspace. Instead, the point is the easy construction of many *conceptually different* overall ciphering functions which the opponents must engage.
- Users who are "Nervous Nellies" could specify that their particular favorite cipher would always be part of the (changing) cipher stack, thus "guaranteeing" at least as much strength as using that cipher alone. (If the adjacent cipher was the same, in decipher mode, using the same key, then there would be no strength. So do not do that. If an arbitrary cipher was likely to reduce strength, that would be an attack, and we see no such attack.)
- A three-layer cipher stack obviously has an execution cost of three layers of ciphering.

Also see: Perfect Secrecy and Ideal Secrecy.

Algorithm

The description of a sequence of operations which does something. Typically,

- a finite procedure,
- composed of discrete steps,
- expressed in a fixed instruction vocabulary.

Also see heuristic, Structured Programming and software patent.

An algorithm intended to execute reliably as a computer program necessarily must handle, or in some way at least deal with, absolutely every error condition which can possibly occur in operation. (We do assume functional hardware, and thus avoid programming around the possibility of actual hardware faults, such as memory or CPU failure.) These "error conditions" normally include Operating System errors (e.g., bad parameters passed to an OS operation, resource not available, various I/O failures, etc.), and arithmetic issues (e.g., division by zero, overflow, etc.) which may halt execution when they occur.

Other possibilities include errors the OS will not know about, including the misuse of programmer-defined data structures, such as buffer overrun.

A practical algorithm must recognize various things which validly may occur, even if such things are exceedingly rare. One example might be in assuming that two floating-point variables which represent the same value will be equal. Another example might be to assume that a floating-point variable will "never" have some particular value (which might lead to a divide-by-zero fault). Yet another example would be to assume that an arbitrary selection of x will lead to a sufficiently long cycle in BB&S, even if the alternative is very, very unlikely.

Algorithmic Complexity

Kolmogorov-Chaitin complexity.

Alias File

My term for a cipher system computer file relating *names* to *keys*. This allows ordinary users to specify which key is to be used by using the far-end name, without knowing the actual key itself. Thus, the actual key can be long and random and can change over time and the user need not coordinate these changes.

In particular, my Cloak2 and Penknife ciphers implemented encrypted alias files of text lines of arbitrary length, each of which included name, start date, and key. New keys were made available only as secure ciphertext, but the alias files were arranged so they could consist of multiple ciphertext files simply concatenated *as ciphertext*. Thus, new keys could be added to the start of the alias file just using a simple and secure file copy operation. When searching for a particular alias, the date was also checked, and that key used only when the correct date had arrived. This allowed an entire office of users to change to a new key *automatically*, at the same time, *without even knowing* they were using a different key. Appropriate functions allowed access to old keys so that email traffic could be archived in ciphertext form.

Obviously, an alias file *must* be encrypted. The single key or keyphrase decrypting an alias file thus provides access to all the keys in the file. But each alias file contains only a subset of the keys in use within an organization, and even those are only valid over a subset of time. An organization security officer could archive old alias files, strip out the old keys and add new ones, then encipher the new alias file under a new pass phrase. In this way, the contents of old encrypted email would not be hidden from the authorizing organization. Alias file maintenance could be either as complex or as simple as one might like.

See, for example,

- the key management features section of my "Cloak2 Features" piece locally, or @: http://www.ciphersbyritter.com/CLO2FEA.HTM#KeyMgt
- and the alias file usage section of my "Cloak2 Cipher User's Manual" locally, or @: http://www.ciphersbyritter.com/PROD/CLO2DOC3.HTM#DetAF

describing the Cloak2 cipher.

Allan Variance

A descriptive variance statistic based on deviation from preceding sample. This is computed as the sum of the squares of differences between each sample and the previous sample, divided by 2, and divided by the number of samples-1.

Allan Variance is useful in analysis of residual noise in precision frequency measurement. Five different types of noise are defined: white noise phase modulation, flicker noise phase modulation, white noise frequency modulation, flicker noise frequency modulation, and random walk frequency modulation. A log-log plot of Allan variance versus sample period produces approximate straight line values of different slopes in four of the five possible cases. A different (more complex) form called "modified Allan deviation" can distinguish between the remaining two cases.

Also see

- the conversation "Allan Variance and Deviation," locally, or @: http://www.ciphersbyritter.com/NEWS6/ALLANVAR.HTM

All or Nothing Transform

(AONT). Basically the idea of a block mixing function in which knowing even all but one of the mixed outputs exposes none of the original input block values. As defined by Rivest in 1997:

> "**Definition.** A transformation *f* mapping a message sequence $m_1,m_2,...,m_s$ into a pseudo-message sequence $m_1',m_2',...,m_s'$ is said to be an *all-or-nothing transform* if:
>
> - The transform *f* is reversible: given the pseudo-message sequence, one can obtain the original message sequence.
> - Both the transformation *f* and its inverse are efficiently computable (that is, computable in polynomial time).
> - It is computationally infeasible to compute any function of any message block if any one of the pseudo-message blocks is unknown."
>
> -- Rivest, R. 1997. All or nothing encryption and the package transform. *Fast Software Encryption 1997.* 210-218.

When used with a conventional block cipher, an AONT appears to increase the cost of a brute-force attack by a factor which is the number of blocks in the message. Rivest also notes that the large effective block size can avoid ciphertext expanding chaining modes by using ECB mode on the large block. Also see huge block cipher advantages.

The Balanced Block Mixing (BBM) which I introduced to cryptography in my article: "Keyed Balanced Size-Preserving Block Mixing Transforms" (locally, or @: http://www.ciphersbyritter.com/NEWS/94031301.HTM) in early 1994 (three years before the Rivest publication), and then developed in a series of subsequent articles, apparently can be an especially fine example of an all-or-nothing transform.

Alphabet

In cryptography, the set of symbols under discussion. Also see universe, population and cardinal.

Alternative Hypothesis

In statistics, the statement formulated to be logically contrary to the null hypothesis. The alternative hypothesis H_1 includes every possible result other than the specific outcome specified in the null hypothesis.

The alternative hypothesis H_1 is also called the research hypothesis, and is logically identical to "NOT-H_0" or "H_0 is not true."

Amplifier

a component or device intended to sense a signal and produce a larger version of that signal. In general, any amplifying device is limited by available power, frequency response, and device maximums for voltage, current, and power dissipation. Also see: voltage divider.

Transistors are analog amplifiers which are basically linear over a reasonable range and so require DC power. In contrast, relays are classically mechanical devices with direct metal-to-metal moving connections, and so can handle generally higher power and AC current. The classic analog amplifier is an operational amplifier.

Unexpected oscillation can be indicated by:

- An unusually hot active device as felt by a **finger**.
- Unexpectedly high current flow as shown by a **multimeter**.
- Unexpected sounds as heard from a **speaker** monitoring the unit under test.
- Unexpected output signal as seen on an **oscilloscope**.
- Unexpected variation when touching various pins, as shown on a **multimeter** measuring the output signal, the output DC level or power supply current.
- Unexpected signal or variation as shown by an **RF voltage probe** connected to **multimeter**.
- Unexpected signal or variation as seen on a **wideband AC voltmeter**.

Oscillation occurs when:

- an amplified signal finds its way back to the amp input; AND
- the gain through the amplifier and feedback exceeds 1.0; AND
- the total phase shift around the feedback loop is 360 degrees.

To stop undesired oscillation:

- Increase isolation between input and output; OR
- Decrease gain; OR
- Change phase.

To Increase Isolation

- **Bypass the amplifier power pins.** All current from the output pin originally comes through a power pin. Signal at the output is necessarily reflected in signal at the power pins. Unless power lines are bypassed with significant storage capacitance, signal on the output will feed back to the input. Serious bypassing should be a part of normal use. The negative supply often needs to be cleaner than the positive supply.

- **Decouple the amplifier power pins.** Add series resistors to the power supply to form low-pass filters (in combination with the bypass capacitors), and thus decrease high-frequency feedback between stages via the supply.
- **Try moving input and output leads as far apart as possible.** If that improves the situation, the feedback path has at least been identified.
- **Try preventing capacitive coupling between input and output.** Interpose a conductive shield (try a finger) between in and out. If that helps, consider shielding the input and output lines. Or put in a permanent metallic shield like a piece of copper sheet or PC board material.
- **For units with single-ended input and output signals and high overall gain, try breaking the ground loop**. Try isolation transformers on input and/or output signal lines. Any stage with 40dB or more of resulting gain can be a particular problem when output signal returns through the same ground used by the input signal.
- **Consider redesigning for balanced input.** Balanced signal lines help prevent signal-line magnetic coupling and feedback.

To decrease gain:

- **Increase high frequency negative feedback.** Try using a small capacitor across the feedback resistor to reduce overall gain starting at a high frequency. (Make capacitive reactance equal to feedback resistance perhaps an octave above the highest desired frequency.)
- **High-pass filter the input.** A small series resistor and shunt capacitor form an RC filter that can take effect above the highest frequency of interest. Differential inputs each with their own input resistor can use a single shunt capacitor across the input pins.
- **Redesign for reduced stage gain.** Use another stage if more gain is necessary.

To change phase:

- **Identify the frequency-determining path.** Sometimes, touching various connections with fingers or a grounded capacitor will affect oscillation. If so, add components or change values to force a frequency which has insufficient gain to support oscillation.
- **For capacitive loads, try a small series output resistor.** Capacitive loads as small as 50pF, or even just cable capacitance, are notorious for causing operational amplifier problems. A small resistor on the output (e.g., the 50 or 75 ohm characteristic impedance of driven coax) before the cable can settle things down considerably.
- **Try damping transformer resonance with a load resistor.** Sometimes transformers can resonate with circuit capacitance.
- **Use a "snubber" series RC across resonant LC tanks.** For resonances at least 100x the highest desired frequency, Ridley Engineering recommends a resistor equal to the inductive (or capacitive, since it is resonant) reactance at the resonant frequency (R = XL = 2 Pi f L), with a series capacitor of C = 1 / (Pi f R). An analysis from Hagerman Technology suggests a resistor of R = SQRT(L / C) with a capacitor equal to 1 / (R f), which is about the same.

Amplitude

The signal level or magnitude, referenced to zero. For balanced bipolar signals, half of the peak-to-peak value; half of: the positive peak value less the negative peak value.

Anagram

To form a recognizable word, phrase or message by rearranging letters. This is the permutation of letters to achieve meaning. Also see: multiple anagramming.

Analog

Pertaining to continuous values. As opposed to digital or discrete quantities.

Analogy

From Greek geometry, "according to a ratio." To "draw a parallel" between situations which seem to have some property in common. A mode of argumentation in which a known relationship or pattern is applied to a new situation. A form of inductive reasoning which supports the creation of new testable consequences.

When two things are related by appropriate similarity in structure or function, we can infer that what is known about one thing also may apply to the other. Such an inference may or may not be true, but it can be examined and tested.

Analysis

1. The decomposition of a whole into component parts. As opposed to synthesis. Also see: hypothesis, argumentation, scientific method, proof. Also see: tree analysis, fault tree analysis and risk analysis. Also see: cryptanalysis.
2. The branch of mathematics concerned with functions and limits. The calculus.

AND

A Boolean logic function which is also mod 2 multiplication.

ANF

Algebraic Normal Form.

Anode

In an electronic circuit element, the end which sources electrons and accepts conventional current flow. The (-) end of a charged battery. The (+) end of a battery being charged. The P side of a PN semiconductor junction. As opposed to cathode.

Antecedent

In the study of logic, the if-clause of an if-then statement. Also see: consequent

AONT

All or Nothing Transform.

Arc

In an FSM, a path segment which does not repeat. As opposed to a cycle. In any FSM of finite size, every arc must eventually lead to a cycle.

Argument

1. In mathematics and programming, a variable which is an input to a function. A parameter. An independent variable.
2. In the study of logic, reasons that support a conclusion. Technically, a sequence of statements (premises) followed by a conclusion statement. An argument is *valid* only if the conclusion necessarily follows from the premises, and may be invalid:

- In **material content**, from misstatement of fact
- In **language wording**, from misuse of terms
- In **formal structure**, from misuse of logic.

Also see: fallacy, argumentation and burden of proof.

Argumentation

Argument applied to a case under discussion. Reasoned discourse to infer a conclusion. There are various classes:

- Analytic: separation into component parts; formal logic
- Synthetic: construction of a whole from component parts
- Dialectic: discussion and debate by questions and answers; the use of conflicting ideas to explore the

issue under discussion
- Rhetoric: techniques of argument presentation; the art of persuasive speech

Also see: claim, extraordinary claims, fallacies, Socratic Method, sophistry, spin and lie.

In *On Sophistical Refutations* (350 B.C.E.), Aristotle (384-322 B.C.E.) lists five goals for countering arguments:

1. Refutation -- to find fault in the argument or claim itself
2. Fallacy -- to find fault in the logic of the argument
3. Paradox -- to show the argument leads to logical contradiction
4. Solecism -- to cause emotional outburst
5. Babbling -- to cause repetition, lengthy explanation, or confusing distinctions

Refutation can occur in various ways. Disputing the evidence being used to support a claim can be considered a new claim and different evidence presented. However, disputing the reasoning itself requires only logic and typically no further evidence at all. See extraordinary claims.

Like cryptography, argumentation is war, and tricks abound when winning is the ultimate goal. But arguing to win is fundamentally unscientific, since learning occurs mainly when an error is found and recognized.

The first requirement of successful argumentation is to have a stated topic or thesis. Without a stated topic, an unscrupulous opponent can lead the argument to some apparently similar but more vulnerable issue, and few in the audience will notice. That is especially true when a topic is introduced casually, and then changed by the opponent in the very first response. Another approach for the opponent is to indignantly bring up and discuss in detail some supposed error on an irrelevant but apparently related topic. A clever topic change also may cause awkward repetition and babbling in the attempt to expose the change and reverse it. The correct response is to be aware enough to recognize the topic change immediately, and return to the original topic; to argue that the comments are off-topic is to introduce a new topic.

There is no way to *make* an opponent stay on-topic, and if they know they will lose on-topic, that actually may be impossible. Moreover, the opponent may pose various questions (on some new topic), and claim you are not being responsive, the discussion of that claim itself being a new topic. But if you want to take your topic to conclusion, you cannot follow an opponent who wants anything but that. (Also see spin.)

The second requirement of successful argumentation is to force the discussion to remain on the material content. If the original argument might be successful, an unscrupulous opponent may seek to divert the discussion to the appropriateness of, or bias in, the symbols or names used for the concept. Or the opponent may find and protest premises stated without mathematical precision. But a conventional argument need be neither mathematically complete nor mathematically precise to be valid. (This is the fallacy of accident.) The correct response is to point out that the comments are irrelevant and return to the material issue; to argue that the comments are wrong is to argue a changed topic.

How to Win

The goal of scientific argumentation is to improve knowledge and insight, not to anoint a "superior" contestant. Sadly, those willing to "win" with dishonesty generally do find an easily mislead audience.

Almost all on-line arguments are technically informal in the sense of depending upon context and definitions. The need for particular context generally leaves ample room to confuse the issue, even for someone who knows almost nothing about the topic.

If the proposed argument is basically unsound, that case can be won on its merits.

How to Attack

If the proposed argument is basically sound, but based on analogy, we need to realize that there are few really good analogies. Examine the analogy in detail and try various cases until one is found that is good in the analogy but bad in the proposed argument.

If the proposed argument is basically sound, one can win anyway by changing the topic and doing so in a smooth way the audience will not notice.

1. **Look for an invalid context.** Generally an argument is valid only within a particular context. Find a different context where the argument does not work and "disprove" it on that basis. Or,
2. **Look for a source of confusion.** Usually the argument will depend upon some particular words, either explicit or implied, that have multiple reasonable definitions. Choose one or more of those, and show how the argument fails with one of the other definitions. Typically, the opponent will reply and explain the proper context, which, of course, you already know.
3. **Look for an error.** In the process of describing the context, the opponent generally will increase the number of word dependencies, which is just more material to exploit. However, if the explanation can itself be interpreted as error, you can expose that error and then claim the opponent is not only wrong, but demonstrably incompetent.

How to Defend

Most responses carry a least a thin patina of respectability. However, many times a response is actually just the first sad shot in a verbal combat that seeks defeat and winning by deception. Unfortunately, it may be difficult to distinguish between mere ignorance and actual attack. It is thus important to actually examine the logic of any response.

Arity

In mathematics and programming, the number of parameters or arguments to a function or operator. Also see unary, binary, monadic and dyadic.

ASCII

A public code for converting between 7-bit values 0..127 (or 00..7f hex) and text characters. ASCII is an acronym for American Standard Code for Information Interchange. Also see Base-64.

DEC	HEX	CTRL	CMD	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
0	00	^@	NUL	32	20	SPC	64	40	@	96	60	'
1	01	^A	SOH	33	21	!	65	41	A	97	61	a
2	02	^B	STX	34	22	"	66	42	B	98	62	b
3	03	^C	ETX	35	23	#	67	43	C	99	63	c
4	04	^D	EOT	36	24	$	68	44	D	100	64	d
5	05	^E	ENQ	37	25	%	69	45	E	101	65	e
6	06	^F	ACK	38	26	&	70	46	F	102	66	f
7	07	^G	BEL	39	27	'	71	47	G	103	67	g
8	08	^H	BS	40	28	(	72	48	H	104	68	h
9	09	^I	HT	41	29	)	73	49	I	105	69	i
10	0a	^J	LF	42	2a	*	74	4a	J	106	6a	j
11	0b	^K	VT	43	2b	+	75	4b	K	107	6b	k
12	0c	^L	FF	44	2c	,	76	4c	L	108	6c	l
13	0d	^M	CR	45	2d	-	77	4d	M	109	6d	m
14	0e	^N	SO	46	2e	.	78	4e	N	110	6e	n
15	0f	^O	SI	47	2f	/	79	4f	O	111	6f	o
16	10	^P	DLE	48	30	0	80	50	P	112	70	p
17	11	^Q	DC1	49	31	1	81	51	Q	113	71	q

18	12	^R	DC2	50	32	2	82	52	R	114	72	r		
19	13	^S	DC3	51	33	3	83	53	S	115	73	s		
20	14	^T	DC4	52	34	4	84	54	T	116	74	t		
21	15	^U	NAK	53	35	5	85	55	U	117	75	u		
22	16	^V	SYN	54	36	6	86	56	V	118	76	v		
23	17	^W	ETB	55	37	7	87	57	W	119	77	w		
24	18	^X	CAN	56	38	8	88	58	X	120	78	x		
25	19	^Y	EM	57	39	9	89	59	Y	121	79	y		
26	1a	^Z	SUB	58	3a	:	90	5a	Z	122	7a	z		
27	1b	^[	ESC	59	3b	;	91	5b	[	123	7b	{		
28	1c	^\	FS	60	3c	<	92	5c	\	124	7c	|		
29	1d	^]	GS	61	3d	=	93	5d	]	125	7d	}		
30	1e	^^	RS	62	3e	>	94	5e	^	126	7e			
31	1f	^_	US	63	3f	?	95	5f	_	127	7f	DEL		

Associative

1. In abstract algebra, a dyadic operation in which two sequential operations on three arguments can first operate on either the first two or the last two arguments, producing the same result in either case: (a + b) + c = a + (b + c).
2. In algebra, the associative law for addition and multiplication. The algebraic law for evaluating the result of grouping terms or factors in different ways, as in conventional arithmetic:

```
a + (b + c) = (a + b) + c
a * (b * c) = (a * b) * c
```

Also see: commutative and distributive.

Assumption

In the study of logic, a statement accepted as true, but not proven. Typically a condition or premise.

In a mathematical proof, each and every assumption must be true for the proof result to be true. If the truth of any assumption is unknown, the proof is formally incomplete and the result has no meaning.

In practice, proofs have meaning only to the extent that each and every required assumption can be assured, including assumptions which may not be immediately apparent. In practical cryptography, while some assumptions possibly could be assured by the user, others could only be assured by the cipher designer, who must then be trusted, along with his company, the entire distribution path and so on. Even worse, still other assumptions may be *impossible* to assure in practice *by any means at all*, which makes any such proof useless for practical cryptography.

Asymmetric Cipher

A public key cipher.

Asynchronous

Not synchronous; not aligned in time. Digital logic systems in which signal changes are not related to an overall clock.

Also RS-232 and similar "serial port" signals, in which byte or character values are transferred bit-by-bit in bit-serial format. Since digital signals require both proper logic levels and proper timing to sense those levels, timing is established by the leading edge of a "start bit" sent at the start of each data byte. See asynchronous transmission.

Asynchronous Stream Cipher

A self-synchronizing stream cipher.

Asynchronous Transmission

The common RS-232 "serial port" signal, where data are sent on a single wire. This is complicated because a bit is not just a logic level, but also a *time* to sample that level. The necessary timing is established by the edge between a "high" stop bit or resting level, and a "low" start bit.

Transmit: The line rests "high." When a character is to be sent, a start bit or "low" level is sent for one bit-time. Then each data bit is sent, for one bit-time each, as are one or two stop or "high" level bit-times. Then, if no more data are ready for sending, the line just rests "high."

Receive: The line is normally "high." The instant the line goes "low" is the beginning of a start bit, and that establishes an origin for bit timing. Exactly 1.5 bit-times later, hopefully in the middle of the first data-bit time, the line level is sampled to record the first incoming bit. The second bit is recorded one bit-time later, and so on. When all bits have been recorded, the receiver sends the resulting character, all bits simultaneously, to a local register or FIFO queue for pickup. Note that all this implies that we know the format of the character with respect to bit time and number of bits.

Timing Accuracy: Everything depends upon both transmit and receive ends having approximately the same bit timing. The leading edge of the start bit temporarily synchronizes the receiver, even though the transmit and receive clock rates may be somewhat different. With 8-bit characters, the last data bit is sampled exactly 8.5 bit-times from the detected leading edge of the start bit. If the receive timing varies as much as +/- 0.5 bit in 8.5, the last bit will be sampled outside the correct bit time. So the total timing accuracy must be within +/- 5.8 percent, for all sources transmit and receive clock variation, including sampling delay in detecting the start bit. Nowadays this is easily achieved with cheap crystal oscillator clock modules and digital count logic.

Attack

General ways in which a cryptanalyst may try to "break" or penetrate the secrecy of a cipher. These are **not** algorithms; they are just *approaches* as a starting place for constructing specific algorithms.

In normal cryptanalysis we start out knowing plaintext, ciphertext, and cipher construction. The only thing left unknown is the key. A practical attack must recover the key. (Or perhaps we just know the ciphertext and the cipher, in which case a practical attack would recover plaintext.) Simply finding a distinguisher (showing that the cipher differs from the chosen model) is not, in itself, an attack. If an attack does not recover the key (or perhaps the particular key-selected internal state used in ciphering), it is not a real attack.

In cryptography, when someone says they have "an attack," the implication is that they have a *successful* attack (a break) and not just another failed attempt. It is obviously much easier to simply *claim* to have an attack than to actually analyze, innovate, build and test a working attack, which makes it necessary to back up such claims with evidence. Arrogant claims, with "proof left as an exercise for the student" or "read the literature" responses, deserve jeers instead of the cowed respect they often get.

A claim to have an attack can be justified by:

1. describing the process in such detail that others can understand it and could use it to break the cipher,
2. actually performing the attack in practice and showing the claimed results (e.g., finding the unknown key, given known plaintext), or
3. demonstrating the ability to do something fundamental which should be impossible (like finding several strings which each have the same cryptographic hash result).

Simply finding a distinguisher is not in itself sufficient to expose keying or plaintext as required of a real attack.

It is *not* sufficient to say: "My interpretation of the theory is that there must be a break, so the cipher is broken"; it is instead necessary to actually devise a process which recovers key or plaintext. Furthermore, there are many attacks which work against scaled-down tiny ciphers, but which do not scale up as valid attacks against the original large cipher: Just because we can solve newspaper-amusement ciphers (tiny versions of conventional block ciphers) does not imply that any real-size block ciphers are "broken." The process used to solve newspaper ciphers is not "an attack" on block ciphers in general.

Classically, attacks were neither named nor classified; there was just: "here is a cipher, and here is 'the' attack." (Many different attacks may be possible, but even one practical attack is sufficient to cause us to avoid that cipher.) And while this gradually developed into named attacks, there is no overall attack taxonomy. Currently, attacks are often classified by the information available to the attacker or *constraints* on the attack, and then by *strategies* which use the available information. Not only ciphers, but also cryptographic hash functions can be attacked, generally with very different strategies.

Information Constraints

We are to attack a cipher which enciphers plaintext into ciphertext or deciphers the opposite way, under control of a key. The available information necessarily constrains our attack strategies.

- **Ciphertext Only:** We have only ciphertext to work with. Sometimes the statistics of the ciphertext provide insight and can lead to a break.
- **Known Plaintext:** We have some, or even an extremely large amount, of plaintext and the associated ciphertext.
- **Defined Plaintext:** We can submit arbitrary messages to be ciphered and capture the resulting ciphertext. (Also Chosen Plaintext and Adaptive Chosen Plaintext.) (A subset of Known Plaintext.)
- **Defined Ciphertext:** We can submit arbitrary messages to be deciphered and see the resulting plaintext. (Also Chosen Ciphertext and Adaptive Chosen Ciphertext.)
- **Chosen Key:** We can specify a change in any particular key bit, or some other relationship between keys.
- **Timing:** We can measure the duration of ciphering operations and use that to reveal the key or data.
- **Fault Analysis:** We can induce random faults into the ciphering machinery, and use those to expose the key.
- **Man-in-the-Middle:** We can subvert the routing capabilities of a computer network, and pose as the other side to each of the communicators. (This is a key authentication attack on public key systems.)

Attack Strategies

The goal of an attack is to reveal some unknown plaintext, or the key (which will reveal the plaintext). An attack which succeeds with less effort than a brute-force search we call a break. An "academic" ("theoretical," "certificational") break may involve impractically large amounts of data or resources, yet still be called a "break" if the attack would be easier than brute force. (It is thus possible for a "broken" cipher to be much stronger than a cipher with a short key.) Sometimes the attack strategy is thought to be obvious, given a particular informational constraint, and is not further classified.

- **Brute Force** (also Exhaustive Key Search): Try to decipher ciphertext under every possible key until readable messages are produced. (Also "brute force" any searchable-size *part* of a cipher.)
- **Codebook** (the classic "codebreaking" approach): Collect a codebook of transformations between plaintext and ciphertext.
- **Differential Cryptanalysis:** Find a statistical correlation between key values and cipher transformations (typically the Exclusive-OR of text pairs), then use sufficient defined plaintext to develop the key.
- **Linear Cryptanalysis:** Find a linear approximation to the keyed S-boxes in a cipher, and use that to reveal the key.
- **Meet-in-the-Middle:** Given a two-level multiple encryption, search for the keys by collecting every possible result for enciphering a known plaintext under the first cipher, and deciphering the known ciphertext under the second cipher; then find the match.
- **Key Schedule:** Choose keys which produce known effects in different rounds.
- **Birthday** (usually a hash attack): Use the birthday paradox, the idea that it is much easier to find two values which match than it is to find a match to some particular value.

- **Formal Coding** (also Algebraic): From the cipher design, develop equations for the key in terms of known plaintext, then solve those equations.
- **Correlation**: In a stream cipher, distinguish between data and confusion, or between different confusion streams, from a statistical imbalance in a combiner.
- **Dictionary**: Form a list of the most-likely keys, then try those keys one-by-one (a way to improve brute force).
- **Replay**: Record and save some ciphertext blocks or messages (especially if the content is known), then re-send those blocks when useful.

Many attacks try to isolate unknown small components or aspects so they can be solved separately, a process known as divide and conquer. Also see: security.

Attack Tree

Supposedly, a formal way to analyze security in a cipher system. A method of cryptanalysis. Based on a threat model, and related to classic risk analysis as a specific kind of tree analysis.

A network in the form of a tree is used, with goals represented as nodes. Various possible ways to achieve a particular goal are represented as branches, which then can be taken as goals with their own branch nodes.

In cryptographic analysis, the idea is that the root node will represent the ultimate security we seek. Each path to the root then represents the accumulated effort needed to break that security. The problem is that it is typically *impossible* to assure that every alternative attack has been considered. And if some unconsidered approach *is* cheaper than any other, *that* becomes the true limit on security, despite not being present in the analysis.

Attack tree analysis does not tend to expose unconsidered attacks. Yet those are exactly the issues which carry the greatest cryptographic risk, because we can at least generally quantify the risk from known attacks. Since an attack tree cannot do what most needs to be done, it would seem to be a strange choice for cryptographic risk analysis. One could even argue that an attack tree is most useful as a formal aid in deluding naive executives and users.

Threat models basically concern *what* is to be protected, from *whom*, and for *how long*. But with ciphers, we seek to protect *all* our data, from *everyone*, *forever*. The extreme nature of these expectations is only part of what makes a conventional threat model unhelpful in understanding ciphering risks.

Cipher failure and exploitation happens in secret, so we cannot know how often it occurs and cannot develop a probability for it. Absent a probability of cipher failure, any attempt to understand ciphering risk is necessarily limited.

A more effective approach to system security is to build with understandable components. In component design, we can define exactly what each component permits. In component analysis, we can consider the security effects and expose the precise range of things each component allows. If none of the allowed things can cause a security problem, we will have no security problems. Components essentially become a custom language of system design which has no way of expressing security faults.

A component-based security design is far more restrictive and so is far more demanding than the conventional mode of hacking through a design and implementation. However, this design process provides a road map for real security, as opposed to belief in results from flawed analytical tools (like attack trees) and *ad hoc* analysis that simply cannot deliver the assurances we need.

The ability to analyze security must be designed into a system; it cannot be just added on to finished systems.

Augmented Repetitions

When sampling with replacement, eventually we again find some object or value which has been found before. We call such an occurrence a "repetition." A value found exactly twice is a double, or "2-rep"; a value found three times is a triple or "3-rep," and so on.

For a known population, the number of repetitions expected at each level has long been understood to be a binomial expression. But if we are sampling in an attempt to *establish* the effective size of an unknown population, we have two problems:

1. The binomial equations which predict expected repetitions do not reverse well to predict population, and
2. Exact repetitions discard information and so are less accurate than we would like. For example, if we have a double and then find another of that value, we now have a triple, and one *less* double. So if we are using doubles to predict population, the occurrence of a triple influences the predicted population in exactly the wrong direction.

Fortunately, there is an unexpected and apparently previously unknown combinatoric relationship between the population and the number of combinations of occurrences of repeated values. This allows us to convert any number of triples and higher *n*-reps to the number of 2-reps which have the same probability. So if we have a double, and then get another of the same value, we have a triple, which we can convert into three 2-reps. The total number of 2-reps from all repetitions (the *augmented 2-reps* value) is then used to predict population.

We can relate the number of samples *s* to the population *N* through the expected number of augmented doubles *Ead*:

```
     Ead(N,s) = s(s-1) / 2N .
```

This equation is **exact**, *provided* we interpret all the exact n-reps in terms of 2-reps. For example, a triple is interpreted as three doubles; the augmentation from 3-reps to 2-reps is (3 C 2) or 3. The augmented result is the sum of the contributions from all higher repetition levels:

```
            n     i
     ad = SUM   ( ) r[i] .
           i=2    2
```

where *ad* is the number of augmented doubles, and *r[i]* is the exact repetition count at the *i*-th level.

And this leads to an equation for predicting population:

```
     Nad(s,ad) = s(s-1) / 2 ad .
```

This predicts the population *Nad* as based on a mean value of augmented doubles *ad*. (For an example and comparison to various other methods, see the conversation:

- "Birthday Attack Calculations," locally, or @: http://www.ciphersbyritter.com/NEWS4/BIRTHDAY.HTM.)

Clearly, we expect the number of samples to be far larger than the number of augmented doubles, but an error in the augmented doubles *ad* should produce a proportionally similar error in the predicted population *Nad.* We typically develop *ad* to high precision by averaging the results of many large trials.

However, since the trials should have approximately a simple Poisson distribution (which has only a single parameter), we could be a bit more clever and fit the results to the expected distribution, thus perhaps developing a bit more accuracy. Also see population estimation, birthday attack, birthday paradox and entropy.

Also see

- my *Cryptologia* article: "Estimating Population from Repetitions in Accumulated Random Samples"

locally, or @: http://www.ciphersbyritter.com/ARTS/BIRTHDAY.HTM
- my early message on randomness testing locally, or @: http://www.ciphersbyritter.com/NEWS2/94080601.HTM
- my article: "Experimental Characterization of Recorded Noise," which calculates both entropy and augmented repetition log population for various noise sources locally, or @: http://www.ciphersbyritter.com/NOISE/NOISCHAR.HTM

Authentication

One of the objectives of cryptography: Typically, assurance that a message was sent by the purported author. This is message authentication and is sometimes assumed into message integrity.

It is possible to authenticate individual blocks, provided they are large enough to minimize the impact of adding extra authentication data in each block (see block code). One advantage lies in avoiding the alternative of buffering an entire message before it can be authenticated. That can be especially important for real-time (e.g., voice) communications.

Other forms of cryptographic authentication include key authentication for public keys, and source or user authentication, for the authorization to send the message in the first place.

Also see certification and certification authority.

Authenticating Block Cipher

A block cipher mechanism which inherently contains an authentication value or field. (Also see homophonic substitution.)

Authority

1. The right to demand acceptance.
2. A recognized, appointed, or certified expert responsible for statements or conclusions.

Science does not recognize mere authority as sufficient basis for a conclusion, but instead requires that facts and reasoning be exposed for review. The simple use of a name does not automatically create an ad verecundiam fallacy ("Appeal to Awe"). A name can identify a body of work giving the needed facts and the reasoning supporting a scientific conclusion.

Authority tends to hide the basis for drawing conclusions. Authority tends to avoid addressing complaints of false reasoning. Authority tends to hide reasoning and insists that a statement is correct simply because of who made it. A person repeating a conclusion from an authority often has no idea of the reasoning behind it, or what it really means with respect to limits or context.

In contrast, scientific thought exposes the factual basis and the reasoning, which tells us what the conclusion really means. Scientific thought is democratic and informs, and ideally gives everyone the same materials from which to draw factual conclusions, some of which may be new, strange and disconcerting, but nevertheless correct.

Autocorrelation

In statistics, the linear correlation of a sequence to itself ("auto"). The extent that any particular sample linearly predicts subsequent and prior samples (typically a different value for each positive or negative delay).

AUTODIN

AUTOmatic DIgital Network. A message switching network for the U.S. military, fielded in the late 1960's. Typically secure on each link between large secure facilities called ASC's (Automatic Switching Centers), with internal computer message switching or forwarding between links. Replaced by the MilNet packet switching network in the early 1990's.

Autokey

In cryptography, a stream cipher which generates the next keystream value from earlier elements elements in that same keystream ("auto"). Typically, the ciphertext output is fed back to modify the state of the random number generator producing the running key or confusion sequence.

Automorphism

In abstract algebra, the special case of an isomorphism where the field or group is mapped onto itself.

AUTOSEVOCOM

AUTOmatic SEcure VOice COMmunications.

Availability

One of the possible goals of cryptography. Keeping services open for use.

Avalanche

The observed property of a block cipher constructed in layers or "rounds" with respect to a tiny change in the input. The change of a single input bit generally produces multiple bit-changes after one round, many more bit-changes after another round, until, eventually, about half of the block will change. An analogy is drawn to an avalanche in snow, where a small initial effect can lead to a dramatic result. As originally described by Feistel:

> "As the input moves through successive layers the pattern of 1's generated is amplified and results in an unpredictable avalanche. In the end the final output will have, on average, half 0's and half 1's . . . ." [p.22] -- Feistel, H. 1973. Cryptography and Computer Privacy. *Scientific American.* 228(5):15-23.

Also see mixing, diffusion, overall diffusion, strict avalanche criterion, complete, S-box. Also see the bit changes section of the "Binomial and Poisson Statistics Functions in JavaScript," locally, or @: http://www.ciphersbyritter.com/JAVASCRP/BINOMPOI.HTM#BitChanges.

Avalanche Effect

The result of avalanche. As described by Webster and Tavares:

> "For a given transformation to exhibit the avalanche effect, an average of one half of the output bits should change whenever a single input bit is complemented." [p.523] -- Webster, A. and S. Tavares. 1985. On the Design of S-Boxes. *Advances in Cryptology -- CRYPTO '85.* 523-534.

Also see the bit changes section of the "Binomial and Poisson Statistics Functions in JavaScript" page (locally, or @: http://www.ciphersbyritter.com/JAVASCRP/BINOMPOI.HTM#BitChanges).

Avalanche Multiplication

In semiconductor electronics, the dominant reverse-voltage breakdown mode in so-called "zener" diodes and other PN junctions with breakdown voltages over 6 or 8 volts. A source of shot noise and additional noise from avalanche which can be much greater than shot noise.

In normal junctions, the space-charge region (depletion region) between P and N materials is fairly broad, so the extreme fields found in Zener breakdown do not occur. However, a combination of applied voltage, temperature, and random motion may cause a covalent bond to break anyway, in a manner similar to normal diode leakage. When a breakdown does occur, the charge carrier is attracted by the opposing potential and drops through the space-charge region, periodically interacting with covalent bonds there. When the field is sufficiently high, a falling charge carrier may build up enough energy to break another carrier free when it hits. Then both the original and resulting carriers continue to accelerate through the space-charge region, each possibly hitting and breaking many other bonds. The result is a growing avalanche of carriers produced by each single breakdown. The avalanche effect can be seen as a form of amplification and can be huge, for example, 10**8.

In a series of almost forgotten semiconductor physics research papers from the 1950's and 1960's, avalanching breakdown was shown to consist of a multitude of "microplasma" events of perhaps 20uA each. These events are not completely independent, but instead interact, but also have some apparently random component, probably thermal. At least some of the microplasma events seem to have negative dynamic resistance and function like tiny like neon bulbs (and may even emit light). One implication if this is an ability of some avalanching "zener" diodes to directly support small, unsuspected oscillations. A series of very extensive discussions on sci.electronics.design in 1997 (search: "zener oscillation") give experimental details. Both LC tank oscillation and RC relaxation oscillation were demonstrated in practice. Thus, avalanche multiplication, often assumed to be unquestionably "quantum random," actually may have a disturbing amount of predictable structure. True Zener breakdown does not appear to have the same problems, nor does thermal noise, as far as we know. Unfortunately, these "purer" sources may be much smaller than noise from avalanche multiplication.

In contrast to Zener breakdown, which has a negative temperature coefficient, avalanche multiplication has a positive temperature coefficient, like most resistances or conductors. Presumably this is due to heat causing increased activity in the crystal lattice, thus preventing electrons from falling as far before interacting, thus reducing the probability of breaking another bond, and reducing the amplification. In junctions that break down at about 6 volts the temperature effects tend to cancel. Also see: "Random Electrical Noise: A Literature Survey" (locally, or @: http://www.ciphersbyritter.com/RES/NOISE.HTM).

Axiom

In the study of logic and argument, a statement (premise) assumed true without proof. A postulate.

Back Door

A cipher design fault, planned or accidental, which allows the apparent strength of the design to be easily avoided by those who know the trick. When the design background of a cipher is kept secret, a back door is often suspected. Also see: trap door.

Balance

1. Equal on each side. The same number of each kind of symbol. Also see bit balance.
2. A term used in S-box and Boolean function analysis. As described by Lloyd:

> "A function is balanced if, when all input vectors are equally likely, then all output vectors are equally likely." -- Lloyd, S. 1990. Properties of binary functions. *Advances in Cryptology -- EUROCRYPT '90.* 124-139.

There is some desire to generalize this definition to describe multiple-input functions. (Is a dyadic function "balanced" if, for one value on the first input, all output values can be produced, but for another value on the first input, only *some* output values are possible?) Presumably a two-input balanced function would be balanced for either input fixed at any value, which would essentially be a Latin square or a Latin square combiner. Also see Balanced Block Mixing. As opposed to bias. Also see Ideal Secrecy and Perfect Secrecy.

Balance is a pervasive requirement in many areas of cryptography; for example:

- Balance is the good random sequence which has no bias.
- Balance is another way of saying random sampling (as in statistics).
- Balance is the uniform distribution we strive for in keying (all keys being equally probable).
- Balance is the creation or selection of a particular object (e.g., a substitution table) from among all possibilities (or perhaps just those with a particular structure, e.g., orthogonal Latin squares).
- Balance is what we expect from whitening (e.g., CBC mode). Making plaintext blocks equally probable helps to hold off codebook attack.

- Balance is what we expect to see in ciphertext values.
- Balance is what we expect to see in ciphertext bit values.
- Balance is what is so good about exclusive-OR and Latin square combining.
- Balanced plaintext produces Shannon Ideal Secrecy.
- Balanced combining between plaintext and key into ciphertext produces Shannon Perfect Secrecy.

Balanced Block Mixer

A process or any implementation (for example, hardware, computer software, hybrids, or the like) for performing Balanced Block Mixing.

Balanced Block Mixing

(BBM). Balanced Block Mixing. The ciphering concept described in U.S. Patent 5,623,549. See

- U.S. Patent 5,623,549 locally or @: http://www.ciphersbyritter.com/PATS/MIXPATA.HTM
- the BBM articles, locally or @: http://www.ciphersbyritter.com/index.html#BBMTech

A mechanism for mixing large block values like those used in block ciphers. A BBM is balanced to avoid leaking information, and is effective in just a single pass, thus avoiding the need for repeated rounds and added hardware. A BBM has no data expansion. A BBM supports the construction of scalable ciphers with large blocks, and can be more efficient, more flexible, and more useful than conventional fixed and smaller designs. (See: ideal mixing, Mixing Cipher, Mixing Cipher design strategy and also the BBM articles, locally, or @: http://www.ciphersbyritter.com/index.html#BBMTech).

Technically, a Balanced Block Mixer is an *m*-input-port *m*-output-port mechanism with various properties:

1. The overall mapping is one-to-one and invertible: Every possible input value (over all ports) to the mixer produces a different output value (including all ports), and every possible output value is produced by a different input value;
2. Each output port is a function of every input port;
3. Any change to any *one* of the input ports will produce a change to *every* output port;
4. Stepping any *one* input port through all possible values (while keeping the other input ports fixed) will step *every* output port through all possible values.

The inverse mixing behaves similarly. Say, for example, we are mixing 64 bytes of message into 64 bytes of result: If we know 63 of the result bytes, we can step through the values of the 64th byte, and get 256 different messages, each of which will produce the 63 bytes we know (a homophonic sort of situation). If the actual messages are random-like and evenly distributed, it will be difficult to know which particular message is implied. The amount of uncertainty we have in the result is reflected in the amount of uncertainty we have about the message.

The Basic Balanced Block Mixer

The basic Balanced Block Mixer is a pair of orthogonal Latin squares. The two input ports affect the rows and columns of both squares, with the selected result in each square being the two output ports. For example, here is a tiny nonlinear "2-bit" or "order 4" BBM:

```
   3 1 2 0   0 3 2 1       30  13  22  01
   0 2 1 3   2 1 0 3   =   02  21  10  33
   1 3 0 2   1 2 3 0       11  32  03  20
   2 0 3 1   3 0 1 2       23  00  31  12
```

Suppose we wish to mix (1,3); 1 selects the second row up in both squares, and 3 selects the rightmost column, thus selecting (2,0) as the output. Since there is only one occurrence of (2,0) among all entry pairs, this discrete mixing function is reversible, as well as being balanced on both inputs.

In practice, we would probably want to use at least order 16, which can be efficiently stored as an ordinary 256-byte "8-bit" substitution table, one with a particular oLs structure in the data.

Scalable Linear Balanced Block Mixers

One way to use the BBM mixing concept is to develop **linear** equations for oLs mixing for scaling to various sizes (see my article: Fencing and Mixing Ciphers from 1996 Jan 16). We can do that in the finite field of mod-2 polynomials with an irreducible modulus. So we can easily have similar mixers of 16, 32, 64, 128 and 256 bit port widths, and so on. By using multiple mixers of different size in various connections, we can easily mix blocks of size compatible to existing ciphers, and much larger.

Explicit Nonlinear Balanced Block Mixers

A usually better way to use the BBM mixing concept is to develop small, **nonlinear** and keyed oLs's for use in FFT-like patterns with 2^n ports. It is easy to construct keyed nonlinear orthogonal pairs of Latin squares of arbitrary 4n order as I describe in my articles:

- "Orthogonal Latin Squares, Nonlinear Balanced Block Mixers, and Mixing Ciphers," locally, or @: http://www.ciphersbyritter.com/ARTS/NONLBBM.HTM
- "Practical Latin Square Combiners," locally, or @: http://www.ciphersbyritter.com/ARTS/PRACTLAT.HTM

In any FFT-style structure, there is exactly one "path" from any input to any output, and "cancellation" cannot occur. Thus, we can **guarantee** that *any* change to any one input must "affect" *each and every* output. Similarly, each input is equally represented in each output, which is ideal mixing. The resulting wide ideal mixing structure, using small BBM tables as each butterfly operation, is *itself* a BBM, and is dynamically scalable to virtually arbitrary size.

Scalable Mixing Advantages

Mixing has long been a problem in block ciphers. The difficulty of mixing wide block values is one reason most conventional block ciphers are small. But having a small block means that there is not much room to add features like:

- dynamic keying,
- homophonic operation, and
- authentication,

on each block (also see block coding and huge block cipher advantages). Per-block authentication, for example, allows blocks to be used in real time, or delivered out of order, without first buffering and authenticating the entire message.

Large blocks also have room to hold sufficient uniqueness to support electronic codebook mode, which is not normally appropriate for block ciphers. Large blocks in ECB mode can support secure ciphering without ciphertext expansion, a goal which is very hard to reach in other ways.

When a BBM is implemented in software, the exact same unchanged routine can handle both wide mixing for real operation and narrow "toy" mixing for thorough experimental testing. This supports both scalable operation, and exhaustive testing of *the exact code* used in actual operation.

In hardware, BBM block throughput or block rate can be *independent of block size*. Wide blocks can be mixed in the same time as narrow blocks by pipelining each sub-layer of the mixing. That of course makes large blocks far faster per byte than small ones.

Also see All or Nothing Transform, Mixing Cipher, Dynamic Substitution Combiner, and Variable Size Block Cipher.

Also see *some* of the development sequence:

- My article: "Keyed Balanced Size-Preserving Block Mixing Transforms" (March 12, 1994), locally, or @: http://www.ciphersbyritter.com/NEWS/94031301.HTM.
- My article: "Fenced DES" (April 17, 1994), locally, or @: http://www.ciphersbyritter.com/NEWS/94042901.HTM
- My article: "Fencing and Mixing Ciphers" (16 Jan 1996). locally, or @: http://www.ciphersbyritter.com/NEWS/96011601.HTM.
- My patent: "United States Patent 5,623,549: Cipher Mechanisms with Fencing and Balanced Block Mixing" (Apr. 22, 1997), locally, or @: http://www.ciphersbyritter.com/PATS/MIXPATA.HTM.
- My article: "Measured Nonlinearity in Mixing Constructions" (1997-12-21), locally, or @: http://www.ciphersbyritter.com/ARTS/MIXNONLI.HTM
- My article: "Practical Latin Square Combiners" (1998-09-15), locally, or @: http://www.ciphersbyritter.com/ARTS/PRACTLAT.HTM
- My article: "Orthogonal Latin Squares, Nonlinear Balanced Block Mixers, and Mixing Ciphers" (1998-09-22), locally, or @: http://www.ciphersbyritter.com/ARTS/NONLBBM.HTM.

Balanced Combiner

In the context of cryptography, a combiner mixes two input values into a result value. A balanced combiner must provide a balanced relationship between each input and the result.

In a *statically-balanced* combiner, any particular result value can be produced by any value on one input, simply by selecting some appropriate value for the other input. In this way, knowledge of only the output value provides no information -- not even statistical information -- about either input.

The common examples of cryptographic combiner, including byte exclusive-OR (mod 2 polynomial addition), byte addition (integer addition mod 256), or other "additive" combining, are perfectly balanced. Unfortunately, these simple combiners are also very weak, being inherently linear and without internal state.

A Latin square combiner is an example of a statically-balanced reversible nonlinear combiner with massive internal state. A Dynamic Substitution Combiner is an example of a dynamically or statistically-balanced reversible nonlinear combiner with substantial internal state.

Balanced Line

In electronics, typically an interconnecting cable with two conductors having an exactly opposite or *symmetric* signal on each. More specifically, a two-wire signal interconnection where each wire has the same impedance to ground or any other conductor. In contrast to *unbalanced* line (like coaxial cable), where one of the conductors (the shield) has a low impedance to ground. In all cases, two conductors are required to transport a loop of current carrying a signal.

Each conductor of a balanced line systems should have similar driver output impedances (ideally low), similar wire effects, and similar receiver termination impedances (ideally high). At audio frequencies cables are not transmission lines, so "cable impedance" is not an issue, and the differential receiver need not match either the cable or the driver. When each wire has a similar impedance to ground, external magnetic and electrostatic fields should act on them similarly, producing a common effect on each wire which can "cancel out."

A transformer winding makes a good balanced line driver. In contrast, operational amplifier circuits with direct outputs probably will have only roughly-similar output impedances. Output resistors (e.g., 100 ohms) typically isolate each op amp output from the cable, and any difference will represent driver imbalance to external noise. After being transported, the differential mode signal is taken between the two conductors, thus ignoring common mode noise. A transformer winding makes a good differential receiver and also provides ground loop isolation. Operational amplifier receivers need a common-mode-rejection null adjustment for best performance.

At audio frequencies, the main advantage of balanced line is rejection of AC hum and related power noises. This can be achieved by driving only one line with the desired audio signal, provided both lines are terminated similarly both in the driver and receiver.

At radio frequencies, balanced line also minimizes undesired signal radiation. When the current changes in each wire are equal but opposite, they radiate "out of phase," resulting in cancellation. This is especially useful in TEMPEST, but does require that both lines be actively driven.

Base-64

A public code for converting between 6-bit values 0..63 (or 00..3f hex) and text symbols accepted by most computers. Also see ASCII.

```
        0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f

   0    A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P
   1    Q  R  S  T  U  V  W  X  Y  Z  a  b  c  d  e  f
   2    g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v
   3    w  x  y  z  0  1  2  3  4  5  6  7  8  9  +  /

   use "=" for padding
```

Base Spreading Resistance

In a bipolar transistor, the internal resistance between the base lead and the functioning base element. Often denoted R_{bb}' or r_b' and having a typical value of perhaps 100 ohms. A major source of thermal noise in a bipolar transistor.

A bipolar transistor is made by diffusing impurities into a thin slice of extremely pure single-crystal semiconductor, such as silicon. Typically, the collector contact is made at the top surface, and the emitter contact is made on the bottom. The base element is essentially a thin film situated between the collector and emitter plates. The base current must flow on the film, which is naturally more resistive than the other thicker elements.

BBM

Balanced Block Mixing.

BBS

Bulletin Board System.

BB&S

Blum, Blum and Shub. A famous article developing a "provably secure" RNG.

Blum, L., M. Blum and M. Shub. 1983. Comparison of Two Pseudo-Random Number Generators. *Advances in Cryptology: CRYPTO '82 Proceedings.* Plenum Press: New York. 61-78.

Blum, L., M. Blum and M. Shub. 1986. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing.* 15:364-383.

The BB&S RNG is basically a simple squaring of the current seed (x) modulo a composite (N) composed of two primes (P,Q) of public key size. Primes P and Q must both be congruent to 3 mod 4, but the BB&S articles say that P and Q also must be special primes. The special primes construction apparently has the advantage of controlling the cycle structure of the system, and is part of the BB&S design in the original articles. Unfortunately, the special primes construction generally is not presented in current texts. Instead the texts deceptively describe a simplified version which they nevertheless call BB&S. Readers who do not study referenced articles will assume they know what BB&S said, but they are only partly correct.

Unlike more common RNG's, the BB&S construction is *not* maximal length, but instead defines systems with multiple cycles, including degenerate, short and long cycles. With large integer factors, state values on short cycles are very rare, but do exist. Short cycles are dangerous with any RNG, because when a RNG sequence begins to repeat, it has just become predictable, despite any theorems to the contrary. Consequently, if we key BB&S by choosing *x[0]* at random, we may unknowingly select a weak short cycle (a weak key), which would make the sequence predictable as soon as the cycle starts to repeat.

The original BB&S articles lay out the technology to compute the exact length of a long-enough cycle in the BB&S system. Since it can be much easier to verify cycle length than to actually traverse the cycle, this is a practical way to verify that *x[0]* selects a long-enough cycle. Values of *x[0]* can be chosen and checked until a long cycle is selected. Modern cryptography insists, to the point of strident intimidation, that such verification is unnecessary. However, the original authors apparently thought it was important enough to include in their work.

The real issue here is not the exposure of a particular weakness in BB&S, since choosing *x[0]* on a short cycle is very unlikely. But "unlikely" is not the same as "impossible." And if the design goal is to eliminate every known weakness, even extensive math which concludes "that particular weakness is too unlikely to worry about" is beside the point: "unlikely" does not satisfy the goal. Mathematics does not get to impose goals on designers or users.

BB&S is said to be "proven secure" in the sense that *if* factoring is hard, *then* the sequence is unpredictable. And many people do think that factoring large composites of public key size is hard. Yet when a short cycle is selected and used, BB&S is obviously insecure, and that is a direct contradiction for anyone who imagines that "proven secure" applies to them. Just knowing the length of a cycle (by finding sequence repetition) should be enough to expose the factors. This is also evidence that the assumption that factoring is hard is not universally true. Of course, we *already know* that factoring is *not* hard -- when we give away the factors. And giving away the factors is pretty much what we do if we allow ourselves to select, use and traverse a short cycle. For other comments on the proof, see the sci.crypt conversations:

- "Randomness Tests; Blum, Blum & Shub," locally, or @: http://www.ciphersbyritter.com/NEWS2/TESTSBBS.HTM#BB&S
- "Is a BB&S System 'Proven Secure'?," locally, or @: http://www.ciphersbyritter.com/NEWS5/SUBGPBBS.HTM
- "Proof and Strength in the BB&S RNG," locally, or @: http://www.ciphersbyritter.com/NEWS5/OTPBBS.HTM

The advantage of the special primes construction apparently is that all "short" (but not degenerate) cycles are "long enough" for use. Thus, we can simply choose *x[0]* at random, and then easily test that it is not on a degenerate cycle. (Just get some *x[0]*, step *x[0]* to *x[1]*, save *x[1]*, step *x[1]* to *x[2]*, then compare *x[2]* to *x[1]* and if they are the same, start over.) The result is a *guarantee* that the selected cycle is "long enough" for use. See the sci.crypt discussion:

- "Simple Seed Selection in BB&S," locally, or @: http://www.ciphersbyritter.com/NEWS6/BBS.HTM

Note that I now recommend taking *two* steps before checking for a degenerate cycle.

It is sometimes said that the special primes construction adds nothing to BB&S, but that really depends more on the goals of the cipher designer than the math. Since BB&S is very slow in comparison to other RNG's, someone selecting BB&S clearly has decided to pay a heavy toll with the expectation of getting an RNG which is "proven secure" in practice. (That actually misrepresents the BB&S proof, which apparently allows weakness to exist provided it is not an easy way to factor N.) The obvious *goal* is to get a practical RNG which has no known weakness at all.

No mere proof can protect us when we ourselves choose and use a weak key, even if doing that is shown to be statistically very unlikely. And if we do use a weak key, the "proven secure" RNG is clearly insecure, which surely contradicts the motive for using BB&S in the first place. In contrast, simply by using the special primes construction and checking for degenerate cycles, weak keys can be *eliminated*, at modest expense. Eliminating a known possibility of weakness, even if that possibility is very small, seems entirely consistent with the *goal* of achieving a practical RNG with no *known* weakness, even if the result is *not* an RNG proven to have absolutely no weakness at all.

Some would say that even the special primes construction is overkill, but without it the so-called "proof of strength" becomes a mere wish or hope that a short cycle is not being used, and I see that as a contradiction. It also might be a cautionary tale as to what mathematical cryptography currently accepts as proof, and as to what such "proof" means in practical use. For other examples of failure in the current cryptographic wisdom, see one time pad, and AES (as an example of the size of the permutation family in real conventional block ciphers), and, of course, old wives' tale. Also see algorithm.

Bel

The base-10 logarithm of the ratio of two power values (which is also the same as the difference between the log of each power value). The basis for the more-common term decibel: One bel equals 10 decibels.

Belief

A subjective conviction in the truth of a proposition. Mere belief is in contrast to a hypothesis, which states some expectation of fact and so may be *verified* or *falsified* by observation and comparison in an experiment or just reality itself. Also see dogma.

Ordinarily we distinguish mere belief from proven truth, belief thus implying something less than conclusive evidence. In this sense, to *believe* is to be willing to accept unproven or even unprovable assumptions, such as having faith, or trusting in some machine or property. One issue is whether such assumptions or trust is reasonable in the real world.

Limiting what one can or should *believe* seems intertwined with freedom of speech and individual rights: Surely, anyone can believe what they want. However, to the extent that we have real responsibilities to others and society at large, unfounded belief *can not* uphold those obligations. In a seminal essay called "The Ethics of Belief" (circa 1877 and reprinted on the web), William Clifford shows how unfounded belief is insufficient support for decisions of life and death and reputation. Many of us would extend that to business planning (the recent *Waltzing with Bears* by DeMarco and Lister (see risk management) reprints the first section of "The Ethics of Belief" as an appendix), as well as scientific discussions and claims.

In that point of view, claiming something is true, when one has not investigated the topic and does not know, is ethically wrong, even if the claim it turns out (by pure dumb luck) to be correct. It is not enough to claim something and hope it works out; it is instead necessary to know that the claim is correct before making the claim. The ethical requirement is to have performed an investigation sufficient to expect to know one way or another, and come to a rationally supportable conclusion. While not rising to the level of known fact, belief is something on which reputation rests. Being wrong thus has consequences to reputation, provided the error is in the essence and not mere correctable detail.

This idea of requiring substantial investigation to come to a belief may seem to conflict with the scientific method, in that a scientist seemingly makes a mere claim, which generally stands until shown false. But in reality we expect that claim to be something beyond "mere." We demand that a scientific investigator have put sufficient professional effort into a conclusion before using a scientific podium to spout off. The investigation is what provides an ethical basis for belief, which still may be wrong or (more likely) incomplete.

For example, scientific publication does not mean that all of science supports the described conclusions, which are still just claims made by particular scientists. Showing (not necessarily proving) a claim to be wrong is *part* *of the process* of science, not unwarranted intrusion. Showing someone wrong in this context naturally *affects* reputation, but rarely results in absolute ruin.

The process of experimentation involves making "claims," often to be disproven, but those are clearly labeled hypotheses for experiment, not conclusions for use by others.

In contrast, when we have conclusive evidence of truth we have *knowledge* and *fact* instead of belief. Facts do not require belief, nor do they respond to voting or authority. Clearly, science depends upon knowledge and fact, not personal beliefs, and it is crucial to know the difference. Also see: scientific method, extraordinary claims and rhetoric.

Bent Function

A Boolean function whose fast Walsh transform has the same absolute value in each term (except, possibly, the zeroth). This means that the bent function has the same distance from every possible affine Boolean function.

We can do FWT's in "the bottom panel" at the end of my: "Active Boolean Function Nonlinearity Measurement in JavaScript" page, locally, or @: http://www.ciphersbyritter.com/JAVASCRP/NONLMEAS.HTM.

Here is every bent sequence of length 4, first in {0,1} notation, then in {1,-1} notation, with their FWT results:

```
   bent {0,1}       FWT           bent {1,-1}       FWT

   0  0  0  1    1 -1 -1  1         1  1  1 -1    2  2  2 -2
   0  0  1  0    1  1 -1 -1         1  1 -1  1    2 -2  2  2
   0  1  0  0    1 -1  1 -1         1 -1  1  1    2  2 -2  2
   1  0  0  0    1  1  1  1        -1  1  1  1    2 -2 -2 -2
   1  1  1  0    3  1  1 -1        -1 -1 -1  1   -2 -2 -2  2
   1  1  0  1    3 -1  1  1        -1 -1  1 -1   -2  2 -2  2
   1  0  1  1    3  1 -1  1        -1  1 -1 -1   -2 -2  2 -2
   0  1  1  1    3 -1 -1 -1         1 -1 -1 -1   -2  2  2  2
```

These sequences, like all true bent sequences, are **not** balanced. Literature references on this point include:

> "Let $Q_n = \{1,-1\}^n$. The defining property of a bent sequence x in Q_n is that the Hadamard transform of x has constant magnitude."
> "Let y be a bent sequence over $\{0,-1\}^n$. . . . "The Hamming weight of y is 2^{2k-1} (+ or -) 2^{k-1}."
> -- Adams, C. and S. Tavares. 1990. Generating and counting binary bent sequences. *IEEE Transactions on Information Theory.* IT-36(5):1170-1173.
>
> "Bent functions, except for the fact that they are never balanced, exhibit ideal cryptographic properties."
> -- Chee, S., S. Lee, K. Kim. 1994. Semi-bent functions. *Advances in Cryptology -- ASIACRYPT '94* 107-118.
>
> ". . . it has often to be considered as a defect from a cryptographic point of view that bent functions are necessarily non-balanced."
> -- Dobbertin, H. 1994. Construction of Bent Functions and Balanced Boolean Functions with High Nonlinearity. *K.U. Leuven Workshop on Cryptographic Algorithms (Fast Software Encryption)*. 61-74.
>
> "**Example 23 (Bent Functions Are Not Balanced)** . . . ."
> -- Seberry, J. and X. Zhang. "Hadamard Matrices, Bent Functions and Cryptography." The University of Wollongong. November 23, 1995.

The zeroth element of the {0,1} FWT is the number of 1's in the sequence.

Here are some bent sequences of length 16:

```
   bent {0,1}    0 1 0 0   0 1 0 0   1 1 0 1   0 0 1 0
    FWT          6,-2,2,-2,2,-2,2,2,-2,-2,2,-2,-2,2,-2,-2
   bent {1,-1}   1 -1 1 1   1 -1 1 1   -1 -1 1 -1   1 1 -1 1
    FWT          4,4,-4,4,-4,4,-4,-4,4,4,-4,4,4,-4,4,4

   bent {0,1}    0 0 1 0   0 1 0 0   1 0 0 0   1 1 1 0
    FWT          6,2,2,-2,-2,2,-2,2,-2,-2,-2,-2,2,2,-2,-2
   bent {1,-1}   1 1 -1 1   1 -1 1 1   -1 1 1 1   -1 -1 -1 1
    FWT          4,-4,-4,4,4,-4,4,-4,4,4,4,4,-4,-4,4,4
```

Bent sequences are said to have the highest possible uniform nonlinearity. But, to put this in perspective, recall that we *expect* a random sequence of 16 bits to have 8 bits different from any particular sequence, linear or otherwise. That is also the *maximum possible* nonlinearity, and here we actually *get* a nonlinearity of 6.

There are various more or less complex constructions for these sequences. In most cryptographic uses, bent sequences are modified slightly to achieve balance.

Berlekamp-Massey

The algorithm used to find the shortest LFSR which can reproduce a given sequence. This length is the linear complexity of the sequence.

Massey, J. 1969. Shift-Register Synthesis and BCH Decoding. *IEEE Transactions on Information Theory*. IT-15(1):122-127.

Bernoulli Trials

In statistics, observations or sampling with replacement which has exactly two possible outcomes, typically called "success" and "failure." Bernoulli trials have these characteristics:

- Each trial is independent,
- Each outcome is determined only by chance, and
- The probability of success is fixed.

Bernoulli trials have a Binomial distribution.

Bias

1. In argumentation, favoring one side over another, typically by making the burden of proof more difficult for one side. Also see extraordinary claims.
2. In statistics, results consistently some amount displaced from the theoretical or practical expectation. This is a clear indication that the measured process has not been completely understood or modeled.
3. In electronics, typically a static or DC voltage or current. A bias voltage is necessary to condition some forms of transducer (see, for example, Geiger-Mueller tube). However, the most common use is to keep some active device (typically a transistor) "partly on," so that it can amplify or respond to both positive and negative parts of an AC signal.

Transistor biasing is trickier than it might seem from knowing the simple purpose of keeping the device "partly on":

- There will be a wide range of different input bias solutions to cover a wide range of no-signal output currents, which also implies a wide range of output resistor values.
- Output bias is normally controlled by the same base or gate used for input signal, which to some extent compromises the distinction between the two different functions.
- Transistors apparently cannot be manufactured to close specifications, so different devices, even of

exactly the same type, may need significantly different input bias values to achieve the same output results.

One common biasing approach is to place a particular DC voltage on the base or gate, and a resistor in the emitter or source lead. Transistor action then tends to increase current until the emitter or source has a voltage related to the base or gate, a form of negative feedback. This sets the output bias current, which with a particular pull-up resistor sets a desired output voltage. One difficulty with this approach is that it demands an input signal with lower impedance than the biasing, so that the AC signal will dominate. Another issue is that the emitter or source resistor will use some of the available voltage simply to establish bias, voltage which then is not available across the device for AC signals. (Also see transistor self-bias.)

Bijection

Functions *f*: X -> Y and the inverse f^{-1}: Y -> X, where, for all *x* in *X*:

```
   if f(x) = y then f⁻¹(y) = x.
```

With a bijection an inverse always exists. (Contrast with: involution.)

A bijection on bit-strings, F: $\{0,1\}^n$ -> $\{0,1\}^n$ is one way to describe the emulated, huge, simple substitution of a conventional block cipher. In such a cipher, the encryption function is a key-selected permutation of all possible input or block values, which is the input alphabet. See: bijective and mapping.

Bijective

A bijection. A mapping *f*: X -> Y which is both injective (one-to-one) and surjective (onto). For each unique *x* in *X* there is corresponding unique *y* in *Y*. An invertible mapping. See: bijection, permutation and simple substitution.

Bijective Compression

Apparently, a form of data compression in which arbitrary values or arbitrary strings can be decompressed into seemingly grammatical text. More generally, the ability to decompress random blocks or strings into data with structure and statistics similar to that expected from plaintext. The phrase "Bijective Compression" should be taken as a name or term of art, and not a mathematical description.

Making random data decompress into language text (necessarily also random in some way) would seem to be difficult. Different classes of plaintext, such as language, database files, program code, or whatever, probably require different compressors or at least different compression models. With respect to language text, such a compressor should decompress random strings into spaced correct words or "word salad." That should complicate attempts to automatically distinguish the original message or block from among other possibilities.

Should bijective compression actually be possible and practical, the significance would be massive. Computerized attacks can succeed only if a correct deciphering can be recognized automatically. When incorrect decipherings have structure which is close to plaintext, a computer may not be able to distinguish them from success. If humans skill is needed to read and judge the result of thousands or millions of brute-force attempts, traversing a keyspace may take tens of millions of times longer than simple computer scanning. Making an attack millions of times harder than it was before could be the difference between complete practical security and almost no security at all. Holding an attack loop down to human reading speeds could produce a massive increase in practical strength.

Binary

From the Latin for "dual" or "pair."
1. Dominantly used to indicate "base 2": The numerical representation in which each digit has an alphabet of only two symbols: 0 and 1. This is just one particular coding or representation of a value which might otherwise

be represented (with the exact same value) as octal (base 8), decimal (base 10), or hexadecimal (base 16). Also see bit and Boolean.
2. The confusing counterpart to unary when describing the arity or number of inputs or arguments to a function, but dyadic is almost certainly a better choice.

Binomial Distribution

Binomial literally means "two names." In statistics, the probability of finding exactly *k* successes in *n* independent Bernoulli trials, each of which has exactly two possible outcomes, when the "success" probability is *p*:

$$P(k,n,p) = \binom{n}{k} p^{k} (1-p)^{n-k}$$

This ideal distribution is produced by evaluating the probability function for all possible *k,* from 0 to *n.*

If we have an experiment which we think *should* produce a binomial distribution, and then repeatedly and systematically find very improbable test values, we may choose to reject the null hypothesis that the experimental distribution is in fact binomial.

Also see the binomial section of my JavaScript page: "Binomial and Poisson Statistics Functions in JavaScript," locally, or @: http://www.ciphersbyritter.com/JAVASCRP/BINOMPOI.HTM#Binomial, and my early message on randomness testing (locally, or @: http://www.ciphersbyritter.com/NEWS2/94080601.HTM).

Bipolar

1. Having two polarities, as in + and - voltages (e.g., AC), and P and N semiconductor.
2. A junction transistor (NPN or PNP). As opposed to a field effect transistor.
3. Circuits or devices implemented with junction transistors.

Birthday Attack

A form of attack in which it is necessary to obtain two identical values from a large population. The "birthday" part is the realization that it is far easier to find an arbitrary matching pair than to match any particular value. Often a hash attack. Also see: population estimation, augmented repetitions, birthday paradox, and the conversation "Birthday Attack Calculations," locally, or @: http://www.ciphersbyritter.com/NEWS4/BIRTHDAY.HTM.

Birthday Paradox

The apparent paradox that, in a schoolroom of only 23 students, there is a 50 percent probability that at least two will have the same birthday. The "paradox" is that we have an even chance of success with just 23 of 365 possible days represented.

The "paradox" is resolved by noting that we have a 1/365 chance of success for each possible *pairing* of students, and there are 253 possible pairs or combinations of 23 things taken 2 at a time. (To count the number of pairs, we can choose any of the 23 students as part of the pair, then any of the 22 remaining students as the other part. But this counts each pair twice, so we have 23 * 22 / 2 = 253 different pairs.) Note that 253 / 365 = 0.693151.

This problem seems to beg confusion between probability and expected counts, since the correct expectation is often fractional. We can relate the probability of finding a "double" of some birthday (Pd) to the expected number of doubles (Ed) as approximately (equations (5.4) and (5.5) from my article):

$$Pd = 1 - e^{-Ed} \quad ,$$

```
so

   Ed = -Ln( 1 - Pd )  .
```

For a success probability of 0.5, the expected doubles are

```
   Ed = -Ln( 1 - 0.5 ) = 0.693147  .
```

One way to model the overall probability of success is from the probability of *failure* (1 - 1/365 = 0.99726) multiplied by itself for each pair that could have a possible match. For the birthday case this model gives the overall failure probability of 0.99726^{253} (0.99726 to the 253rd power) or 0.4995, for a success probability of 0.5005.

A different model addresses the probability of success for each sample, instead of each pair. For population (N) and samples (s) (equation (1.2) from my article):

```
  Pd(N,s) = 1 - (1-1/N)(1-2/N)..(1-(s-1)/N)  ,
```

which gives a success probability for 23 samples of 0.5073.

Sometimes the problem is to find the number of samples (s) needed for a given probability of success in finding doubles (Pd) from a given population (N). Starting with equation (2.5) from my article and substituting (5.5), we get:

```
   s(N,p) = (1 + SQRT(1 - 8N Ln( 1 - Pd ))) / 2  .
```

For the birthday case the number of samples needed from a population of 365 for an even chance of success is:

```
   s(365,0.5) = (1 + SQRT(1 - (8 * 365 * -0.693))) / 2
              = (1 + SQRT( 2024.56 )) / 2
              = 45.995 / 2
              = 22.997  .
```

This result means that 23 samples should meet with success just a little more often than the 1 time in 2 demanded by p = 0.5.

Also see: birthday attack, population estimation, augmented repetitions, my *Cryptologia* "birthday" article: "Estimating Population from Repetitions in Accumulated Random Samples," locally, or @: http://www.ciphersbyritter.com/ARTS/BIRTHDAY.HTM, and an example and comparison to various other methods in the conversation "Birthday Attack Calculations," locally, or @: http://www.ciphersbyritter.com/NEWS4/BIRTHDAY.HTM.

Bit

1. The smallest possible discrete unit of information. A Boolean value: True or False; Yes or No; one or zero; Set or Cleared. A contraction of "binary digit," apparently coined by J. W. Tukey. Virtually all information to be communicated or stored digitally is coded in some way which fundamentally relies on individual bits. Alphabetic characters are often stored in eight bits, which is a byte.
2. The real number average of information in bits per symbol as used in entropy, assuming the computation uses base-2 logs.

In digital electronics, bits generally are represented by voltage levels on connected wires, at a given time. When the bit-value on a wire changes, some time will elapse until the wire reaches the new voltage level. Until that happens, the wire voltage is not a valid digital level and should not be interpreted as having a particular bit value. Also see: logic level.

Bit Balance

The goal of achieving a balance, or equal numbers of 1's and 0's bits, in some vector or block. (Especially see Dynamic Transposition.)

There are various ways this might be achieved:

- Use a block code in which every codeword is balanced. This will expand the message somewhat.
- Use a code in which *most* codewords are balanced, with a state machine to count the unbalance and correct it as soon as possible (e.g., 8b10b). This also expands the message.
- Use a scrambler or simple Vernam cipher to randomize and statistically balance the data. This does not expand the message, but also does not guarantee perfect balance, although blocks of reasonable size will be "nearly" balanced to high probability.
- Use my approach as described in my article "Dynamic Transposition Revisited Again" (40K) (locally, or @: http://www.ciphersbyritter.com/ARTS/DYNTRAGN.HTM):

  "Exact bit-balance can be achieved by accumulating data to a block byte-by-byte, only as long as the block can be balanced by adding appropriate bits at the end."

  "We will always add at least one byte of 'balance data' at the end of the data, a byte which will contain both 1's and 0's. Subsequent balance bytes will be either all-1's or all-0's, except for trailing 'padding' bytes, of some balanced particular value. We can thus transparently remove the balance data by stepping from the end of the block, past any padding bytes, past any all-1's or all-0's bytes, and past the first byte containing both 1's and 0's. Padding is needed both to allow balance in special cases, and when the last of the data does not completely fill the last block."

  "This method has a minimum expansion of one byte per block, given perfectly balanced binary data. ASCII text may expand by as much as 1/3, which could be greatly reduced with a pre-processing data compression step."

Bit Permutation

A re-arrangement of the bits in a block of data. Also see bit permutation cipher.

Bit Permutation Cipher

A transposition cipher that re-arranges the positions of bits in the block of data being enciphered (bit permutation). For security even when the block is all-1's or all-0's, the data can be bit-balanced before transposition (see Dynamic Transposition).

(My article "A Keyed Shuffling System for Block Cipher Cryptography," illustrates key hashing, a nonlinearized RNG, and byte shuffling. We would do a similar thing for bit-permutation, but with a larger and wider RNG and shuffling bits instead of bytes. See either locally, or @: http://www.ciphersbyritter.com/KEYSHUF.HTM).

Ciphering by bit-transposition has unusual resistance to known plaintext attack because many, many different bit-permutations of the plaintext data will each produce exactly the same ciphertext result. Consequently, even knowing both the plaintext and the associated ciphertext does not reveal the shuffling sequence. Bit-permutation thus joins double-shuffling in hiding the shuffling sequence, which is important when we cannot guarantee the strength of that sequence (as we generally cannot).

A transposition cipher is "dynamic" when it "never" permutes two blocks in the same way. Dynamic bit-permutation ciphers can be a very competitive practical alternative to both stream ciphers and conventional block ciphers. Although bit-shuffling may be slower, it has a clearer and more believable source of strength than the other alternatives.

Also see my article: "Dynamic Transposition Revisited Again" (40K) (locally, or @:

http://www.ciphersbyritter.com/ARTS/DYNTRAGN.HTM), the Dynamic Transposition Ciphering conversation (730K) (locally, or @: http://www.ciphersbyritter.com/NEWS5/REDYNTRN.HTM).

Bit Shuffling

Bit transposition by the shuffle algorithm, see bit permutation.

Bit Transposition

Transposition by bit, see bit permutation.

Black

In TEMPEST, electrical wiring or signals carrying ciphertext which thus **can** be exposed without danger.

Black Box

An engineering term for a component only specified with input and output values, with the internal implementation irrelevant. In this way, the real complexity of the component need not be considered in understanding a system using the component.

Digital logic IC's are wildly successful examples of hardware black box components. Externally, they perform useful digital functions, and in most cases, digital designers need not think about the internal construction. Internally, however, the "digital" devices use analog transistors to effect digital operation.

An example of black box software design is a subroutine or Structured Programming module, where all interaction with the caller is in the form of parameters. The module uses the given resources, does what it needs, completes, and returns to the caller. As long as the module does what we want, there is no need to know *how* the module works, so we can avoid dealing with internal complexity at the lower level. And when the module does *not* work, it can be debugged in a minimal environment which avoids most of the complexity of the larger system, thus making debugging far easier.

Block

1. A fixed amount of data treated as a single unit.
2. More than one element treated as a single unit.

As opposed to a sequence of elements, as in a stream.

In a discussion of block cipher concepts, cryptography implicitly uses definition (2), because it is the accumulation of *multiple* characters (and the resulting larger ciphering alphabet) which is characteristic of conventional block ciphers. A one-element "block" simply cannot exhibit the various block issues (such as mixing, diffusion, padding and expansion) that we see in a real block cipher, and so fails to model both the innovation and the resulting problems. Similar effects occur when any scalable model is simplified beyond reason. (See: scientific method.) It is also possible to cipher blocks of dynamically selectable size, or even fine-grained variable size.

All real block ciphers are in fact streamed to handle more than one block of data. The actual ciphering might be seen as a stream meta-ciphering using a block cipher transformation. The point of this is not to provide a convenient academic way to contradict any possible response to a question of "stream or block," but instead to identify the origin of various ciphering properties and problems (see: a cipher taxonomy).

It is not possible to block-cipher just a single bit or byte of a block. (When that *is* possible, we may be dealing with a stream cipher.) If individual bytes really must be block-ciphered, it will be necessary to fill out each block with padding in some way that allows the padding to be distinguished from the actual plaintext data after deciphering.

Partitioning an arbitrary stream of into fixed-size blocks generally means the ciphertext handling must support data expansion, if only by one block. But handling even minimal data expansion may be difficult in some systems.

The distinction between "block" and "stream" corresponds to the common distinction between "block" and "character" device drivers in operating systems. This is the need to accumulate multiple elements and/or pad to a full block before a single operation, versus the ability to operate without delay but requiring multiple operations. This is a common, practical distinction in data processing and data communications.

A competing interpretation of block versus stream operation seems to be based on transformation "re-use": In that interpretation, block ciphering is about having a complex transformation, which thus directly supports re-use (providing each plaintext block "never" re-occurs). In that same interpretation, stream ciphering is about supporting transformation re-use by changing the transformation itself. These effects do of course exist (although in my view they are not the most fundamental issues for analysis or design). But that interpretation also allows both qualities to exist simultaneously at the same level of design, and so does not provide the full analytical benefits of a true logical dichotomy.

Block Cipher

A cipher which requires the accumulation of multiple data characters (or bytes) in a block before ciphering can start. This implies a need for *storage* to hold the accumulation, and *time* for the accumulation to occur. It also implies a need to handle partly-filled blocks at the end of a message. In contrast, a stream cipher can cipher bytes immediately, as they occur, and as many or as few as are required. Block ciphers can be called "codebook-style" ciphers, and are typically constructed as product ciphers, thus showing a broad acceptance of multiple encryption. Also see variable size block cipher and a cipher taxonomy.

There is some background:

- The exact definition of a block cipher is oddly controversial, see: block cipher definitions.
- There is some confusion over appropriate block and stream scientific models, see block cipher and stream cipher models.
- There is a disturbing insistence by some academics that only one model deserves the name "block cipher." That is a problem because other types of cipher do operate on data in blocks, yet do not follow that model. The problem is that general characteristics of "block ciphering" are unlikely to be resolved when only one type of cipher is allowed under that name. This Glossary does not recognize those limitations.

Conventional Block Ciphers

A conventional block cipher is a transformation between all possible plaintext block values and all possible ciphertext block values, and is thus an emulated simple substitution on huge block-wide values. Within a particular block size, both plaintext and ciphertext have the same set of possible values, and when the ciphertext values have the same ordering as the plaintext, ciphering is obviously ineffective. So *effective* ciphering depends upon *re-arranging* the ciphertext values from the plaintext ordering, and this is a permutation of the plaintext values. A conventional block cipher is keyed by constructing a *particular* permutation of ciphertext values for each key.

The mathematical model of a conventional block cipher is bijection, and the set of all possible block values is the alphabet. In cryptography, the bijection model corresponds to an invertible table having a storage element associated with each possible alphabet value. Since each different table represents a different permutation of the alphabet, the number of possible tables is the factorial of the alphabet size.

In particular, a conventional block cipher with a 64-bit block has an alphabet size of 2^{64} (that is, 2**64) elements (about 18446744073709552000), and the potential number of keys is the factorial of that, a number which needs 1.15397859e+22 bits (about 10**22 bits or more than a million million million bits) for full representation. (For these and similar computations, try the "Base Conversion, Logs, Powers, Factorials, Permutations and Combinations in JavaScript" page locally, or @: http://www.ciphersbyritter.com/JAVASCRP/PERMCOMB.HTM) But, in practice, a popular block cipher like DES can only select from among 2^{56} (that is, 2**56) different tables using 56 key bits. To calculate the ratio of two values expressed as exponents we subtract, so to find the proportion of tables we can actually use, we have $1.15 \times 10^{22} - 56$, (or 1.15*(10**22)), which does not even change the larger expressed value. Thus, DES and other conventional block ciphers generally support only an almost infinitesimal fraction of the keys possible under their own mathematical and cryptographic model. This is an inherent selection of a tiny subset of keys, which is a massive deviation from the model of balanced, flat or unbiased keying across all possibilities. Also see AES.

Block Cipher Data Diffusion

In an ideal conventional block cipher, changing even a single bit of the input block will change all bits of the ciphertext result, each with independent probability 0.5. This means that *about* half of the bits in the output will change for any different input block, even for differences of just one bit. This is overall diffusion and is present in a block cipher, but usually not in a stream cipher. Data diffusion is a simple consequence of the keyed invertible simple substitution nature of the ideal block cipher.

Improper diffusion of data throughout a block cipher can have serious strength implications. One of the functions of data diffusion is to hide the different effects of different internal components. If these effects are not in fact hidden, it may be possible to attack each component separately, and break the whole cipher fairly easily.

Partitioning Messages into Fixed Size Blocks

A large message can be ciphered by partitioning the plaintext into blocks of a size which can be ciphered. This essentially creates a stream meta-cipher which repeatedly uses the same block cipher transformation. Of course, it is also possible to re-key the block cipher for each and every block ciphered, but this is usually expensive in terms of computation and normally unnecessary.

A message of arbitrary size can always be partitioned into some number of whole blocks, with possibly some space remaining in the final block. Since partial blocks cannot be ciphered, some random padding can be introduced to fill out the last block, and this naturally expands the ciphertext. In this case it may also be necessary to introduce some sort of structure which will indicate the number of valid bytes in the last block.

Block Partitioning without Expansion

Proposals for using a block cipher supposedly *without* data expansion may involve creating a tiny stream cipher for the last block. One scheme is to re-encipher the ciphertext of the preceding block, and use the result as the confusion sequence. Of course, the cipher designer still needs to address the situation of files which are so short that they *have* no preceding block. Because the one-block version is *in fact* a stream cipher, we must be very careful to never re-use a confusion sequence. But when we only *have* one block, there *is* no prior block to change as a result of the data. In this case, ciphering several very short files could expose those files quickly. Furthermore, it is dangerous to encipher a CRC value in such a block, because exclusive-OR enciphering is transparent to the field of mod 2 polynomials in which the CRC operates. Doing this could allow an opponent to adjust the message CRC in a known way, thus avoiding authentication exposure.

Another proposal for eliminating data expansion consists of ciphering blocks until the last short block, then re-positioning the ciphering window to end at the last of the data, thus re-ciphering part of the prior block. This is a form of chaining and establishes a sequentiality requirement which requires that the last block be deciphered *before* the next-to-the-last block. Or we can make enciphering inconvenient and deciphering easy, but one way will be a problem. And this approach cannot handle very short messages: its minimum size is one block. Yet any general-purpose ciphering routine *will* encounter short messages. Even worse, if we have a short message, we still need to somehow indicate the correct length of the message, and this must expand the message, as we saw before. Thus, overall, this seems a somewhat dubious technique.

On the other hand, it does show a way to chain blocks for authentication in a large-block cipher: We start out by enciphering the data in the first block. Then we position the next ciphering to start *inside* the ciphertext of the previous block. Of course this would mean that we would have to decipher the message in reverse order, but it would also propagate any ciphertext changes through the end of the message. So if we add an authentication field at the end of the message (a keyed value known on both ends), and that value is recovered upon deciphering (this will be the first block deciphered) we can authenticate the whole message. But we still need to handle the last block padding problem and possibly also the short message problem.

Block Size and Plaintext Randomization

Ciphering raw plaintext data can be dangerous when the cipher has a relatively small block size. Language plaintext has a strong, biased distribution of symbols and ciphering raw plaintext would effectively reduce the number of possible plaintext blocks. Worse, some plaintexts would be vastly more probable than others, and if some known plaintext were available, the most-frequent blocks might already be known. In this way, small blocks can be vulnerable to classic codebook attacks which build up the ciphertext equivalents for many of the plaintext phrases. This sort of attack confronts a particular block size, and for these attacks Triple-DES is no stronger than simple DES, because they both have the same block size.

The usual way of avoiding these problems is to randomize the plaintext block with an operating mode such as CBC. This can ensure that the plaintext data which is actually ciphered is evenly distributed across all possible block values. However, this also requires an IV which thus expands the ciphertext.

Worse, a block scrambling or randomization function like CBC is *public*, not private. It is easily reversed to check overall language statistics and thus distinguish the tiny fraction of brute force results which produce potentially valid plaintext blocks. This directly supports brute force attack, as well as any attack in which brute force is a final part. One alternative is to use a preliminary cipher to randomize the data instead of an exposed function. Pre-ciphering prevents easy plaintext discrimination; this is multiple ciphering, leading in the direction Shannon's Ideal Secrecy.

Another approach (to using the full block data space) is to apply data compression to the plaintext before enciphering. If this is to be used *instead* of plaintext randomization, the designer must be very careful that the data compression does not contain regular features which could be exploited by the opponents.

An alternate approach is to use blocks of sufficient size for them to be expected to have a substantial amount of uniqueness or entropy. If we expect plaintext to have about one bit of entropy per byte of text, we might want a block size of at least 64 bytes before we stop worrying about an uneven distribution of plaintext blocks. This is now a practical block size.

Block Cipher Defintions

As far as we know, the original automated ciphers were stream ciphers developed from the Vernam work with teleprinter encryption, as patented in 1919. Block terminology seems to have come along much later, to distinguish fundamentally different designs from the old, well-known streams. This distinction occurred long before modern open cryptographic analysis. Distinguishing "block" from "stream" in the present day is important because it is *useful*: a true dichotomy is the friend both of the analyst and student. (Also see a cipher taxonomy.)

It may be helpful to recall a range of published distinctions between "stream cipher" and "block cipher" (and if anyone has any earlier references, please send them along). Note that open discussion was notably muted during the Cold War, especially during the 50's, 60's and 70's. I see the earlier definitions as attempts at describing an existing codification of knowledge, which was at the time tightly held but nevertheless still well-developed.

Some Early Definitions

- **1976.** "Much as error correcting codes are divided into convolutional and block codes, cryptographic systems can be divided into two broad classes: *stream ciphers* and *block ciphers*. Stream ciphers process the plaintext in small chunks (bits or characters), usually producing a pseudo-random sequence of bits which is added modulo 2 to the bits of the plaintext. Block ciphers act in a purely combinatorial fashion on large blocks of text, in such a way that a small change in the input block produces a major change in the resulting output." (p. 646) *[Unfortunately, these two different definitions establish **four** classes, not just the **two** classes suggested by the first sentence. For example, what do we call a cipher which acts on "blocks of text" but **not** in a "purely combinatorial fashion"? Having two different distinctions for only two classes is an error that we need to recognize and get beyond. /tfr]*. -- Diffie, W. and M. Hellman. "New Directions in Cryptography." *IEEE Transactions on Information Theory.* Vol. IT-22, No. 6, November 1976.

- **1979.** "Transposition is a *block* cipher which operates on words (blocks) of length *n*, employing as keys the *n!* positional permutations to transpose the letters of a word." (p. 290) -- Lempel, A. "Cryptology in Transition." *ACM Computing Surveys.* Vol. 11, No. 4, Dec. 1979.

- **1979.** "A primitive distinction among cryptosystems is the structural classification into stream and block ciphers." "A stream cipher operates on the plaintext symbol by symbol to produce a sequence of cipher symbols . . . ." "In a block cipher a block of symbols from *A* is operated on jointly by the encryption algorithm . . . ." (p. 306) -- Simmons, G. "Symmetric and Asymmetric Encryption." *ACM Computing Surveys.* Vol. 11, No. 4, Dec. 1979.

- **1979.** "**Block ciphers** divide the plaintext into blocks, usually of a fixed size, and operate on each block independently." (p. 415) "**Stream ciphers**, in contrast, do not treat the incoming characters independently." (p. 415) *[Note that* blocking *and* independence *are **different** concepts, thus introducing the confusion of exactly which part of the definition to follow. /tfr]* -- Diffie, w. and M. Hellman. "Privacy and Authentication: An Introduction to Cryptography." *Proceedings of the IEEE.* Vol. 67, No. 3, March 1979.

- **1982.** "A block cipher transforms a string of input bits of fixed length (an input block) into a string of output bits of fixed length (an output block)." (p. 23) "A stream cipher employs a bit-stream generator to produce a stream of binary digits . . . which is then combined either with plaintext . . . to produce ciphertext, or with ciphertext . . . to recover plaintext." (p. 53) -- Meyer, C. and S. Matyas. *Cryptography: A New Dimension in Data Security.*

Some Current Definitions

- **1996.** "Symmetric algorithms can be divided into two categories. Some operate on the plaintext a single bit (or sometimes byte) at a time; these are called **stream algorithms** or **stream ciphers**. Others operate on the plaintext in groups of bits. The groups of bits are called **blocks**, and the algorithms are called

block algorithms or **block ciphers**." (p. 4) -- Schneier, B. *Applied Cryptography.*

- **1997.** "A block cipher is a function which maps *n*-bit plaintext blocks to *n*-bit ciphertext blocks; *n* is called the blocklength. It may be viewed as a simple substitution cipher with large character size." (p. 224) *[But this definition excludes the case of a cipher with an output block larger than the input block, so what would such a cipher be called? /tfr]* "Stream ciphers . . . are, in one sense, very simple block ciphers having block length equal to one." "They also can be used when the data must be processed one symbol at a time (e.g., if the equipment has no memory, or buffering of data is limited)." (p. 20) -- Menezes, A., van Oorschot, P. and Vanstone, S. *Handbook of Applied Cryptography.*

The intent of classification is understanding and use. Accordingly, it is up to the analyst or student to "see" a cipher in the appropriate context, and it is often useful to consider a cipher to be a hierarchy of ciphering techniques. For example, it is extremely rare for a block cipher to encipher exactly one block. But when that same cipher is re-used again that seems a lot like repeated substitution, which is the basis for stream ciphering. (Of course repeatedly using the same small substitution would be ineffective, but if we attempt to classify ciphers by their effectiveness, we start out assuming what we are trying to understand or prove.) So an alternate way to "see" the re-use of a block cipher is as a higher-level stream "meta-cipher" which uses a block cipher component. But that is exactly what we call "block ciphering."

Alternate Distinctions

Some academics insist upon distinguishing *stream* versus *block* ciphering by saying that block ciphers have no retained state between blocks, while stream ciphers do. Simply saying that, however, does not make it true, and only one example is needed to expose the distinction as false and misleading. A good example for that is my own Dynamic Transposition cipher, which is a block cipher in that it requires a full block of data before processing can begin, yet also retains state between blocks. So if DT is not a block cipher, what is it? We would hope to define only two categories, not four or more. Note that Lempel (1979, above) explicitly says that transposition *is* a block cipher. Again, see: a cipher taxonomy to see one approach on how ciphers relate.

Blocks by Implementation

Another issue is that stream ciphers can be implemented in ways that accumulate a block of data before ciphering. Internally, such systems generally have a streaming system which traverse the block element-by-element, perhaps multiple times. It is important to see beyond an apparent block requirement stemming from data manipulation only, which thus contributes no strength, to the internal ciphers which (hopefully) *do* provide strength.

It is also possible to have multiple stream ciphers work on the same "block," and then we do have a legitimate "block cipher" (or perhaps a "block meta-cipher") formed by multiple encryption of stream ciphers. (Although multiple ciphering with additive stream ciphers is usually unhelpful, most conventional block ciphers are in fact multiple encryptions internally, so internal multiple ciphering is hardly a crazy approach.) But if we want to understand strength, we still need to consider the fundamental ciphering operations which, here, are streams. Simply making something work like a block cipher does not give it the same model as a conventional block cipher, and so does not provide for analysis at that level. In the end, we might see such a construction as a block meta-cipher composed of internal stream ciphers.

Block Cipher and Stream Cipher Models

In the study of technology, it is often important to create a scientific model which describes observed reality. In cryptography, things are more fluid than in the physical sciences, because the reality of ciphering is itself is a construction, and there are many such constructions. Consequently, it is not always obvious what model delivers the best insight. In fact, different models may provide different insights on exactly the same reality.

Currently, there are three main block cipher models:

- A transformation which is a keyed substitution.
- The transformation which has no retained state.
- The simple collection of multiple elements.

Each of these models has widely different implications, advantages and problems.

The Block as a Bijection

The common academic model of a block cipher is the mathematical bijection, which cryptography calls simple substitution. In practice, such a cipher requires a table far too large to instantiate, and so the actual cipher only *emulates* a huge, keyed table.

One advantage of the bijection model is that specific, measurable mathematical things can be said about a bijection. Of course exactly the same things also can be said about simple substitution, and the field of ciphering is cryptography, not mathematics.

One problem with the bijection model is that it does not attempt to establish a dichotomy. In the bijection model, "block cipher" just another label in a presumably endless sequence of such labels, each representing a distinct ciphering approach. Consequently, the bijection model makes a poor contribution toward an overall cipher taxonomy useful in the analysis of arbitrary cipher designs.

Another problem with the bijection model is that it establishes yet another term of art: The word block is well known, understood, and rarely disputed. The word cipher is also widely agreed upon. The phrase "block cipher" obviously includes nothing about bijections. So to define "block cipher" in terms of bijections is to take the phrase far beyond the simple meaning of the terms. We could scarcely describe this as anything other than misleading.

Yet another problem with the bijection model, is that, since it presumes to define "block cipher" as a particular type of cipher, what are we to do with ciphers which operate on blocks and yet do not function as bijections (e.g., transposition cipher)? No longer are ciphers related by their proper description. This is even more misleading.

Ultimately, the problem with the bijection model is not the model itself: The model is what it is because substitution is what it is. The problem is the insistence by some academics that this is the *only* valid model for a "block cipher." A much better choice for the bijection model is the phrase: "conventional block cipher."

The Block as Static State

The static state model puts forth the proposition that stream ciphers dynamically change their internal state, whereas block ciphers do not. Typically, there is also an understanding that the bijective block cipher model applies.

One problem with the static state definition is again in the name itself: The phrase "block cipher" does not include the word "state." To use the phrase "block cipher" for a property of state is to create yet another term of art, preempting the obvious meaning of the phrase "block cipher," and preventing related block-like ciphers from having similar descriptions, thus misleading both instructor and student.

Another problem with the static state model is that we can build stream-like ciphers which do *not* change their internal state (in fact, I claim we stream a substitution table when we repeatedly use it across a message, just like we stream DES). Similarly, we can build block-like ciphers which *do* change their internal state (I usually offer my Dynamic Transposition cipher as an example, but so is a block cipher built from multiple internal stream operations). So if we accept the static state model, what do we call those ciphers which function on blocks, and yet *do* change state? Why preempt the well-known terms "block" or "stream" for the fundamentally different properties of internal state?

Ultimately, what insight does state classification provide that warrants usurping the obvious descriptive phrases "block cipher" and "stream cipher" instead of thinking up something appropriate?

The Block as Multiple Elements

The original mechanized ciphers were stream ciphers, starting with the Vernam cipher of 1919. The term "block cipher" may have been introduced in the secret world of government security to draw a practical distinction between the well-known stream concept, and the newer designs that operated on a block. (That would have been in the 50's, 60's or even 70's; hopefully, someone will either confirm this or correct it.) In the multiple-element model, a block cipher requires the accumulation of more than one data element before ciphering can begin.

One advantage of the multiple-element definition is that it forms an easy dichotomy with the definition of a stream cipher as a cipher which does *not* require such accumulation. Also note that this is no mere semantic issue, but is instead just one representation of a broader concept of "one versus many" which rises repeatedly in computing practice, including:

- "block" versus "character" device drivers in an OS, and
- "parallel" versus "serial" buses in digital electronics.

The various consequences of the single-element versus multiple-element dichotomy are well known: When blocks are accumulated from individual elements, storage is required for that accumulation, and time is required as well, which can imply latency. In contrast, when elements need not be accumulated, there need be neither storage nor latency, but the total overhead may be greater. While latency probably is not much of an issue for email ciphering, latency can be significant for real-time streams like music or video, or interactive handshake protocols. Overhead is, of course, a significant issue in system design.

To see how the multiple-element block cipher definition works, consider the following:

- What is a transposition cipher? It is **one form** of a block cipher.
- What is a cipher which repeatedly runs several different internal stream-like ciphers across a data block? That is also **one form** of a block cipher.
- What is simple substitution? Also **one form** of a block cipher. Although *not* **the** form of a block cipher, it is the "*conventional* block cipher."

Strangely, a degenerate block is exactly the same as a degenerate sequence: just one element. In neither case does that element teach about the larger object: a one-element block does not have diffusion between elements, and a one-element stream does not have correlation between elements. (Similarly, is a single electronic wire with a fixed voltage one-wire "parallel" or one-value "serial"?) From this we conclude that the most important aspects of cryptographic (and electronic) design and analysis simply do not exist as a single element, so it is inappropriate to either use or judge a model at that level.

Block Code

In coding theory, given alphabet A, and length or block size *n*, the set of all sequences or codewords which consist of n selections from among set A, denoted $\{A\}^n$.

A block size of *n* bits typically means that 2^n different codewords or "block values" can occur. An (*n*,*k*) block code uses those 2^n codewords to represent the equal or smaller count of 2^k different messages. Thus, a 64-bit block cipher normally encodes 64 plaintext bits into 64 ciphertext bits as a simple (64,64) code. But if 16 input bits are reserved for other use, the coding expands 48 plaintext bits into 64 ciphertext bits, so we have a (64,48) code.

The normal use for extra codewords is to implement some form of error detection and/or error correction. This overhead is not normally called "inefficient coding," but is instead a simple cost of providing improved quality. In cryptography, the extra code words may be used to add security or improve performance by implementing:

- homophonic coding,
- dynamic keying or
- per-block authentication.

Also see 8b10b and huge block cipher advantages.

Block Size

The amount of data in a block. For example, the size of the DES block is 64 bits or 8 bytes or 8 octets.

Blum, Blum and Shub

BB&S.

Boolean

TRUE or FALSE; one bit of information.

Boolean Algebra

The logic of digital electronics. The algebraic ring formed by the set {0,1} with operations:

```
NOT as complementation, indicated by ' (single quote)
    0' = 1
    1' = 0
OR as addition, denoted "+"
    0 + 0 = 0
    0 + 1 = 1
    1 + 0 = 1
    1 + 1 = 1
AND as multiplication, denoted "*" as usual
    0 * 0 = 0
    0 * 1 = 0
    1 * 0 = 0
    1 * 1 = 1
XOR a useful but not essential operation
    0 XOR 0 = 0
    0 XOR 1 = 1
    1 XOR 0 = 1
    1 XOR 1 = 0

1. Addition is Commutative:          x + y = y + x
2. Addition is Associative:          x + (y + z) = (x + y) + z
3. The Additive Identity:            0 + x = x
4. The Additive Inverse:             x + x' = 1
5. Multiplication is Associative:    x(yz) = (xy)z
6. Multiplication is Distributive:   x(y + z) = xy + xz
7. Multiplication is Commutative:    xy = yx
8. The Multiplicative Identity:      1 * x = x

Other:
         1 + x = x
         x + x = x
         x * x = x
        (x')' = x

DeMorgan's Laws:
```

```
             (x + y)' = x'y'
             (xy)' = x' + y'

      XOR:
              x XOR y = xy' + x'y
             (x XOR y)' = xy + x'y'
```

Boolean Function

A function which produces a Boolean result. The individual output bits of an S-box can each be considered to be separate Boolean functions. Also see logic function.

Boolean Function Nonlinearity

The number of bits which must change in the truth table of a Boolean function to reach the closest affine Boolean function.

Typically computed as the fast Walsh-Hadamard transform (FWT) of the function being measured. For more details, see the topic unexpected distance and the "Active Boolean Function Nonlinearity Measurement in JavaScript" page (locally, or @: http://www.ciphersbyritter.com/JAVASCRP/NONLMEAS.HTM).

Note that the FWT computation is done for efficiency only. It is wholly practical to compute the nonlinearity of short sequences by hand. It is only necessary to manually compare each bit of the measured sequence to each bit of an affine Boolean function. That gives us the distance from that particular function, and we repeat that process for every possible affine Boolean function of the measurement length.

Especially useful in S-box analysis, where the nonlinearity for the table is often taken to be the minimum of the nonlinearity values computed for each output bit.

Also see my articles:

- "Measured Boolean Function Nonlinearity in Mixing Cipher Constructions" locally, or @: http://www.ciphersbyritter.com/ARTS/MIXNONLI.HTM
- "Measured Boolean Function Nonlinearity in Variable Size Block Ciphers" locally, or @: http://www.ciphersbyritter.com/ARTS/VSBCNONL.HTM
- "Measured Boolean Function Nonlinearity in Feistel Cipher Constructions" locally, or @: http://www.ciphersbyritter.com/ARTS/FEISNONL.HTM

Boolean Logic

The logic which applies to variables which have only two possible values. Also the digital hardware devices which realize such logic, and are used to implement a electronic digital computers. Also see Boolean algebra.

Boolean Mapping

A mapping of some number *n* Boolean variables into some number *m* Boolean results. For example, an S-box.

Braid

The stream cipher formed by intermingling or *multiplexing* two or more streams of data into a single stream of ciphertext. For encryption, presumably some form of keyed RNG would select which plaintext stream would contribute the next bit or byte to the ciphertext stream. For decryption, a similar keyed RNG would *demultiplex* or allocate each ciphertext bit or byte to the appropriate plaintext stream. If the different streams are similar, from the ciphertext it should be difficult to distinguish which bits or bytes belong to which plaintext stream. Accordingly, the scheme may work best as a super encryption or a meta-cipher handling already encrypted and thus very similar random-like streams. See: "Simon's Braided Stream Cipher" (locally, or @: http://www.ciphersbyritter.com/BRAID/BRAID.HTM). Also see ciphertext expansion, transposition and null.

Branch Number

Measures of block mixing: The minimum number of *nonzero* elements (e.g., bytes) in both the input and output, over all possible inputs. Or the minimum number of *changing* elements in both the input and output, over all possible input changes. When that mixing is applied between layers of S-boxes, this can be the minimum number of changing or active S-boxes.

The Technical Definitions

The original definition is for linear mapping *theta*,

> "The minimum total Hamming weight [w_h] of (a,theta(a)) is a measure for the minimum amount of diffusion that is realized by a linear mapping."
>
> **Definition 6.10** *The branch number B of a linear mapping theta is given by*
>
> ```
> B(theta) = min(w_h(a),w_h(theta(a))), for a<>0
> ```
>
> -- Daemen, J. 1995. Cipher and Hash Function Design, Strategies Based on Linear and Differential Cryptanalysis. Thesis. Section 6.8.1.

Branch number specifically applies only to a linear mixing. Actually, even that is not quite right: the real problem is keying, not nonlinearity (although in practice, keying may imply nonlinearity). To the extent that we can experimentally traverse the input block, a branch number certainly can be developed for a nonlinear mixer.

But while any particular mixer *can* have a branch number, a keyed mixer will have a branch number *for every possible key*. Moreover, we would expect the minimum over all those nonlinear mixings to be very low, just like the minimum strength of any cipher over all possible keys (the opponent trying just one key) is also very low. Yet we do not attempt to characterize ciphers by their minimum strength over all possible keys.

No keyed structure can be properly characterized by the extrema over all keys. When we have random variables such as keying, we should be thinking of the distribution of values, and the probability of encountering extreme values. And that is not branch number.

More insight is available in the description of the SQUARE cipher:

> "It is intuitively clear that both linear and differential trails would benefit from a multiplication polynomial that could limit the number of nonzero terms in input and output difference (and selection) polynomials. This is exactly what we want to avoid by choosing a polynomial with a high diffusion power, expressed by the so-called *branch number*.
>
> Let $w_h(a)$ denote the Hamming weight of a vector, i.e., the number of nonzero components in that vector." *[Normally, Hamming weight applies to bits, but here it is being used for bytes./tfr]* "Applied to a state a, a difference pattern a' or a selection pattern u, this corresponds to the number of non-zero bytes. In [2] the branch number B of an invertible linear mapping was introduced as
>
> ```
> B(theta) = for a<>0, min w_h(a)+ w_h(a)
> ```
>
> This implies that the sum of the Hamming weights of a pair of input and output difference patterns (or selection patterns) to theta is at least B. It can easily be shown that B is a lower bound for the number of active S-boxes in two consecutive rounds of a linear or differential trail."
>
> "In [15] it was shown how a linear mapping over $GF(2^m)^n$ with optimal B ($B = n + 1$) can be constructed from a maximum distance separable code."

-- Daemen, J., L. Knudsen and V. Rijmen." 1997. "The Block Cipher {SQUARE}." *Fast Software Encryption, Lecture Notes in Computer Science* Vol. 1267:149-165. Section 4.

Measurement of Linear Mixing

In the wide trail strategy, branch number applies to a particular *unkeyed* and *linear* diffusion mechanism. In the SQUARE design, branch number also applies to a particular *unkeyed* and *linear* polynomial multiplication. So branch number might also describe the simple linear form of Balanced Block Mixing used in Mixing Ciphers. But linear BBM's apparently do not have an optimal branch number (over all possible input data changes), although in most cases they do have a good branch, and are dynamically scalable to both tiny and huge blocks on a block-by-block basis.

Nonlinear Mixing

Instead of linear diffusion, it should be "intuitively obvious" that *non*linear diffusion would be a better choice for a cipher, if such could be obtained with good quality at reasonable cost. Nonlinear Balanced Block Mixing occurs when the butterfly functions are keyed. Keying is easily accomplished by constructing appropriate orthogonal Latin squares using the fast checkerboard construction. But "branch number" does not apply to these keyed nonlinear constructions.

The Optimal Branch Value

The "optimal" branch value for the MDS codes in the SQUARE design is given as $n + 1$. The branch number is basically the minimum number of *input* and *output* elements which are guaranteed to change, and there are $2n$ such elements. But when we try all possible cases of changed input data across a block, somewhere in there we ourselves create various worst-case inputs with only one element changed. In those cases, even if *all* n outputs change, we only get a branch of $n + 1$, so that is the most possible.

Break

1. To destroy an item, or offend and thus destroy an agreement.
2. In cryptography, an attack which destroys the advantage of a cipher in hiding information. We would call such an attack "successful."

From the Handbook of Applied Cryptography:

> "1.23 Definition. An encryption scheme is said to be *breakable* if a third party, without prior knowledge of the key pair (e,d), can systematically recover plaintext from corresponding ciphertext in some appropriate time frame." [p.14]
>
> "*Breaking* an information security service (which often involves more than simply encryption) implies defeating the objective of the intended service." [p.15]

The term "break" seems to be a term of art in academic cryptanalysis, where it apparently means a successful attack which takes less effort than brute force (or the cipher design strength, if that is less), even if the effort required is impractical, and even if the attack is easily prevented at the cipher system level. This meaning of the term "break" can be seriously misleading because, in English, "break" means "to render unusable" or "to destroy," and not just "to make a little more dubious."

The academic meaning of "break" is also controversial, as it can be used as a slander to demean both cipher and designer without a clear analysis of whether the attack really succeeds. And even if the attack *does* succeed, the question is whether it actually reveals data or key material, thus making the cipher dangerous for use in practice.

Everyone understands that a cipher is "broken" when the information in a message can be extracted without the key, or when the key itself can be recovered, with less effort than the design strength. And a break is particularly significant when the work involved need not be repeated on every message. But when the amount of work involved is impractical, the situation is best described as a *theoretical* or academic break. The concept of an "academic break" is especially an issue for ciphers with a very large keyspace, in which case it is perfectly possible for a cipher with an academic break to be *more* secure than ciphers with lesser goals which have no "break." It is also at least conceivable that an attack can be surprising and insightful and, thus, "successful" even if it takes *more* effort than the design strength, which would be no form of "break" at all.

In my view, a documented flaw in a cipher, such as some statistic which distinguishes a practical cipher from some model, but without an attack which recovers data or key, at most should be described as a "theoretical" or "certificational" *weakness*. Unfortunately, even a problem which has no impact on security is often promoted (improperly, in my view) to the term academic break or even "break" itself.

Brute Force Attack

A form of attack in which each possibility is tried until success is obtained. Typically, a ciphertext is deciphered under different keys until plaintext is recognized. On average, this should take about half as many decipherings as there are keys. Of course, the possibility exists that the correct key might be chosen first.

Even when the key length of a cipher is sufficient to prevent brute force attack, that key will be far too small to produce every possible plaintext from a given ciphertext (see Shannon's Perfect Secrecy). Combined with the fact that language is redundant, this means that very few of the decipherings will be words in proper form. So most wrong keys could be identified immediately.

On the other hand, recognizing plaintext may not be easy. If the plaintext itself -- and all known structure in the plaintext -- could be hidden, even a brute force attack on the keys could not succeed, for the correct deciphering could not be recognized when it occurred. If the plaintext was not language, but computer code, compressed text, or even ciphertext from another cipher, recognizing a correct deciphering could be difficult. It seems odd that more systems do not seek to leverage this advantage. See: known plaintext, Shannon's Ideal Secrecy and multiple encryption.

Brute force is the obvious way to attack a cipher, and the way most ciphers can be attacked, so ciphers are designed to have a large enough keyspace to make this much too expensive to succeed in practice. Normally, the design strength of a cipher is based on the cost of a brute-force attack.

Bug

Technical slang for "error in design or implementation." An unexpected system flaw. Debugging is a normal part of system development and interactive system design.

Burden of Proof

A legal term of art basically meaning that those who put forth an argument based on certain facts have the burden of supplying proof that the facts are as stated. Also see: extraordinary claims.

Butterfly

A pair of dyadic functions used as the fundamental operation for "in place" forms of FFT. The name comes from a common graphic depiction of FFT operation which has the shape of a standing "hourglass," or butterfly wings on edge. Also see fast Walsh transform.

In most FFT diagrams, the input elements are shown in a vertical column at the left, and the result elements in a vertical column on the right. Lines represent signal flow from left to right. There are two computations, and each requires input from each of the two selected elements. In an "in place" FFT, the results conveniently go back into the same positions as the input elements. So we have two horizontal lines between the same elements, and two diagonal lines going to each "other" element, which cross. This is the "hourglass" shape or "butterfly wings" on edge.

Bypass

In electronics, typically a capacitor from a power lead to circuit ground, thus "bypassing" noise to ground instead of distributing noise to other circuitry. Also see decoupling and amplifier.

A source of stable power is the most important requirement for any electronic device. In particular, digital logic functions can only be trusted to produce correct results if their power is kept within specified limits. It is up to the designer to provide sufficient correct power and guarantee that it remain within limits despite whatever else is going on.

Most digital logic families use "totem pole" outputs, which means they have a transistor from Vcc or Vdd (power) to the output pin, and another transistor from the output pin to Vss (ground). Normally, only one transistor is ON, but as the output signal passes from one state to another, transiently, *both* transistors can be ON, leading to short, high-current pulses on both the Vcc and ground rails. These current pulses are essentially RF energy, and can and do produce ringing on power lines and a general increase in system noise. The pulses are also strong enough to potentially change both the Vcc and ground voltage levels in the power distribution system near the device, which can affect nearby logic and operation. Typically this occurs at some random moment when the worst conditions coincide to cause a logic fault. To avoid that, we want to bypass the current pulse away from the power system in general, so other devices are not affected.

For many years, a typical rule of thumb was to use a 0.1uF ceramic disc for each supply at each bipolar chip, plus a 1uF tantalum for every 8 chips. That may still be a good formula for slower analog chips and older digital logic like LSTTL. But as chip speed has increased, bypassing has become more complex.

Ideally, a bypass capacitor will be connected from every supply pin to the ground pin right at each chip. Ideally, there will be no lead left on either end of the capacitor: not 1/4 inch, not 1/8 inch, which is one reason why surface-mount capacitors are desirable. Ideally, any necessary lead will be wide, flat copper. But the ideal system is a goal, not reality.

One of the effects of higher system speeds is that normal system operation now covers the resonant frequency of the bypass capacitors. Unfortunately, this resonance is not a fixed constant, even for a particular type of part. Bypass resonance is instead a circuit condition, involving the reactance of the closest bypass capacitor, plus the inductance in power connections, and reactance in other bypass capacitors. Although it is virtually impossible to remove inductance from PC-board traces, it is possible to use whole copper layers as "power planes" for power distribution.

Resonance means that an impulse causes "ringing," in which energy is propagated back and forth between inductance and capacitance until it finally dissipates in circuit resistance or is radiated away, but the resulting signal from many devices may appear as increased system noise.

Resonance would actually seem to be the ideal bypass situation, in that a resonant bypass presents the minimum impedance to ground. But it does that only at one frequency; lower and higher frequencies are less rejected. It seems quite impractical to tune for resonance with the "random" pulses occurring in complex logic. And, above resonance, inductance dominates and then higher-frequency noise and pulses are more able to affect the rest of the system.

Another approach has been to use various bypass capacitors, typically 0.01uF and 0.1uF in parallel, "sprinkled around" the PC layout. The idea was that self-resonance in any one bypass capacitor would be hidden by the other capacitor of different value and, thus, different resonant frequency. Alone, either a 0.01uF or a 0.1uF cap may do an effective job. However, recent modeling indicated, and experimentation has confirmed, that using both together can be substantially *worse* than using either value alone.

The inherent limitation in bypassing is that the normal bypass process is not "lossy" or dissipative. Pulse energy can be stored in the inductance of short leads or PC-board traces, and then "ring" in resonance with the usual ceramic bypass capacitors. Having many bypass caps often leads to complex RF filter-like structures which just pass the ringing energy around. An alternative is the wide use of tantalum bypass capacitors, since tantalum becomes increasingly lossy at higher frequencies and will dissipate pulse energy.

Several approaches seem reasonable:

- Use one value and type of small bypass capacitor throughout.
- Use more tantalum capacitors, which have a surprisingly good high frequency bypass capability as well as considerable power storage.
- Use ferrite beads to isolate individual chips or small groups of chips from the shared power distribution. This would introduce a few ohms of resistance in distribution leads at high frequencies only. Resonance effects might still exist, but would be simplified, localized and isolated from the rest of the system. Related isolationist approaches have a long history in radio technology, where it is commonly called decoupling.

Byte

A collection of eight bits. Also called an "octet." A byte can represent 256 different values or symbols. The common 7-bit ASCII codes used to represent characters in computer use generally are stored in a byte; that is, one byte per character.

C

In math notation, the complex numbers.

CA

Certification authority.

Capacitor

A basic electronic component which acts as a reservoir for electrical power in the form of voltage. A capacitor acts to "even out" the voltage across its terminals, and to "conduct" voltage changes from one terminal to the other. A capacitor "blocks" DC and conducts AC in proportion to frequency. Capacitance is measured in Farads: A current of 1 Amp into a capacitance of 1 Farad produces a voltage change of 1 Volt per second across the capacitor.

If we know the capacitance C in Farads and the frequency f in Hertz, the capacitive reactance X_C in Ohms is:

```
XC = 1 / (2 Pi f C)
Pi = 3.14159...
```

Capacitors in parallel are additive. Two capacitors in series have a total capacitance which is the product of the capacitances divided by their sum.

A capacitor is typically two conductive "plates" or metal foils separated by a thin insulator or *dielectric*, such as air, paper, or ceramic. An electron charge on one plate attracts the opposite charge on the other plate, thus "storing" charge. A capacitor can be used to collect a small current over long time, and then release a high current in a short pulse, as used in a camera strobe or "flash."

The simple physical model of a component which is a simple capacitance and nothing else works well at low frequencies and moderate impedances. But at RF frequencies and modern digital rates, there is no "pure" capacitance. Instead, each capacitance has a series inductance that often does affect the larger circuit. See bypass.

Also see inductor and resistor.

Cardinal

A count; the number of items in a group. Also see: ordinal, alphabet, population, and universe.

Cartesian Product

The Cartesian product A x B is the set of all ordered pairs (a,b) where the first component is taken from set A, and the second component is taken from set B.

Cascade

The general concept of a sequence of stages: first one, then another, then another. In electronics, a sequence of operations or components. This usage long precedes the current use in cryptography of cascade ciphering.

Cascade Ciphering

Generally speaking, multiple encryption, product ciphering or multiple ciphering. (Also see cascade.) Some sources define cascade ciphering as implying the use of statistically independent keys, whereas product ciphers may use keys which are *not* independent. Surprisingly, neither usage is consistent with the original definitions:

The earliest definition of "cascade cipher" I know (1983) does not mention key independence:

> "A *Cascade Cipher* (CC) is defined as a concatenation of block cipher systems, thereafter referred to as its stages; the plaintext of the CC plays the role of the plaintext of the first stage, the ciphertext of the i-th stage is the plaintext of the (i+1)-st stage and the ciphertext of the last stage is the ciphertext of the CC.
>
> "We assume that the plaintext and ciphertext of each stage consists of *m* bits, the key of each stage consists of *k* bits and there are *t* stages in the cascade."
>
> *[Note the lack of the term "independent."/tfr]*
>
> -- Even, S. and O. Goldreich. 1983. "On the power of cascade ciphers." *Advances in Cryptology: Proceedings of Crypto '83.* 43-50.

A modern academic definition is:

> "[The] Product of several ciphers is also a product cipher, such a design is sometimes called a *cascade* cipher."
>
> *[Note the lack of anything like "with independent keys."/tfr]*
>
> -- Biryukov, Alex. (Faculty of Mathematics and Computer Science, The Weizmann Institute of Science.) 2000. *Methods of Cryptanalysis*. "Lecture 1. Introduction to Cryptanalysis."

Similarly, the term "product encipherment" is defined in Shannon 1949 (and is quoted here under Algebra of Secrecy Systems) as the use of one cipher, then another *with independent keys*. Thus, the independent key terminology was defined in cryptography over half a century ago, and probably 34 years before "cascade ciphering" was defined for the same idea *without* the key independence requirement. Both terms are commonly and legitimately confused in use. Anyone using the terms "cascade ciphering" or "product ciphering" would be well advised to explicitly state what the term is supposed to mean, or to not complain when someone takes it to mean something else.

Cathode

In an electronic circuit element, the end which accepts electrons and sources conventional current flow. The (+) end of a charged battery. The (-) end of a battery being charged. The N side of a PN semiconductor junction. As opposed to anode.

CBC

The Cipher Block Chaining block cipher operating mode.

c.d.f.

In statistics, *cumulative distribution function.* A function which gives the probability of obtaining a particular value or lower.

Certify

To attest to something being genuine. Also see authentication.

Certification Authority

CA. The third party needed by public key systems to certify public keys; to assure that the public key being used actually belongs to the desired party instead of a man-in-the-middle. Also see public key infrastructure.

CFB

The Ciphertext Feedback operating mode.

Chain

An operation repeated in a sequence, such that each result depends upon the previous result, or an initial value. One example is the CBC operating mode.

Particularly inappropriate as a description of multiple encryption, because a physical "chain" is is only as strong as the weakest link, while a sequence of ciphers is as strong as the strongest link.

Chance

Something which happens unpredictably.

Chaos

The unexpected ability to find numerical relationships in physical processes formerly considered random. Typically these take the form of iterative applications of fairly simple computations. In a chaotic system, even tiny changes in state eventually lead to major changes in state; this is called "sensitive dependence on initial conditions." It has been argued that every good computational random number generator is "chaotic" in this sense.

In physics, the state of an analog physical system cannot be fully measured, which always leaves some remaining uncertainty to be magnified on subsequent steps. And, in many cases, a physical system may be slightly affected by thermal noise and thus continue to accumulate new information into its state.

In a computer, the state of the digital system is explicit and complete, and there is no uncertainty. No noise is accumulated. All operations are completely deterministic. This means that, in a computer, even a "chaotic" computation is completely predictable and repeatable.

Characteristic

The characteristic of a finite field is the number of times the multiplicative identity must be added to itself to produce zero. The characteristic will be some prime number *p* or, if the summation will never produce zero, the characteristic is said to be zero.

Checkerboard Construction

Finding or building a large number of different keyed Latin squares, and especially orthogonal Latin squares, can be extremely difficult. Although some simple constructions are available for statistical use, few of those support making huge numbers of essentially random squares. One solution which appears to be new to cryptography is what I call the *Checkerboard Construction*:

One way to construct a larger square is to take some Latin square and replace each of the symbols or elements with a full Latin square. By giving the replacement squares different symbol sets, we can arrange for symbols to be unique in each row and column, and so produce a Latin square of larger size.

If we consider squares with numeric symbols, we can give each replacement square an offset value, which is itself determined by a Latin square. We can obtain offset values by multiplying the elements of a square by its order:

```
   0 1 2 3            0  4  8 12
   1 2 3 0   * 4 =    4  8 12  0
   2 3 0 1            8 12  0  4
   3 0 1 2           12  0  4  8
```

To simplify the example, we can use the same original square for all of the replacement squares:

```
   0+ 0 1 2 3    4+ 0 1 2 3    8+ 0 1 2 3   12+ 0 1 2 3
      1 2 3 0       1 2 3 0       1 2 3 0       1 2 3 0
      2 3 0 1       2 3 0 1       2 3 0 1       2 3 0 1
      3 0 1 2       3 0 1 2       3 0 1 2       3 0 1 2

   4+ 0 1 2 3    8+ 0 1 2 3   12+ 0 1 2 3    0+ 0 1 2 3
      1 2 3 0       1 2 3 0       1 2 3 0       1 2 3 0
      2 3 0 1       2 3 0 1       2 3 0 1       2 3 0 1
      3 0 1 2       3 0 1 2       3 0 1 2       3 0 1 2

   8+ 0 1 2 3   12+ 0 1 2 3    0+ 0 1 2 3    4+ 0 1 2 3
      1 2 3 0       1 2 3 0       1 2 3 0       1 2 3 0
      2 3 0 1       2 3 0 1       2 3 0 1       2 3 0 1
      3 0 1 2       3 0 1 2       3 0 1 2       3 0 1 2

  12+ 0 1 2 3    0+ 0 1 2 3    4+ 0 1 2 3    8+ 0 1 2 3
      1 2 3 0       1 2 3 0       1 2 3 0       1 2 3 0
      2 3 0 1       2 3 0 1       2 3 0 1       2 3 0 1
      3 0 1 2       3 0 1 2       3 0 1 2       3 0 1 2
```

which produces the order-16 square:

```
   0  1  2  3     4  5  6  7     8  9 10 11    12 13 14 15
   1  2  3  0     5  6  7  4     9 10 11  8    13 14 15 12
   2  3  0  1     6  7  4  5    10 11  8  9    14 15 12 13
   3  0  1  2     7  4  5  6    11  8  9 10    15 12 13 14

   4  5  6  7     8  9 10 11    12 13 14 15     0  1  2  3
   5  6  7  4     9 10 11  8    13 14 15 12     1  2  3  0
   6  7  4  5    10 11  8  9    14 15 12 13     2  3  0  1
   7  4  5  6    11  8  9 10    15 12 13 14     3  0  1  2

   8  9 10 11    12 13 14 15     0  1  2  3     4  5  6  7
   9 10 11  8    13 14 15 12     1  2  3  0     5  6  7  4
  10 11  8  9    14 15 12 13     2  3  0  1     6  7  4  5
  11  8  9 10   15 12 13 14    3  0  1  2    7  4  5  6

  12 13 14 15    0  1  2  3    4  5  6  7    8  9 10 11
  13 14 15 12    1  2  3  0    5  6  7  4    9 10 11  8
  14 15 12 13    2  3  0  1    6  7  4  5   10 11  8  9
  15 12 13 14    3  0  1  2    7  4  5  6   11  8  9 10
```

Clearly, this Latin square exhibits massive structure at all levels, but this is just a simple example. In practice we would create and use a *different* order-4 table for each position, and yet another for the offsets. We would also shuffle all rows, columns, and symbols in the larger square. And we could use order-16 squares to construct a keyed square of order-256. The result is a balanced table in a difficult to predict arrangement, a distinct selection from among a plethora of similar tables, and, thus, apparently ideal for cryptographic use as a Latin square combiner.

Keyspace

There are 576 Latin squares of order 4, any one of which can be used as any of the 16 replacement squares. The offset square is another order-4 square. So we can construct 576^{17} (about 8 x 10^{46} or 2^{155}) different squares of order 16 in this way. Then we can shuffle the resulting square in 16! * 15! (about 2 x 10^{25} or 2^{84}) different ways, thus producing about 2 x 10^{72} squares, for about 240 bits of keying per square. (Even if we restrict ourselves to using only the 144 order 4 squares formed by shuffling a single standard square, we still have a 206-bit keyspace.) We could store two of the resulting order 16 "4 bit" squares in a 256-byte table for use as a "pseudo-8 bit" combiner, and might even select a combiner dynamically from an array of such tables.

The construction is also applicable to orthogonal Latin squares. See my articles:

- "Practical Latin Square Combiners," locally, or @: http://www.ciphersbyritter.com/ARTS/PRACTLAT.HTM
- "Orthogonal Latin Squares, Nonlinear Balanced Block Mixers, and Mixing Ciphers," locally, or @: http://www.ciphersbyritter.com/ARTS/NONLBBM.HTM

Checksum

An early and simple form of error detecting code. Commonly, an actual summation of data values in a register of some reasonable size like 16 bits. Unfortunately, addition is not particularly effective in detecting errors: In one massive set of experiments with real data, a 16-bit checksum was shown to detect errors about as well as a 10-bit CRC.

Improved checksums (e.g., Fletcher's checksums) include both data values and data positions and may perform within a factor of 2 of CRC. One advantage of a true summation checksum is a minimal computation overhead in software (in hardware, a CRC is almost always smaller and faster). Another advantage is that when header values are changed in transit, a summation checksum is easily updated, whereas a CRC update is more complex and many implementations will simply re-scan the full data to get the new CRC.

The term "checksum" is sometimes applied to any form of error detection, including more sophisticated codes like CRC.

Chi-Square

In statistics, a goodness of fit test used for comparing two distributions. Mainly used on nominal and ordinal measurements. Also see: Kolmogorov-Smirnov, one-sided test, and two-sided test.

In the usual case, "many" random samples are counted by category or separated into value-range "bins." The reference distribution gives us the the number of values to expect in each bin. Then we compute a X^2 test statistic related to the difference between the distributions:

```
X2 = SUM( SQR(Observed[i] - Expected[i]) / Expected[i] )
```

("SQR" is the squaring function, and we require that each expectation not be zero.) Then we use a tabulation of chi-square statistic values to look up the probability that a particular X^2 value or lower (in the c.d.f.) would occur by random sampling if both distributions were the same. The statistic also depends upon the "degrees of freedom," which is almost always one less than the final number of bins. See the chi-square section of the "Normal, Chi-Square and Kolmogorov-Smirnov Statistics Functions in JavaScript" page (locally, or @: http://www.ciphersbyritter.com/JAVASCRP/NORMCHIK.HTM#ChiSquare).

The c.d.f. percentage for a particular chi-square value is the area of the statistic distribution to the left of the statistic value; this is the probability of obtaining that statistic value *or less* by random selection when testing two distributions which are exactly the same. Repeated trials which randomly sample two identical distributions should produce about the same number of X^2 values in each quarter of the distribution (0% to 25%, 25% to 50%, 50% to 75%, and 75% to 100%). So if we repeatedly find only very high percentage values, we can assume that we are probing different distributions. And even a single very high percentage value would be a matter of some interest.

Any statistic probability can be expressed either as the proportion of the area to the *left* of the statistic value (this is the "cumulative distribution function" or c.d.f.), or as the area to the *right* of the value (this is the "upper tail"). Using the upper tail representation for the X^2 distribution can make sense because the usual chi-squared test is a "one tail" test where the decision is always made on the upper tail. But the "upper tail" has an opposite "sense" to the c.d.f., where higher statistic values always produce higher percentage values. Personally, I find it helpful to describe all statistics by their c.d.f., thus avoiding the use of a wrong "polarity" when interpreting any particular statistic. While it is easy enough to convert from the c.d.f. to the complement or vise versa (just subtract from 1.0), we can base our arguments on either form, since the statistical implications are the same.

It is often unnecessary to use a statistical test if we just want to know whether a function is producing something like the expected distribution: We can *look* at the binned values and generally get a good idea about whether the distributions change in similar ways at similar places. A good rule-of-thumb is to expect chi-square totals similar to the number of bins, but distinctly different distributions often produce huge totals far beyond the values in any table, and computing an exact probability for such cases is simply irrelevant. On the other hand, it can be very useful to perform 20 to 40 independent experiments to look for a reasonable statistic distribution, rather than simply making a "yes / no" decision on the basis of what might turn out to be a rather unusual result.

Since we are accumulating *discrete* bin-counts, any fractional expectation will always differ from any actual count. For example, suppose we expect an even distribution, but have many bins and so only accumulate enough samples to observe about 1 count for every 2 bins. In this situation, the absolute best sample we could hope to see would be something like (0,1,0,1,0,1,...), which would represent an even, balanced distribution over the range. But even in this best possible case we would still be off by half a count in each and every bin, so the chi-square result would not properly characterize this best possible sequence. Accordingly, we need to accumulate enough samples so that the quantization which occurs in binning does not appreciably affect the accuracy of the result. Normally I try to expect at least 10 counts in each bin.

But when we have a reference distribution that trails off toward zero, *inevitably* there will be some bins with few counts. Taking more samples will just expand the range of bins, some of which will be lightly filled in any case. We can avoid quantization error by summing both the observations and expectations from multiple bins, until we get a reasonable expectation value (again, I like to see 10 counts or more). This allows the "tails" of the distribution to be more properly (and legitimately) characterized. (The technique of merging adjacent bins is sometimes called "collapsing.")

Chosen Plaintext

Defined plaintext.

Cipher

1. Any system which uses encryption; a cipher system.
2. A key-selected secret transformation between plaintext and ciphertext. A key-selected function which takes plaintext to ciphertext, and an inverse function which takes that ciphertext back to the original plaintext.
3. Specifically, a secrecy mechanism or process which operates on individual characters or bits independent of semantic content. As opposed to a secret code, which generally operates on words, phrases or sentences, each of which may carry some amount of complete meaning.

Also see: cryptography, block cipher, stream cipher, substitution, permutation, cipher system, a cipher taxonomy, cryptanalysis, attack, security and strength.

A good cipher can transform secret information into a multitude of different intermediate forms, each of which represents the original information. *Any* of these intermediate forms or ciphertexts can be produced by ciphering the information under some key value. The intent is that the original information only be exposed by *one* of the many possible keyed interpretations of that ciphertext. Yet the correct interpretation is available merely by deciphering under the appropriate key.

A cipher appears to reduce the protection of secret information to enciphering under some key, and then keeping that key secret. This is a great reduction of effort and potential exposure, and is much like keeping your valuables in your house, and then locking the door when you leave. But there are also similar limitations and potential problems.

With a good cipher, the resulting ciphertext can be stored or transmitted otherwise exposed without also exposing the secret information hidden inside. This means that ciphertext can be stored in, or transmitted through, systems which have no secrecy protection. For transmitted information, this also means that the cipher itself must be distributed in multiple places, so in general the cipher cannot be assumed to be secret. With a good cipher, only the deciphering key need be kept secret. (See: Kerckhoffs' requirements, but also security through obscurity.)

Note that a cipher does not, in general, hide the *length* of a plaintext message, nor the fact that the message exists, nor when it was sent, nor, usually, the addressing to whom and from whom. Thus, even the *theoretical* one time pad (often said to be "proven unbreakable") *does* expose *some* information about the plaintext message. If message length is a significant risk, random amounts of padding can be added to confuse that, although padding can of course only *increase* message size, and is an overhead to the desired communications or storage. This typically would be handled at a level outside the cipher design proper, see cipher system.

Cipher Engineering

It is important to understand that **ciphers are unlike any other modern product design**, in that we cannot know when a cipher "works." For example:

- A bridge is designed with beams of known strength. The strength of the resulting structure can be simulated and computed. When built, we can roll something heavy across and see that the bridge "works." Over time, we can develop trust that the bridge will work (as a bridge) when we need it.
- A car "works" when it moves, and over time we can develop trust that it will move when and how we want. We *know* when it does not work.
- A typical computer program does something and we can see the results, so we know if what we want actually occurs. Over time, we can build trust that the program will do what we want.
- With a medicine, if we cannot see (or show) that it is working for us, we switch to something else.

Ciphers are like *none* of those things because an opponent may break the cipher and not bother to tell us: we simply cannot know when a cipher "works." Since we do not know if a cipher is keeping our secrets safe, repeated use cannot build trust. And since we cannot see the outcome, we cannot know how to design a cipher to "guarantee" (in any sense at all) that the result will do what we need. In industrial quality-control terms, cipher design is literally "out of control." Also see: scientific method.

Cipher Block Chaining

(CBC). Cipher Block Chaining is an operating mode for block ciphers. CBC mode is essentially a crude meta-stream cipher which streams block-size transformations. As a consequence, the resulting meta-cipher is not a bijection like the block cipher component.

In CBC mode the ciphertext value of the preceding block is exclusive-OR combined with the plaintext value for the current block. This randomization has the effect of distributing the resulting block values evenly among all possible block values, and so tends to prevent codebook attacks. But ciphering the *first* block generally requires an IV or initial value to start the process. And the IV necessarily expands the ciphertext by the size of the IV.

[There are various possibilities other than CBC for avoiding plaintext block statistics in ciphers. One alternative is to pre-cipher, presumably with a different cipher and key, thus producing randomized plaintext blocks (see multiple encryption). Another alternative is to use a block at least 64 bytes wide, which, if it contains language text, can be expected to contain sufficient unknowable randomness to avoid codebook attacks (see huge block cipher advantages).]

Note that the exposed nature of the CBC randomizer (the previous block ciphertext) does not hide plaintext or plaintext statistics. When simple deciphering exposes plaintext, the vast majority of possible plaintexts can be rejected automatically, based on their lack of bit-level and character and word structure. Normal CBC does not improve this situation much at all.

CBC First Block Problems

In CBC mode, each randomizing value is the ciphertext from each previous block. Clearly, all the ciphertext is exposed to the opponent, so there would seem to be little benefit associated with hiding the IV, which, after all, is just the first of these randomizing values. Clearly, in the usual case, if the opponent makes changes to a ciphertext block in transit, that will hopelessly garble two blocks (or perhaps just one) of the recovered plaintext. As a result, it is very unlikely that an opponent could make systematic changes in the plaintext simply by changing the ciphertext.

But the IV is a *special case*: if the IV is not enciphered, and if the opponents can intercept and change the IV in transit, they can change the first-block plaintext bit-for-bit, *without* a block-wide garble. That means an opponent *could* make systematic changes to the first block plaintext simply by changing the IV. So, if the opponents know the first block plaintext (which could be a logo, name, date, or fixed dollar value), the stage is set for a potentially serious man-in-the-middle (MITM) problem. (The far more serious public-key MITM problem is an authentication failure with respect to the *key*, not an IV or data, and is a completely different issue.) Note that the CBC first block problem is completely independent of the cipher key and whether or not it changes, and is an even larger problem with the modern wider blocks used in AES.

CBC First Block Solutions

Despite howls of protest to the contrary, it is easy to see that the CBC first-block problem is a confidentiality problem, *not* an authentication problem. To see this, we simply note that all that is necessary to avoid the problem is to keep the IV secret. When the IV is protected, the opponent cannot know which changes to make to reach a desired plaintext. And, since the problem can be fixed without any authentication at all, it is clear that the problem was not a lack of authentication in the first place. Instead, the problem was caused by exposing the IV, and solving that is the appropriate province of the CBC and block level, instead of a MAC at the cipher system and message level.

To fix the CBC first-block problem it is *not* necessary to check the plaintext for changes by using a MAC. Nor is a MAC necessarily the only way to authenticate a message. But if we are going to use a MAC anyway, that is one way to solve the problem. That works because a MAC can detect the systematic changes which a lack of confidentiality may have allowed to occur. But if a MAC is not otherwise desired, introducing a MAC to solve the CBC first-block problem is probably overkill, because only the block-wide IV needs to be protected, and not the entire message.

The reason we might not want to use a MAC is that a MAC carries some inherent negative consequences. One of those is a processing latency, in that we cannot validate the recovered plaintext until we get to the end and check the digest. Latency can be a serious problem with streaming data like audio and video, and with interactive protocols. But even with an email message we have to buffer the whole message as decrypted and wait for the incoming data to finish before we can do anything with it (or we can make encryption hard and decryption easy, but one side will be a problem). Or we can set up some sort of packet structure with localized integrity checks and ciphertext expansion in each packet. But that seems like a lot of trouble when an alternative is just to encipher the IV.

Even when a MAC is used at a higher level anyway, it may be important for Software Engineering and modular code construction to handle at the CBC level as many of the problems which CBC creates as possible. This avoids forcing the problem on, and depending upon a correct response from, some unknown programmer at the higher level, who may have other things on the mind. Handling security problems where they occur and not passing them on to a higher layer is an appropriate strategy for security programming.

As the problems compound themselves, it seems legitimate to point out that the CBC first-block problem is a CBC-level security issue caused by CBC and by transporting the IV in the open. The CBC first-block problem is easily prevented simply by transporting the IV securely, by encrypting the IV before including it with the ciphertext. Also see "The IV in Block Cipher CBC Mode" conversation (locally, or @: http://www.ciphersbyritter.com/NEWS6/CBCIV.HTM).

Ciphering

The use of a cipher. The general term which includes both enciphering and deciphering.

Cipher System

A larger system which includes one or more ciphers and generally involves much more, including:

- **Key Management.** Facilities to support secure key loading, user storage, use, re-use, archival storage, loss, and destruction. Also possibly facilities for secure key creation and transport encryption. Key storage may involve creating a password-encrypted mini-database of actual key values selected by open alias (see: alias file). Key transport may involve a full public key component, with its own key construction, storage, use, loss and destruction reqirements, especially including some form of cryptographic key certification, and possibly including a large, complex, and expensive certification infrastructure. (Also see hybrid.) Also see the Cloak2 documents for implemented key management features (locally, or @: http://www.ciphersbyritter.com/CLO2FEA.HTM) and alias file usage (locally, or @: http://www.ciphersbyritter.com/PROD/CLO2DOC3.HTM#DetAF).
- **Keyphrase Hashing.** In general, language phrases must be converted to randomized and dense binary values for cipher use. In general, a common hash such as CRC is sufficient for the task, and a cryptographic hash is not required.
- **Message Key Creation and Use.** In general, a random value is used as the key for the actual data. Typically, the random message key value will be the only thing encrypted by the key from the alias file, or the public key component.
- **Unknowable Randomness Creation or Collection.** For message keys, other keys and possibly other protocols.
- **Message Integrity Coding.** Detect when a message has changed in transit. This could be a MAC, or

even a simple hash protected by a conventional block cipher.

- **Cipher Selection.** A cipher system need not always use the same cipher, and a multiple encryption system may use different ciphers in different sequences. A dynamic selection protocol.
- **Data Compression.** Messages often can be compressed, leading to shorter messages and apparently more-complex plaintext. (But if the compression method is known and unkeyed, compression may not add much cryptographic advantage.) Data compression may help obscure the original message length, however.
- **Message Length and Position Concealment.** In general, ciphers conceal message data, not message length. That can be a serious security hazzard which may need to be addressed. Random amounts of random data (e.g., nulls) can be added to the top and bottom of a message, or even inside the data itself. Null positions can be keyed and then might essentially constitute another cipher layer in a multiple encryption system.
- **Data Ciphering.** The actual ciphers themselves, keyed by the transported and decrypted message key.
- **Data Length Limitation and Re-Keying.** Most ciphers can securely handle only some maximum amount of data before the keying must change to preserve security. For example, if an opponent collects a codebook of known-plaintext ciphertexts, we can expect that to become useful at something like the square-root of the number of different block values (see birthday attack). So a 64-bit block cipher probably should be limited to ciphering 2**32 blocks or less under a single key. Since various academic attacks on DES need something like 2**47 known-plaintexts, simply limiting the amount of data processed under one key prevents those attacks. Indeed, *many* academic attacks require a huge amount of known plaintext or even defined plaintext ciphered under a single key. By supporting automatic re-keying, the cipher system can assure that the required message volume simply cannot exist.
- **IV Creation and Use.** Most operating modes for conventional block ciphers need an IV, which then (usually) needs to be protected and made part of the ciphertext.
- **Block Partitioning and End Handling.** Messages of arbitrary length need to be partitioned into the fixed-size blocks used by conventional block ciphers, and any remaining partial-block of data padded into a full block or otherwise handled. It is important that any such solution handle 1-byte messages.
- **Secure File Overwrite.** Operating systems generally do not erase files, but instead simply make the deleted file space available for use. It is possible to read "deleted" data from a disk until the released sectors are actively overwritten.
- **Secure Memory Overwrite.** Memory buffers holding keys and data should be cleared by active overwrite as soon as their use is complete.
- **Secure Plaintext Creation and Display.** Unless plaintext is "born secure," the operating system is going to be a serious security problem. A multitasking "swap file" is often a major security risk.

Also see traffic analysis, Software Engineering, Structured Programming, and comments in the "Cipher Review Service" document (locally, or @: http://www.ciphersbyritter.com/CIPHREVU.HTM).

A Cipher Taxonomy

Taxonomy is classification or grouping by common characteristics, instead of by name, development path, or surface similarity. Developing a useful taxonomy is one of the goals and roles of science. The advantage of a taxonomy is that we can study general concepts which then apply to the many different things which fit in a single group. We can also compare and contrast different groups and their effects on the things we study. Ideally, a taxonomy will support future developments without major changes in structure. One way to do that is to start with a dichotomy, which separates the universe of all possibilities into exactly two groups. Not only will all *known* things fit into one or the other of those groups, but all *future* developments also will fit in those same groups. Fairly quickly, though, we must turn to enumeration to describe distinct sub-classes.

For the analysis of cipher operation it is useful to collect ciphers into groups based on their functioning (or *intended* functioning). The goal is to group ciphers which are *essentially similar,* so that as we gain an understanding of one cipher, we can apply that understanding to others in the same group. We thus classify *not* by the components which make up the cipher, but instead on the "black-box" operation of the cipher itself.

We seek to hide distinctions of size, because *operation* is independent of size, and because size effects are usually straightforward. We thus classify conventional block ciphers as keyed simple substitution, just like newspaper amusement ciphers, despite their obvious differences in strength and construction. This allows us to compare the results from an ideal tiny cipher to those from a large cipher construction; the grouping thus can provide *benchmark* characteristics for measuring large cipher constructions.

We *could* of course treat each cipher as an entity unto itself, or relate ciphers by their dates of discovery, the tree of developments which produced them, or by known strength. But each of these criteria is more or less limited to telling us "this cipher is what it is." We already know that. What we *want* to know is what other ciphers function in a similar way, and then whatever is known about *those* ciphers. In this way, every cipher need not be an island unto itself, but instead can be judged and compared in a related community of similar techniques.

Our primary distinction is between ciphers which handle all the data at once (block ciphers), and those which handle some, then some more, then some more (stream ciphers). We thus see the usual repeated use of a block cipher as a stream *meta-cipher* which has the block cipher as a component. It is also possible for a stream cipher to be re-keyed or re-originate frequently, and so appear to operate on "blocks." Such a cipher, however, would not have the overall diffusion we normally associate with a block cipher, and so might usefully be regarded as a stream meta-cipher with a stream cipher component.

The goal is not to give each cipher a label, but instead to seek insight. Each cipher in a particular general class carries with it the consequences of that class. And because these groupings ignore size, we are free to generalize from the small to the large and so predict effects which may be unnoticed in full-size ciphers.

A. BLOCK CIPHER
   - Requires the accumulation of more than one data element for ciphering.
   - Individual data elements cannot be ciphered independently.
   - Real-time information of dynamic size may mean waiting until more information completes a previous block.
   - (Sometimes stream ciphers accumulate data for convenience, as in cylinder ciphers, but nevertheless logically cipher each character independently.)
   - Probably will require padding to fill out a full block when no more data remains, which is ciphertext expansion.
   - Probably will require some added information to allow padding to be distinguished from data and removed, which is more expansion.
   - Generally reveals message length, with an uncertainty only of block size.

   - **SIMPLE SUBSTITUTION**
      - Examples: Newspaper cryptograms, grade school ciphers, codebooks, and conventional block ciphers in electronic codebook mode.
      - A full bijective, invertible, non-expanding mapping.
      - Keying constitutes a permutation or re-arrangement of the table of code values.
      - Since the multiple data elements of a block combine to select a table entry, the smallest possible change to any one of those data elements should select a new and apparently random value, and thus "affect" the full ciphertext block. This is avalanche.
      - Binary-oriented simple substitution distributes bit-changes between all code values binomially, and this effect can be sampled and examined statistically, for a simple substitution signature.
      - Avalanche is two-way diffusion in the sense that "later" plaintext can change "earlier" ciphertext, within a single block, which is also a signature.
      - A conventional block cipher is intended to *simulate* a keyed substitution table of a size which must be vastly larger than anything which could be practically realized. At issue is the quality of that simulation.

- Few conventional block cipher designs are scalable to toy size which would support exhaustive testing.
- Any substitution table becomes weak when its code values are re-used.
- Code value re-use can be minimized by randomizing the plaintext block (e.g., CBC). This distributes the plaintext evenly across the possible block values, but at some point the transformation itself must change or be exposed. And open randomization does not hide plaintext structure from brute force attack.
- Another alternative is to have a huge block size so that code value re-use is made exceedingly unlikely. A large block also has room for a dynamic keying field which would make code value re-use even more unlikely. (Also see huge block cipher advantages.)

- **TRANSPOSITION CIPHER**
  - Example: Dynamic Transposition.
  - Arguably a special case of Simple Substitution with dynamic keying, but with sufficiently different signatures and characteristics to treat independently.
  - All elements to be ciphered first must be collected; this is the block cipher signature.
  - Avalanche is neither present, nor needed, nor helpful so the Simple Substitution signature does not exist.
  - By itself, transposition has various known weaknesses, which can be avoided by design. But unless that is done, we do not have a serious transposition cipher. As a consequence, we have just one example of the group properties.
  - Secure keying is necessarily dynamic, a block-by-block permutation of plaintext elements.
  - Ciphering by re-ordering elements necessarily requires that not all elements have the same value.
  - Ideally, the data should be value-balanced. Balance can be enforced by pre-coding or scrambler randomization.
  - Best transposition ciphering is by bit, not byte.
  - Blocks can be of variable size.
  - These characteristics combine to allow a plethora of different permutations to produce the exact same ciphertext result, thus obscuring the actual permutation involved. This hides the keying sequence.

- **HOMOPHONIC SUBSTITUTION**
  - No known example.
  - One construction is to have a a "homophonic" field as part of the plaintext block. A random value in that field thus selects a particular ciphertext from the many which each reproduce exactly the same "data" field.
  - Arguably a special case of Simple Substitution, but it is not clear that homophonic ciphers could not be built in other ways.
  - A ciphertext expanding cipher.
  - Multiple ciphertexts map to the same plaintext.
  - Coding redundancy can be used to convey per-block authentication or dynamic keying. (See huge block cipher advantages.)
  - Keying constitutes a permutation or re-arrangement of the fixed set of possible code values, *plus* the dynamic selection of a particular homophonic ciphertext.

- **DYNAMICALLY SELECTABLE BLOCK SIZE**
  - Block size selectable on a real-time block-by-block basis.
  - Block size typically available in powers-of-2 elements, subject to some strength minimum (e.g., 8 bytes).
  - Supports the block size in existing designs, modern larger blocks, and stronger huge

    - blocks, all in exactly the same software executable code.
    - Huge blocks support efficient per-block authentication, homophonic operation and per-block dynamic keying. (See huge block cipher advantages.)
    - Supports real-time information of dynamic size *without* using chaining (e.g., CBC).
    - Can minimize padding expansion with just one block of each smaller block size.
    - Huge blocks can support ECB mode, thus avoiding the need for an IV and associated ciphertext expansion.
    - Hardware implementations can feature a constant execution time per block, making huge blocks much faster per byte than small blocks.
    - Scalable down to toy size for exhaustive testing.
    - My Mixing Ciphers may have introduced this concept to open cryptography (see my Balanced Block Mixing development articles, starting in March, 1994).

- **DYNAMICALLY VARIABLE SIZE BLOCK**
    - Block size variable on a real-time block-by-block basis.
    - Block size variable in one-element units, subject to some strength minimum (e.g., 8 bytes).
    - Supports block size in existing designs as well as stronger huge blocks.
    - Huge blocks support efficient per-block authentication, homophonic operation and per-block dynamic keying. (See huge block cipher advantages.)
    - Supports real-time information of dynamic size without external chaining.
    - Can eliminate block padding expansion in messages of arbitrary length.
    - Huge blocks can support ECB mode, thus avoiding the need for an IV and associated ciphertext expansion.
    - Scalable down to toy size for exhaustive testing.
    - My Variable Size Block Ciphers may have introduced this concept to open cryptography (see the August, 1995 "VSBC Newsgroup Discussion" locally, or @: http://www.ciphersbyritter.com/NEWS/VSBCNEWS.HTM).

B. STREAM CIPHER

- Can (but may choose not to) cipher individual data elements immediately, as they arrive. This is a stream cipher signature, and can be identified by analysis of the design.
- Does not need to fill a block, so does not need block padding.
- Does not need padding, so does not need a padding removal structure.
- The basic concept of streaming does not need a confusion sequence or a RNG generator.
- Generally does reveal message length.
- Does not need data diffusion or avalanche.
- May *have* data diffusion, but if so, it is necessarily one-way (from earlier to later elements). This is a stream cipher indication.
- The simple re-use of a block cipher to cover more data than a single block is a stream meta-cipher.

- **CONFUSION SEQUENCE**
    - The classic character-by-character confusion sequence and simple additive combiner cipher.
    - Keying is the sequence itself, or the state which selects a particuar sequence in an RNG generator.
    - With an unpredictable sequence we have a one time pad.
    - With a pseudorandom confusion sequence (from an RNG) we have a classic Vernam cipher.
    - Unfortunately, an additive combiner immediately exposes the confusion sequence under known plaintext attack, and so contributes no strength at all beyond mere balanced mixing.

        - Since the structure of the RNG is assumed to be known, an exposed confusion sequence supports attempts to reconstruct the RNG state.
        - Stronger combiners include nonlinear and keyable combiners with state, that do not immediately expose the confusion sequence.
        - More complex combiners may imply the need for correspondingly less strong confusion sequences.
        - RNG-based stream ciphers do need a message key or something of similar nature to prevent confusion sequence re-use.
        - Having a message key generally implies that amount of ciphertext expansion.

    - **Autokey**
        - A confusion sequence generator (RNG) which is not closed and "free running," but which is affected by the ciphertext.
        - Ciphertext, or possibly plaintext, modifies the state in the RNG and thus the subsequent keystream.
        - Different messages sent under the same key end up having different confusion sequences without any message key.
        - If ciphertext essentially becomes the entire RNG state, we can create a random-like confusion stream which will re-synchronize after ciphertext data loss. However, data loss is rarely an applications-level issue in modern communications systems.
        - Under known plaintext attack, the common "ciphertext feedback" form exposes both the output from, and the input to, the confusion sequence RNG, which puts a lot of pressure on RNG strength.

- **FILTERING**
    - No known example.
    - The plaintext data are directly processed.
    - No confusion sequence is generated.
    - Presumably some sort of shift register with feedback.
    - Presumably something like a digital filter.

- **MONOALPHABETIC** (e.g., DES CBC)
    - The repeated use of a single, fixed substitution table.
    - Examples include: newspaper cryptograms, grade school ciphers, codebooks, and conventional block ciphers in ECB mode.
    - A substitution becomes weak when its code values are re-used.
    - Code value re-use can be minimized by randomizing the plaintext block (e.g., CBC). This distributes the plaintext evenly across the possible block values, but at some point the transformation itself must change or be exposed. And open randomization does not hide plaintext structure from brute force attack.
    - Another alternative is to use a very large block so that code value re-use is made exceedingly unlikely. A large block also has room for a dynamic keying field which would make code value re-use even more unlikely. (See huge block cipher advantages.)

- **POLYALPHABETIC**
    - The repeated use of multiple fixed substitution tables with an implicit or rotating confusion sequence.
    - By itself, the use of a known number of multiple alphabets in a regular sequence is not much stronger than a single alphabet.
    - It is of course possible to select an alphabet at pseudo-random, for example by re-keying DES after every block ciphered. This requires an RNG and an IV to select the starting state. Re-keying DES will take some time, however.

    - An improvement over random alphabets is the use of a Latin square combiner which effectively selects among a balanced set of different fixed substitution alphabets.

  - **Cylinder**
    - A cipher which has or simulates multiple alphabet disks on a single rod. The plaintext message is entered one letter per disk by turning each disk so the correct letter shows in a particular row. The ciphertext is read off some other row.
    - Although operation typically occurs in "chunks" which fill up the cylinder, a full block is not required and individual characters can be ciphered. This is the stream cipher signature.
    - Primary keying is the arrangement of the alphabet around each disk, and the selection and arrangement of disks on the rod.
    - By entering the plaintext on one row, any of n-1 other rows can be sent as ciphertext; this selection is an IV.
    - If the plaintext data are redundant, it is possible to avoid sending the IV by selecting the one of n-1 possible decipherings which shows redundancy. But this is not generally possible when ciphering arbitrary binary data.
    - If an IV is selected first, each character ciphering in that "chunk" is independent of each other ciphering. There is no data diffusion.
    - In general, each disk is used at fixed periodic intervals through the text, which is weak.
    - The ciphertext selection is homophonic, in the sense that different ciphertext rows each represent exactly the same plaintext.
    - Cylinder operation is **not** polyphonic in the usual sense: While a single ciphertext *can* imply any other row is plaintext, generally only one row has a reasonable plaintext meaning.

- **DYNAMIC**
  - The use of one (monoalphabetic) or multiple (polyalphabetic) substitution tables whose contents *change* during ciphering.
  - Keying constitutes a starting permutation or arrangement of the table of code values.
  - An explicit substitution table which is efficiently re-keyed after every use.
  - Used as the combiner part of a conventional confusion sequence stream cipher, or to combine multiple RNG's.
  - Dynamic Substitution is important because it directly confronts the classic attack on conventional stream ciphers: The usual additive combiner immediately exposes the confusion sequence under known plaintext attack. Since the structure of the RNG is assumed to be known, an exposed confusion sequence supports attempts to reconstruct the RNG state. In contrast, a Dynamic Substitution combiner does not expose the confusion sequence, which should make the cipher stronger.

- **ITERATIVE**
  - Multiple encryption using one stream cipher repeatedly with a new random IV on each iteration so as to eventually achieve the effect of a much larger message key.
  - Each iteration seemingly must expand the ciphertext by the size of the IV, although this is probably about the same expansion we would have with a message key.
  - Unfortunately, each iteration will take some time.
  - Particularly appropriate for a cylinder cipher, as shown by W.T. Shaw.

Cipher Testing

If a cipher was like any other technological construction, we could just test it to see how good it was. Unfortunately, cipher strength is not a measurable engineering quantity, and as a result, strength is simply "out of control" with respect to design and manufacturing quality. However, some basic tests can and should be done to weed out the most obvious problems.

In general, absent special coding for transmission (such as converting full binary into base-64 for email) ciphertext should be "random-like." Accordingly, we can run all sorts of tests to try to find any sort of structure or correlation in the ciphertext, or between plaintext, key, and ciphertext. The many available statistical randomness tests should provide ample opportunity for virtually unlimited testing.

Testing Conventional Block Ciphers

The usual or conventional block cipher is intended to emulate a huge, keyed, substitution table. Mathematically, such a function is a bijection, and the symbols in the table are a permutation. These structures might be measured, at least in theory. But very few conventional block ciphers are scalable to tiny size, and the vast size of a real block cipher allows only statistical sampling.

One obvious issue in block cipher construction is diffusion. If the resulting emulated table really is a permutation, if we change the input value in any way, we expect the number of bits which change in the output to occur in a binomial distribution. In addition, we expect each output bit to have a 50 percent probability of changing. We can measure these things.

Typically, we pick some random input value and cipher to get the result; then we change some bit of the input and get the new result and note which and how many bits changed. One advantage of the binomial distribution is that, as block size increases, the distribution becomes increaingly narrow (for any reasonable probability). Thus, we can hope to peer into tremendously small probabilities, which may be about as much error as we can expect to find.

We also can develop a mean value for each output bit, or analyze a particular bit more closely, looking for correlations between input and output, or between key and output, or between the key and some aspect of the transformation between input and output. We might look at correlations between each key bit and each output bit, or between any combination of key bits versus any combination of output bits and so on. With increasingly large experiments, we can perform increasingly fine statistical analyses.

An issue of at least potential concern is that conventional block cipher designs do not implement a completely keyed transformation , but instead implement only a tiny, tiny fraction of all possible tables of the block size. This opens the possibility of weakness in some form of correlation resulting from a tiny subset of implemented permutations. The issue then becomes one of trying to measure possible structural correlations between the set of implemented permutations and the key, including individual bits, or even arbitrary functions of arbitrary multiple bits. At real cipher size, such measurements will be difficult. Or perhaps knowledge of some subset of the transformation could lead to filling out the rest of the transformation; at real cipher size, this may be very difficult to see.

Block Cipher Scalability

Cipher designs which *are* scalable can be tested at real size when that is useful, or as tiny "toy" versions, when *that* is useful. Naturally, the tiny versions are not intended to be as strong as the real-size versions, nor even to be a useful cipher at that size. One purpose is to support exhaustive correlation testing to reveal structural problems which should be easier to discern in the smaller construction. The goal would be to find fault at the tiny size, and then use that to develop insight leading to a scalable attack. That same insight also should help improve the cipher design.

One advantage of scalability is to support attacks on the same cipher at different sizes. Once we find an attack on a toy-size version, we can measure how hard that approach really is by actually doing it. Then we can scale up the cipher slightly and measure how much the difficulty has increased. That can provide true *evidence* which can be used to extrapolate the strength of the real-size cipher, under the given attack. I see this as vastly more believable information than we have for current ciphers.

Another thing we might do is to measure Boolean function nonlinearity values. This measure at least has the advantage of directly addressing one form of strength: the linear predictability of each key-selected permutation.

Yet another thing we might investigate is the number of keys that are actually different. That is, do any keys produce the same emulated table, and if not, how close are those tables? Can we find any two keys that produce the same ciphertext from the same plaintext? (See population estimation and multiple encryption.)

Testing Stream Ciphers

The conventional stream cipher consists of a keyed RNG or confusion generator and some sort of data and confusion combiner, usually exclusive-OR. Since exclusive-OR has absolutely no strength of its own, the strength of the classic stream cipher depends solely on the RNG. Such testing is a common activity in cryptography, using various available statistical randomness tests. (But recall that many strengthless statistical RNG's do well on such tests.) I particularly recommend runs up/down, because we can develop a useful non-flat distribution of results and then compare that to the theoretical expectation. We can do similar things with birthday tests, which are also useful in confirming the coding efficiency or entropy of really random generators.

Modern stream ciphers with nonlinear combiners (see, for example: Dynamic Substitution) seem harder to test. Presumably we can test the ciphertext for randomness, as usual, yet that would not distinguish between the combiner and the RNG. Possibly we could test the combiner with RNG, and then the RNG separately, and compare distributions. However, it is not clear what sort of tests would provide useful insight to this construction. Alternate suggestions are welcomed.

Ciphertext

The result of encryption. As opposed to plaintext.

Ciphertext contains the same information as the original plaintext, hopefully in a form which cannot be easily understood. Cryptography hides information by transforming a plaintext message into any one of a vast multitude of different ciphertexts, as selected by a key. Ciphertext thus can be seen as a code, in which the exact same ciphertext has a vast number of different plaintext interpretations. As a goal, it should be impractical to know which interpretation represents the original plaintext without knowing the key.

Normally, ciphertext will appear random; the values in the ciphertext should occur in a generally balanced way. Normally, we do not expect ciphertext to compress to a smaller size; that implies efficient coding (also see entropy), but only for the random-like ciphertext. Since the amount of plaintext information in the message may be far smaller, from that point of view the ciphertext coding may be very *in*efficient.

It also may happen that the ciphertext can be encoded inefficiently (perhaps as base-64 for email transmission). Note that such encoding does *not* require distinct steps for ciphering and then encoding: Some ciphers directly produce encoded (and, thus, expanded) ciphertext (see, for example: Penknife). Such ciphertext will be compressible, simply because representing information with a subset of ASCII characters is inherently less efficient than a binary representation. Thus, inefficiently coded ciphertext may well compress, and that does *not* imply weakness in the cipher itself.

Ciphertext Expansion

When the ciphertext is larger than the original plaintext. Also see IV, block code, nulls, Braid, Homophonic Substitution, and Penknife.

Ciphertext expansion is the general situation: Stream ciphers need a message key, and block ciphers with a small block need some form of plaintext randomization, which generally needs an IV to protect the first block. Only block ciphers with a large size block generally can avoid ciphertext expansion, and then only if each block can be expected to hold sufficient uniqueness or entropy to prevent a codebook attack.

It is certainly true that in most situations of new construction a few extra bytes are not going to be a problem. However, in some situations, and especially when a cipher is to be installed into an existing system, the ability to encipher data *without* requiring additional storage can be a big advantage. Ciphering data without expansion supports the ciphering of data structures which have been defined and fixed by the rest of the system, provided only that one can place the cipher at the interface "between" two parts of the system. This is also especially efficient, as it avoids the process of acquiring a different, larger, amount of store for each ciphering. Such an installation also can apply to the entire system, and not require the re-engineering of all applications to support cryptography in each one.

Ciphertext Feedback

1. A way of producing an autokey stream cipher, by using ciphertext values to modify the internal state of the keyed RNG keystream generator.
2. CFB. Ciphertext Feedback is an operating mode for a block cipher.

CFB is closely related to OFB, and is intended to provide some of the characteristics of a stream cipher from a block cipher. CFB generally forms an autokey stream cipher. CFB is a way of using a block cipher to form a random number generator. The resulting pseudorandom confusion sequence can be combined with data as in the usual stream cipher.

CFB assumes a shift register of the block cipher block size. An IV or initial value first fills the register, and then is ciphered. Part of the result, often just a single byte, is used to cipher data, and the resulting ciphertext is also shifted into the register. The new register value is ciphered, producing another confusion value for use in stream ciphering.

One disadvantage of this, of course, is the need for a full block-wide ciphering operation, typically for each data byte ciphered. The advantage is the ability to cipher individual characters, instead of requiring accumulation into a block before processing.

Ciphertext Only

The information condition of an opponent knowing only the ciphertext for an encryption, without any of the related plaintext. Also see known plaintext, defined plaintext and ciphertext only attack.

Ciphertext Only Attack

Any attack which takes place under ciphertext only information conditions. Implicitly, however, the opponent also must know *something* about the plaintext, or it will be impossible to know when a deciphering is correct. Also see known plaintext attack and defined plaintext attack.

In a sense, the idea of a ciphertext-only attack is inherently incomplete. By themselves, symbols and code values have no meaning. So we can have all the ciphertext we want, but unless we can find some sort of structure or relationship to plaintext, we have nothing at all. The extra information necessary to identify a break could be the bit structure in the ASCII code, the character structure of language, or any other known relation. But the ciphertext is *never* enough if we know *absolutely nothing* about the plaintext. It is our knowledge or insight about the plaintext, the statistical structure, or even just the known use of one plaintext concept, that allows us to know when deciphering is correct.

In practice, ciphertext-only attacks typically depend on some error or weakness in the encryption design which somehow relates some aspect of plaintext in the ciphertext. For example, codes that always encrypt the same words in the same way naturally leak information about how often those words are used, which should be enough to identify the plaintext. And the more words identified, the easier it is to fill in the gaps in sentences, and, thus, identify still more words. Modern ciphers are less likely to fall into that particular trap, making ciphertext-only attacks generally more academic than realistic (also see break).

Ciphony
Audio or voice encryption. A contraction of "ciphered telephony."

Circuit
In electronics, the "circular" flow of electrons from a power source, through conductors and components and back to the power source. Or the arrangement of components which allows such flow and performs some function.

Claim
1. In commerce, an assertion of value due.
2. In law, a statement of ownership, as in recovering a lost item, or filing for mining or patent rights. Also see patent claims.
3. In argumentation, a conclusion to be shown correct.

Cleartext
Generally speaking, plaintext. Messages transmitted without encryption or "in the clear." For time-sensitive and transient tactical information, getting the message through as soon as possible may be far more important than secrecy.

Cloak2
A DOS-era second-generation secret key stream cipher which I designed and implemented. Based on the nonlinear Dynamic Substitution combiner technology which I invented, patented (see locally, or @: http://www.ciphersbyritter.com/PATS/DYNSBPAT.HTM), and own. More than just a cipher, a full cryptosystem with various cipher system features:

- Huge 992-bit keyspace.
- Huge keystream Additive RNG with 310,048 bits of state and jitterizer nonlinear isolation.
- Two sequential keyed Dynamic Substitution combinings.
- The second combining is selected on a byte-by-byte basis from among 16 keyed Dynamic Substitution combiners.
- Ciphertext can be concatenated.
- Strong key-management by alias file.
- Ciphers individual files, multiple files, or an entire disk.
- Supports user-programmed batch operations.

See the documentation:

- "Cloak2 Features" (locally, or @: http://www.ciphersbyritter.com/CLO2FEA.HTM)
- "The Cloak2 Cipher User's Manual" (locally, or @: http://www.ciphersbyritter.com/PROD/CLO2DOC3.HTM)
- "Cloak2 Quick Start" (locally, or @: http://www.ciphersbyritter.com/PROD/CLO2QUIC.HTM)
- "The Cloak2 Cipher Design" (locally, or@: http://www.ciphersbyritter.com/CLO2DESN.HTM)

Also see Penknife.

Clock
A repetitive or cyclic timing signal to coordinate state changes in a digital system. A clock coordinates the movement of data and results through various stages of processing. Although a clock signal is digital, the source of the repetitive signal is almost always an analog circuit, typically a crystal oscillator.

In a digital system we create a delay or measure time by simply counting pulses from a stable oscillator. Since counting operations are digital, noise effects are virtually eliminated, and we can easily create accurate delays which are as long as the count in any counter we can build.

Closure

When an operation on a set produces only elements in that set.

Code

From the Latin *codex*, for tree trunk or wax-covered wooden tablet.
1. A list of rules.
2. In cryptography, symbols, colors, shapes, flowers, musical notes, finger positions or values which stand for symbols, values, words, sentences, ideas, sequences, or even operations (as in computer "opcodes"). Often just a simple substitution between numeric values.

Code values can easily represent not only symbols or characters, but also words, names, phrases, and entire sentences (also see nomenclator). In contrast, a cipher operates only on individual characters or bits. Classically, the meaning of each code value was collected in a codebook. Codes may be open (public) or secret.

Coding is a very basic part of modern computation and generally implies no secrecy or information hiding. In modern usage, a code is often simply a correspondence between information (such as character symbols) and values (such as the ASCII code or Base-64). Because a code can represent entire phrases with a single number, one early application for a public code was to decrease the cost of telegraph messages.

In general, secret codes are weaker than ciphers, because a typical code will simply substitute or transform each different word or letter into a corresponding value. Thus, the most-used plaintext words or letters also become the most-used code or ciphertext values and the statistical structure of the plaintext remains exposed. Then the opponent easily can find the most-used ciphertext values and realize that they represent the most-used plaintext words. Accordingly, it is common to superencipher a coded message in an attempt to hide the codebook values.

A meaningful code is more than just data, being also the *interpretation* of that data. The main concept of modern cryptography is the use of a key to select one interpretation from among vast numbers of different interpretations, so that meaning is hidden from those who do not have both the appropriate decryption program and key. Each particular ciphertext is interpreted by the decryption system to produce the desired plaintext. The pairing of *value* plus *interpretation* to produce or do something occurs in various places:

- In cryptography, we have *ciphertext* plus *the key-selected transformation* which interprets that ciphertext.
- In cryptanalysis, we have *pseudorandom sequences*, plus *the finite state machine and state* which creates a particular sequence.
- In computer hardware, we have *opcode values* plus a *computer* which interprets those different values as different operation commands.
- In computer programming, we typically have raw binary *data* plus *procedures* which interpret that data.
- In computer-based documents, we have various *formats* (e.g., .DOC or .PIF files) plus *a program* to properly display those documents.
- In biology we have *DNA* plus *the cell* which interprets DNA, both being required to express the original meaning.

In real life, many useful things do require a particular thing to use them. For example, gasoline provides energy for cars, but only because cars have the appropriate engine to perform the desired conversion. Similarly, bullets require guns, radio broadcasting stations require radios and so on. But that probably reaches beyond the idea of a code, which is basically limited to information- or symbol-oriented transformations.

Codebook

Literally, the listing or "book" of code transformations. More generally, any collection of such transformations. Classically, letters, common words and useful phrases were numbered in a codebook; messages transformed into those numbers were "coded messages." Also see nomenclator. A "codebook style cipher" refers to a block cipher.

Codebook Attack

A form of attack in which the opponent simply tries to build or collect a codebook of all the possible transformations between plaintext and ciphertext (under a single key). This is the classic approach we normally think of as "codebreaking." Also called "bookbreaking."

The usual ciphertext only approach depends upon the plaintext having strong statistical biases which make some values far more probable than others, and also more probable in the context of particular preceding known values. While this is not known plaintext, it *is* a form of *known structure* in the plaintext. Such attacks can be defeated if the plaintext data are randomized and thus evenly and independently distributed among the possible values (see balance).

When a codebook attack is possible on a block cipher, the complexity of the attack is controlled by the size of the block (that is, the number of elements in the codebook) and not the strength of the cipher. This means that a codebook attack would be equally effective against either DES or Triple-DES.

One way a block cipher can avoid a codebook attack is by having a large block size which will contain an unsearchable amount of plaintext "uniqueness" or entropy. Another approach is to randomize the plaintext block, by using an operating mode such as CBC, or multiple encryption. Yet another approach is to change the key frequently, which is one role of the message key introduced at the cipher system level.

Codebreaking

Specifically, the work of attempting to attack and break a secret code. More generally, attempting to defeat any kind of secrecy system.

Codebreaking is what we normally think of when hearing the WWII crypto stories, especially the Battle of Midway, because many secrecy systems of the time were codes. According to the story, the Japanese are preparing an attack on Midway island, and have given Midway the coded designation "AF." American cryptanalysts have exposed the designator "AF," but not what it represents. Assuming the "AF" to be Midway, American codebreakers have Midway falsely report the failure of their fresh-water plant in open traffic. Then, two days later, intercepted Japanese traffic states that "AF" is short of fresh water. Thus, "AF" is confirmed as Midway.

Note that there had to be a way to identify the actual target (plaintext) with the code value (ciphertext) before the meaning was exposed. Simply having the ciphertext itself, without finding structure in the ciphertext or some relationship to plaintext, is almost never enough, see ciphertext-only attack.

Codeword

In coding theory, a particular sequence or block of n elements, each element taken from alphabet A. A particular block-encoded result value.

Coding Theory

The engineering field concerned with encoding data for communications and storage. Different codings have various properties, including:

- Size
- Error-detection
- Error-correction
- Multiple logical channels
- Control codes
- Bit balance

See block code and entropy.

Coefficient

In a mathematical expression, a factor of a term. Typically a constant value or simple variable which multiplies the parameter of interest (often *X*). However, any proper subset grouping of factors technically would be a coefficient of the overall product.

Cognitive Dissonance

The uncomfortable psychological reaction to the experience of finding that some of our core beliefs are factually wrong.

The classic example is of a cult who believed the Earth was going to end at a particular time. Supposedly, many members gave up their houses and jobs and so on, but the Earth did not end. As a consequence, less-involved members generally accepted that their belief was false. But more-involved members instead insisted that the actions of the cult showed their faith, which was then rewarded by the Earth not ending.

Obviously it is difficult to use logic to address issues of faith, but science is not a faith and does not require belief. Therefore, when we find that current scientific positions are wrong, they can be changed with only minor discomfort and anguish. Supposedly. (Also see mere semantics and old wives' tale.)

Combination

In mathematics, the term for any particular subset of symbols, independent of order. (Also called the binomial coefficient.) The number of combinations of *n* things, taken *k* at a time, read "*n* choose *k*" is:

```
 n
( ) = C(n,k) =  n! / (k! (n-k)!)
 k
```

Also,

```
 n     n           n
( ) = ( ) = 1     ( ) = n
 0     n           1
```

See the combinations section of the "Base Conversion, Logs, Powers, Factorials, Permutations and Combinations in JavaScript" page (locally, or @: http://www.ciphersbyritter.com/JAVASCRP/PERMCOMB.HTM#Combinations). Also see permutation.

Combinatoric

In mathematics, combinatorics is related to counting selections, arrangements or other subsets of finite sets. One result is to help us understand the probability of a particular subset in the universe of possible values.

Consider a conventional block cipher: For any given size block, there is some fixed number of possible messages. Since every enciphering must be reversible (deciphering must work), we have a 1:1 mapping between plaintext and ciphertext blocks. The set of all plaintext values and the set of all ciphertext values is the same set; particular values just have different meanings in each set.

Keying gives us no more ciphertext values, it only re-uses the values which are available. Thus, keying a block cipher consists of selecting a particular arrangement or permutation of the possible block values. Permutations are a combinatoric topic. Using combinatorics we can talk about the number of possible permutations or keys in a block cipher, or in cipher components like substitution tables.

Permutations can be thought of as the number of unique arrangements of a given length on a particular set. Other combinatoric concepts include binomials and combinations (the number of unique given-length subsets of a given set).

Combiner

In a cryptographic context, a combiner is a mechanism which mixes two data sources into a single result. A "combiner style cipher" refers to a stream cipher.

Reversible combiners are pretty much required to encipher plaintext into ciphertext in a stream cipher. The ciphertext is then deciphered into plaintext using a related inverse or extractor mechanism. The classic examples are the stateless and strengthless *linear* additive combiners, such as addition, exclusive-OR, etc.

Reversible and ***nonlinear*** **keyable combiners with state** are a result of the apparently revolutionary idea that not all stream cipher security need reside in the keying sequence. Examples include:

- table selection combiner
- Latin square combiner
- Dynamic Substitution combiner

Irreversible or non-invertible combiners are often proposed to mix multiple RNG's into a single confusion sequence, also for use in stream cipher designs. But that is harder than it looks. For example, see:

- Geffe combiner

Also see balanced combiner, complete, and also "The Story of Combiner Correlation: A Literature Survey," locally or @: http://www.ciphersbyritter.com/RES/COMBCORR.HTM.

Common Mode

In electronics, typically an identical signal on both conductors of a balanced line. A transformer winding across that line would ignore common mode signals. As opposed to differential mode.

Commutative

In abstract algebra, a dyadic operation in which exchanging the two argument values must produce the same result: a + b = b + a.

Also see: associative and distributive.

Complete

A term used in S-box analysis to describe a property of the value arrangement in an invertible substitution or, equivalently, a conventional block cipher. If we have some input value, and then change one bit in that value, we expect about half the output bits to change; this is the result of diffusion; when partial diffusion is repeated we develop avalanche; and the ultimate result is strict avalanche. *Completeness* tightens this concept and requires that changing a particular input bit produce a change in a particular output bit, at some point in the transformation (that is, for at least one input value). Completeness requires that this relationship occur at least once for *every* combination of input bit and output bit. It is tempting to generalize the definition to apply to multi-bit element values, where this makes more sense.

Completeness does *not* require that an input bit change an output bit for *every* input value (which would not make sense anyway, since *every* output bit must be changed at *some* point, and if they all had to change at *every* point, we would have *all* the output bits changing, instead of the desired half). The inverse of a complete function is not necessarily also complete.

As originally defined in Kam and Davida:

> "For every possible key value, every output bit c_i of the SP network depends upon all input bits $p_1,...,p_n$ and not just a proper subset of the input bits." [p.748] -- Kam, J. and G. Davida. 1979. Structured Design of Substitution-Permutation Encryption Networks. *IEEE Transactions on Computers.* C-28(10):747-753.

Complex Number

An ordered pair of real numbers (x,y) treated as the vector from the origin at (0,0) to (x,y), and represented either as real and imaginary rectangular coordinates (x,y) or as the magnitude and angle (mag,ang) of the vector. Denoted **C**. The rectangular representation is also called "Cartesian"; the magnitude and angle form is called "polar."

To build an appropriate algebra and make complex numbers a field, the rectangular representation is written as (x+iy) [or (x+jy)], where i [or j] has the value SQRT(-1). The symbol i is called "imaginary," but we might just consider it a way for the algebra to relate the values in the ordered pair. Clearly, i * i = -1.

With appropriate rules like:

```
addition:        (a+bi) + (c+di) = (a+c) + (b+d)i
multiplication:  (a+bi) * (c+di) = (ac-bd) + (bc+ad)i
           c+di   ac+bd    ad-bc
division:  ---- = ----- + (-----)i
           a+bi   aa+bb    aa+bb
```

we get complex algebra, and can perform most operations and even evaluate trignometric and other complex functions like we do with reals.

In cryptography, perhaps the most common use of complex numbers occurs in the FFT, which typically transforms values in rectangular form. Sometimes we want to know the magnitude or length of the implied vector, which we can get by converting the rectangular (x,y) representation into the (mag,ang) representation:

```
magnitude:   mag(z) = SQRT( x*x + y*y )
angle:       ang(z) = arctan( y / x)

Note: Computer arctan(x) functions are generally unable to
  place the angle in the proper quadrant, but arctan2(x,y)
  routines -- with two input parameters -- may be available
  to do so.
```

Component

A part of a larger construction; a building-block in an overall design or system. Modern digital design is based on the use of a few general classes of pre-defined, fully-specified parts. Since even digital logic can use or even require analog values internally, by enclosing these values the logic component can hide complexity and present the appearance of a fully digital device.

The most successful components are extremely general and can be used in many different ways. Even as a brick is independent of the infinite variety of brick buildings, a flip-flop is independent of the infinite variety of logic machines which use flip-flops.

The source of the ability to design and build a wide variety of different electronic logic machines is the ability to interconnect and use a few very basic but very general parts.

Electronic components include

- passive components like resistors, capacitors, and inductors;
- active components like transistors and even relays, and
- whole varieties of active electronic logic devices, including flip-flops, shift registers, and state storage, or memory.

The use of individual components to produce a working complex system in production requires: first, a comprehensive specification for each part; and next, full *testing* to guarantee that each part actually meets the specification (see: quality management).

Digital logic is normally specified to operate correctly over a range of supply voltage, temperature, loading, clock rates, and other appropriate parameters. Specified limits (minimum's or maximum's) guarantee that a working part will operate correctly even with the worst case of all parameters *simultaneously.* This process allows large, complex systems to operate properly in practice, *provided* the designer makes sure that none of the parameters can exceed their correct range.

Cryptographic system components include:

- Nonlinear transformations, such as S-boxes / substitution tables,
- key hashing, such as CRC,
- random number generators, such as additive RNG's,
- sequence isolators such as jitterizers,
- combiners, such as Dynamic Substitution, Latin squares, and exclusive-OR,
- mixers, such as Balanced Block Mixers, or orthogonal Latin squares.

Composite

An integer which is not prime, and is, therefore, a product of at least two primes. See factor.

Compression

See data comprression.

Compromise

To expose a secret.

Computer

Originally the job title for a person who performed a laborious sequence of arithmetic computations. Now a machine for performing such calculations.

A logic machine with:

1. Some limited set of fundamental computations. Typical operations include simple arithmetic and Boolean logic. Each operation is selected by a particular operation code value or "opcode." This is a hardware interpretation of the opcode.
2. The ability to follow a list of instructions or commands, performing each in sequence. Thus capable of simulating a wide variety of far more complex "instructions."
3. The ability to execute or perform at least some instructions conditionally, based on parameter values or intermediate results.
4. The ability to store values into a numbered "address space" which is far larger than the instruction set, and later to recover those values when desired.

The general model of mechanical computation is the finite state machine, which is absolutely deterministic and, thus, predictable.

Also see: source code, object code, software, system design, Software Engineering and Structured Programming.

COMSEC

The military abbreviation for "communications security." Communications security includes:

- Cryptosecurity (a cryptosystem or cipher system using cryptography)
- Emission security (incidental electromagnetic radiation and other TEMPEST concerns)
- Physical security (document, key and equipment access control)
- Transmission security (use of exotic transmission methods to minimize detection, traffic analysis and faking of the transmission itself; e.g., spread spectrum,)

Conclusion

In the study of logic, the last statement in an argument, deduced from the preceeding premises.

Condition

A requirement for a statement to be true. A limitation on statement universal applicability. Also see: assumption and premise.

Conductor

A material in which electron flow occurs easily. Typically a metal; usually copper, sometimes silver, brass or aluminum. A wire. As opposed to an insulator and a semiconductor.

As a rule of thumb, a cubic centimeter (cc) of a solid has about 10^{24} or 1E24 atoms. In a metal, usually each atom contributes one or two electrons, so a metal has about 10^{24} (1E24) free electrons per cc. This massive number of free electrons has a tiny resistance to current flow of something like 10^{-6} ohms across a cubic centimeter of copper, or about one microhm per cm^3. Apparently the International Annealed Copper Standard (IACS) says that annealed copper with a cross sectional area of a square centimeter should have a resistance of about 1.7241 microhms/cm (at 20 degrees Celsius), which is satisfactorily close.

A cube with one millimeter sides has 1/100 the cross sectional area of a centimeter cube (and is about like AWG 17 wire), and so would have 100x the resistance per cm., but also is only 1/10 the length, for about 17 microhms per millimeter copper cube. A meter of AWG 17 wire would have 1000 millimeter-size cubes at 17 microhms each, so we would expect it to have about 17 milliohms total resistance. As a check, separate wire tables give the resistance of AWG 17 at 5.064 ohms per 1000ft (304.8m), which is 0.017 ohms (17 milliohms) per meter.

RESISTANCE RELATIVE TO COPPER (cm cube = 1.7241 microhms)

	Resist	Temp Coef	Thermal Cond	Melts (deg C)	
Silver (Ag)	0.95	0.0038	4.19	960.5	
Copper (Cu)	1.00	0.00393	3.88	1083	
Gold (Au)	1.416	0.0034	2.96	1063	
Aluminum (Al)	1.64	0.0039	2.03	660	
Bronze (Cu+Sn)	2.1	---	---	1280	
Tungsten (W)	3.25	0.0045	1.6	3370	
Zinc (Zn)	3.4	0.0037	1.12	419	
Brass (Cu+Zn)	3.9	0.002	1.2	920	
Nickel (Ni)	5.05	0.0047	0.6	1455	
Iron (Fe)	5.6	~0.005	0.67	1535	
Tin (Sn)	6.7	0.0042	0.64	231.9	
Chromium (Cr)	7.6	---	---	2170	
Steel (Fe+C)	~10	---	0.59	1480	
Lead (Pb)	12.78	0.0039	0.344	327	
Titanium (Ti)	47.8	---	0.41	1800	
Stainless (-->)	52.8	---	0.163	1410	(Fe+Cr+Ni+C)
Mercury (Hg)	55.6	0.00089	0.063	-38.87	
Nichrome (Ni+Cr)	65	0.00017	0.132	1350	
Graphite (C)	590	---	---	3800	
Carbon (C)	2900	-0.0005	---	3500	

Confidential

Information intended to be secret.

Confusion

Those parts of a cipher mechanism which change the correspondence between input values and output values. In contrast to diffusion.

Confusion Sequence

The sequence combined with data in a conventional stream cipher. Normally produced by a random number generator, it is also called a running key or keystream.

Conjecture
A statement which is hoped to be true. A proposed theorem. Also see: proposition.

Consequent
In the study of logic, the then-clause of an if-then statement. Also see: antecedent

Conspiracy
1. A secret agreement among like-minded individuals to work toward a particular hidden goal despite outward appearances.
2. Legally, an agreement to break the law.

In a conspiracy, multiple individuals can each contribute a minor action to accumulate a large effect. One obvious approach is to use gossip to give the impression that all right-thinking people are against some one or some thing. A conspiracy can be difficult to oppose, because a major effect can be achieved with minor actions that individually do not call for a major response.

Constant
In mathematics and computing, a symbol or explict value which does not or cannot change during the operation. As opposed to a variable. Also see: expression and equation.

Contextual
In the study of logic, an observed fact dependent upon other facts *not* being observed. Or a statement which is conditionally true, provided other unmentioned conditions have the appropriate state. As opposed to absolute.

Contradiction
In the study of logic, an expression which is always false and cannot be true. In contrast to tautology. Also see counterexample and paradox.

Conventional Block Cipher
That proper subset of block ciphers which emulates a huge simple substitution or a huge bijection. Also see block cipher and stream cipher models.

Conventional Cipher
A secret key cipher.

Congruence
Casually speaking, the remainder after a division of integers.

In number theory we say than integer a (exactly) *divides* integer b (denoted a | b) if and only if there is an integer k such that ak = b.

In number theory we say that integer a is *congruent* to integer b *modulo* m, denoted a = b (mod m), if and only if m | (a - b). Here m is the divisor or *modulus.*

Conventional Current Flow
In electronics, the idea that current flow occurs in the direction *opposite* to the direction of electron flow. This occurs because we assign a *negative* charge to electrons. Conventional current flow occurs from anode to cathode within a device, and from the cathode of one device to the anode of another. For example, conventional current flow starts at the cathode or (+) end of a battery, connects to the anode of a component and flows out the cathode, which connects to the anode or (-) end of the battery.

Convolution

Polynomial multiplication. A multiplication of each term against each other term, with no "carries" from term to term. Also see correlation.

Used in the analysis of signal processing to develop the response of a processing system to a complicated real-valued input signal. The input signal is first separated into some number of discrete impulses. Then the system response to an impulse -- the output level at each unit time delay after the impulse -- is determined. Finally, the expected response is computed as the sum of the contributions from each input impulse, multiplied by the magnitude of each impulse. This is an approximation to the convolution integral with an infinite number of infinitesimal delays. Although originally accomplished graphically, the process is just polynomial multiplication.

It is apparently possible to compute the convolution of two sequences by taking the FFT of each, multiplying these results term-by-term, then taking the inverse FFT. While there is an analogous relationship in the FWT, in this case the "delays" between the sequences represent mod 2 distance differences, which may or may not be useful.

Copyright

A U.S. federal (and worldwide) right by which the owner of a creative work can prevent others from copying and thus stealing that work. Copyright covers "original works of authorship" that are "fixed in a tangible form of expression," including: books, pamphlets, musical scores and plays; pictures, graphics and sculpture; movies, audio recordings and architecture. Included as literary works are computer program source code, and compilations of existing material, facts or data, which may include even simple lists of facts, when produced by creative selection. A copyright owner has the exclusive right to: reproduce the work, to derive subsequent works, to distribute copies, and to perform or display the work. Copyright is an intellectual property right; also see plagiarism.

Copyright protects a particular *expression*, but not the underlying idea, process or *function* it may perform, which is the province of patent protection. Copyright protects *form*, not *content*: Copyright can protect particular text and diagrams, but not the described concept. In general, copyright comes into existence simply by *creating* a picture or manuscript or making a selection; theoretically, no notice or registration is required. (See the Library of Congress circular "Copyright Basics": http://www.loc.gov/copyright/circs/circ1.html#cr). However, formal registration is required before a lawsuit can be filed, and registration within 3 months of publication supports recovery of statutory damages and attorney fees; otherwise, apparently only actual damages can be recovered. Similarly, no copyright notice is required, but having one like this:

> Copyright 1991 Terry Ritter. All Rights Reserved.

may avoid an "innocent infringement" defense. Protection currently lasts 70 years beyond the death of the author, or 95 years from date of publication for works for hire. Copyright is not handled by the PTO but instead by the United States Copyright Office (http://lcweb.loc.gov/copyright/) in the Library of Congress.

Corollary

An incidental result from the proof of a theorem. Also see: lemma.

Correlation

1. A statistical relationship, not necessarily linear, typically between two variables. A co-relation (Galton, 1869).
2. The probability that two sequences of symbols will, in any position, have the same symbol. (We expect two random binary sequences to have the same bit values about half the time.)
3. The general idea that symbols or sequences of symbols will have some non-random relationship in value. For example, each new bit in an LFSR sequence is perfectly correlated to some computation involving earlier bits in the sequence. (Also see independent and rule of thumb.)

One way to evaluate a common correlation of two real-valued sequences is to multiply them together term-by-term and sum all results. If we do this for all possible "delays" between the two sequences, we get a "vector" or 1-dimensional array of correlations which is a convolution. Then the maximum value represents the delay with the best simple correlation.

Correlation Coefficient

A statistic or measure of simple linear correlation between two binary sequences. Correlation coefficient values range from -1 to +1 and are related to the probability that, given a symbol from one sequence, the other sequence will have that same symbol. A value of:

- -1 implies a 0.0 probability (the second sequence is the complement of the first),
- 0 implies a 0.5 probability (no simple correlation found)
- +1 implies a 1.0 probability (the sequences are the same).

"The correlation coefficient associated with a pair of Boolean functions *f(a)* and *g(a)* is denoted by C(f,g) and is given by C(*f*,*g*) = 2 * prob(*f(a)* = *g(a)*) - 1 ." -- Daemen, J., R. Govaerts and J. Vanderwalle. 1994. Correlation Matrices. *Fast Software Encryption.* 276. Springer-Verlag.

Counterexample

An exception to a conjecture. A statement which meets the requirements or premises of a conjecture, but is known to have a different conclusion. Or a true statement which contradicts some part of the conjecture. A statement which refutes the proof of a lemma or theorem.

There are two classes: A *local* counterexample refutes a lemma, but not necessarily the main conjecture. A *global* counterexample refutes the main conjecture.

Counter Mode

That operating mode of a block cipher in which a count value is enciphered and the result used for some ciphering purpose. Typically, the counter and cipher are used as an RNG or confusion sequence generator in stream cipher systems.

Note that integer counting produces perhaps the best possible signal for investigating block cipher deficiencies in the rightmost bits. Accordingly, incrementing by some large random constant, or using some sort of LFSR or other polynomial counter which changes about half its bits on each step may be more appropriate.

Counting Number

The set: {1, 2, 3, ...}. The positive integers. The natural numbers without zero.

Covariance

In statistics, a measure of the extent to which random variable X will predict the value of random variable Y. When X and Y are independent, the covariance is 0. When X and Y are linearly related, the covariance squared is the variance of X times the variance of Y.

Coverage

The affected proportion of the whole. In fault tolerance, the probability a system failure is prevented, given a particular fault. In testing, the tested proportion of the whole specification.

CRC

Cyclic Redundancy Check: A fast error detecting hash based on mod 2 polynomial operations for speed and simplicity. Also see checksum.

CRC error-checking is widely used in practice to check the data recovered from magnetic storage. When data are written to disk, a CRC of the original data is computed and stored along with the data itself. When data are recovered from disk, a new CRC is computed from the recovered data and that result compared to the recovered

CRC. If the CRC's do not match, we have a "CRC error."

Computer disk-read operations always have some chance of a "soft error" which does not re-occur when the same sector is re-read, so the usual hardware response is to try again, some number of times. If that does not solve the problem, the error may be reported to the user and could indicate the start of serious disk problems.

A CRC operation is essentially a remainder over the huge numeric value which is the data; the mod 2 polynomials make this "division" both faster and simpler than one might expect. Related techniques like integer or floating point division can have similar power, but are unlikely to be as simple. In general, "division" techniques only miss errors which are some product of the divisor, and so n-bit codes miss only 1 out of every 2^n (that is, 2**n) possible errors, on average. Earlier techniques, such as checksum are significantly less able to detect errors in real data.

The CRC result is an excellent (but linear) hash value corresponding to the data. Compared with other hash alternatives, CRC's are simple and straightforward. They are well-understood. They have a strong and complete basis in mathematics, so there can be no surprises. CRC error-detection is mathematically tractable and provable without recourse to unproven assumptions. And CRC hashes do not need padding. None of this is true for most cryptographic hash constructions.

For error-detection, the CRC register is first initialized to some fixed value known at both ends, nowadays typically "all 1's." Then each data element is processed, each of which changes the CRC value. When all of the data have been processed, the CRC result is sent or stored at the end of the data. Frequently the CRC result first will be complemented, so that a CRC of the data and the complemented result will produce a fixed "magic number." This allows efficient hardware error-checking, even when the hardware does not know how large the data block will be in advance. (Typically, the end of transmission, after the CRC, is indicated by a hardware "done" signal.)

Nowadays, CRC's are often computed in software which is generally more efficient with larger data quantities. Thus we see 8-bit, 16-bit or 32-bit data elements being processed. However, CRC's can be computed on individual data **bits**, and on records of arbitrary bit length, including zero bits, one bit, or any uneven or dynamic number of bits. As a consequence, no padding is ever needed for CRC hashing.

Here is a code snippet for a single-bit left-shift CRC operation:

```
if (msb(crc) == databit)
   crc = crc << 1;
else
   crc = (crc << 1) ^ poly;
```

This fragment needs to execute 8 times to compute the CRC for a full data byte. However, a better way to process a byte in software is to pre-compute a 256-element table representing every possible CRC change corresponding to a single byte. The table value is selected by a data byte XORed with the current top byte of the CRC register (in a left-shift implementation).

In the late 60's and early 70's, the first CRC's were initialized as "all-0's." Then it was noticed that extra or missing 0-bits at the start of the data would not be detected, so it became virtually universal to init the CRC as "all-1's." In this case, extra or missing zeros at the start *are* detected, and extra or missing ones at the start *are* detected as well.

It is possible for multiple errors to occur and the CRC result to end up the same as if there were no error. But unless the errors are introduced *intentionally*, this is very unlikely. Various common errors *are* detected *absolutely*, such as:

- Any single bit added, anywhere.

- Any single bit deleted.
- Any single bit changed.
- All "burst errors" (contiguous bits in error) of length smaller than the polynomial.

If we have enough information, it is relatively easy to compute error patterns which will take a CRC value to any desired CRC value. Because of this, data can be changed in ways which will produce the original CRC result. Consequently, no CRC has any appreciable cryptographic strength, but some applications in cryptography *need* no strength:

- One example is key processing, where the uncertainty in a User Key phrase of arbitrary size is collected into a hash result of fixed size. In general, the hash result would be just as good for the opponent as the original key phrase, so no strength shield could possibly improve the situation.
- Another example is the hash accumulation of the uncertainty in slightly uncertain physically random or really random events. When true randomness is accumulated, it is already as unknowable as any strength shield could make it.

On the other hand, a CRC, like most computer hashing operations, is normally used so that we *do not* have "enough information." When substantially more information is hashed than the CRC can represent, any particular CRC result will be produced by a vast number of different input strings. In this way, even a linear CRC can be considered an irreversible "one way" or "information reducing" transformation. Of course, when a string shorter than the CRC polynomial is hashed, it should not be too difficult to find the one string that could produce any particular CRC result.

The CRC polynomial need not be particularly special. Unlike the generator polynomials used in LFSR's, a CRC poly need not be primitive nor even irreducible. Indeed, the early 16-bit CRC polys were composite with a factor of "11" which is equivalent to the information produced by a parity bit. (Since parity was the main method of error-detection at the time, the "11" factor supported the argument that CRC was better.) However, modern CRC polys generally *are* primitive, which allows the error detection guarantees to apply over larger amounts of data. It also allows the CRC operation to function as an RNG. But the option exists to use secret random polynomials to detect errors without being as predictable as a standard CRC. Polynomial division does not require mathematical structure (such as an irreducible or primitive), beyond the basic mod 2 operations.

Different CRC implementations can shift left or right, take data lsb or msb first, and be initialized as zeros or ones, each option naturally producing different results. Various CRC standards specify different options. Obviously, both ends must do things the same way, but it is not necessary to conform to a standard to have quality error-detection for a private or new design. Variations in internal handling can make a CRC with one set of options produce the same result as a CRC with other options.

When the logical complement of a CRC result is appended to the data and processed msb first, the CRC across that data and the result produces a "magic" value which is a constant for a particular poly and set of options. In general, the sequence reverse of a good poly is also a good poly, and there is some advantage to having CRC polys which are about half 1's. In some notations we omit the msb which is always 1 (as is the lsb). For notational convenience, we can write $x^3 + x^2 + x + 1$ as: 3,2,1,0. These are the positions of 1-bits in the poly. Common CRC polys include:

Name	Hex	Set Bits
CRC16	**8005**	**16,15,2,0**
CCITT	**1021**	**16,12,5,0**
CRC24a	800063	
CRC24b	800055	
CRC24c	861863	
CRC32	**04c11db7**	**32,26,23,22,16, 12,11,10,8,7,5,4,2,1,0**
SWISS-PROT		64,4,3,1,0

```
    SWISS-PROT Impr     64,63,61,59,58,56,55,52,49,48,
    (D. Jones)          47,46,44,41,37,36,34,32,
                        31,28,26,23,22,19,16,
                        13,12,10,9,6,4,3,0
```

Also see: reverse CRC, and

- my article "The Great CRC Mystery," locally, or @: http://www.ciphersbyritter.com/ARTS/CRCMYST.HTM
- the discussion "Randomness and the CRC" locally, or @: http://www.ciphersbyritter.com/NEWS2/CRCRAND.HTM
- and the discussion "Hashing and CRC" locally, or @: http://www.ciphersbyritter.com/NEWS5/CRCHASH.HTM

Crib

In cryptanalysis, a small amount of known plaintext which is assumed or guessed to help attack a cipher.

CRNG

cryptographic random number generator. A cryptographic RNG.

CRT

1. Cathode Ray Tube. The large vacuum tube with phosphor display which is still vastly the most common television and computer monitor display.
2. In mathematics, Chinese Remainder Theorem.

Cryptanalysis

That aspect of cryptology which concerns the strength of a cryptographic system, and the penetration or breaking of a cryptographic system. Also known as codebreaking, cryptanalysis is first of all analysis, the picking apart of a cipher and/or cipher system to form a deep understanding of the operations and their consequences. Also see: cryptography war and traffic analysis. Also see: risk analysis, tree analysis, fault tree analysis and threat model. Also see: hypothesis.

In normal cryptanalysis we start out knowing plaintext, ciphertext, and cipher construction. The only thing left unknown is the key. A practical attack must recover the key. (Or perhaps we just know the ciphertext and the cipher, in which case a real attack would recover plaintext.) Simply finding a distinguisher (showing that the cipher differs from the chosen model) is not, in itself, an attack or break.

What Makes a Cipher "Strong"?

Because no theory guarantees strength for any conventional cipher (see, for example, the one time pad and proof), ciphers traditionally have been considered "strong" when they have been used for a long time with "nobody" knowing how to break them easily.

Expecting cipher strength because a cipher is not known to have been broken is the logic fallacy of ad ignorantium: a belief which is claimed to be true because it has not been proven false.

Cryptanalysis seeks to extend this admittedly-flawed process by applying known attack strategies to new ciphers (see heuristic), and by actively seeking new attacks. Unfortunately, real attacks are directed at particular ciphers, and there is no end to different ciphers. Even a successful break is just one more trick from a virtually infinite collection of unknown knowledge.

In cryptanalysis it is normal to assume that at least known-plaintext is available; often, defined-plaintext is assumed. The result is typically some value for the amount of work which will achieve a break (even if that value is impractical); this is the strength of the cipher under a given attack. Different attacks on the same cipher may thus imply different amounts of strength. While cryptanalysis *can* demonstrate "weakness" for a given level of effort, cryptanalysis *cannot* prove that there is no simpler attack (see, for example, attack tree and threat model):

Lack of proof of weakness is not proof of strength.

Indeed, when ciphers are used for real, the opponents can hardly be expected to advertise a successful break, but will instead work hard to reassure users that their ciphers are still secure. The fact that *apparently* "nobody" knows how to break a cipher is somewhat less reassuring from this viewpoint. (Also see the discussion: "The Value of Cryptanalysis," locally, or @: http://www.ciphersbyritter.com/NEWS3/MEMO.HTM). For this reason, using a wide variety of different ciphers can make good sense: That reduces the value of the information protected by any particular cipher, which thus reduces the rewards from even a successful attack. Having numerous ciphers also requires the opponents to field far greater resources to identify, analyze, and automate breaking (when possible) of each different cipher. Also see: Shannon's Algebra of Secrecy Systems.

How Are Ciphers Cracked?

In general, a cipher can be seen as a key-selected set of transformations from plaintext to ciphertext (and vise versa for deciphering). A conventional block cipher takes a block of data or a block value and transforms that into some probably different value. The result is to emulate a huge, keyed substitution table, so, for enciphering, we can write this *very simple* equation:

```
    E[K][PT] = CT,  or  E[K,PT] = CT
```

where PT = plaintext block value, K = Key, and CT = ciphertext block value. The brackets "[ ]" mean the operation of indexing: the selection of a particular position in an array and returning that element value. Here, E[K] represents a particular, huge, emulated encryption "table," while E[K][PT] selects a single entry (a block value) from that table. So we can recover the plaintext with:

```
    D[K][CT] = PT,  or  D[K,PT] = CT
```

where D[K] represents an inverse or decryption table. But an attacker does not know and thus must somehow develop the decryption table.

We assume that an opponent has collected quite a lot of information, including lots of plaintext and the associated ciphertext (a condition we call known plaintext). The opponent also has a copy of the cipher and can easily compute every enciphering or deciphering transformation. What the opponent does *not* have, and what he is presumably looking for, is the key. The key would expose the myriad of other ciphertext block values for which the opponent has no associated plaintext.

We might imagine the opponent attacking a cipher with a deciphering machine having a huge "channel-selector" dial to select a key value. As one turns the key-selector, each different key produces a different deciphering result on the display. So all the opponent really has to do is to turn the key dial until the plaintext message appears. Given this extraordinarily simple attack (known as brute force), how can any cipher be considered secure?

In a real cipher, we make the key dial very, very, *very* big! The keyspace of a real cipher is much too big, in fact, for anyone to try each key in a reasonable amount of time, even with massively-parallel custom hardware. That leaves the opponent with a problem: brute force does not work.

Nevertheless, the cipher equation seems *exceedingly simple*. There is one particular huge emulated table as selected by the key, and the opponent has a sizable set of positions and values from that table. Moreover, all the known and unknown entries are created by exactly the same mechanism and key. So, if the opponent can in some way relate the known entries to the rest of the table, thus predicting unknown entries, the cipher may be broken. Or if the opponent can somehow relate known plaintexts to the key value, thus *predicting* the key, the key may be exposed. And with the key, ciphertext for which there is no corresponding plaintext can be exposed, thus breaking the cipher. Finding these relationships is where the cleverness of the individual comes in. In a real sense, a cipher is a *puzzle*, and we currently cannot guarantee that there is no particular "easy" way for a smart team to solve it.

One peculiarity of conventional block ciphers is that they cannot emulate all possible tables, but instead only a tiny, tiny fraction thereof (see block cipher). Even what we consider a huge key simply cannot select from among all possible tables because there are far too many. Now, the "tiny fraction" of tables actually emulated is still too many to traverse (this is a "large enough" keyspace), but, clearly, some special selection is happening which might be exploited. Having even one particular value at one known table position is sufficiently special that we expect that only one key would produce that particular relationship in a conventional cipher. So, in practice, just *one* known-plaintext pair generally should be sufficient to identify the correct key, if only we could find some way to do it.

More Realistic Attacks

In academic cryptanalysis we normally assume that we do not know the key, but do know the cipher and everything about it. We also assume essentially unlimited amounts of known plaintext to use in an attack to find the key. In practice things are considerably different.

In practical cryptanalysis we may not know which cipher has been used. The cipher may not ever have been published, or may have been modified from the base version in various ways. Even a cipher we basically know may have been used in a way which will disguise it from us, for example:

- The external key may have been custom processed or hashed and may not be the key the internal cipher uses. The key value could have been casually bit-reversed or byte-reversed or oddly padded or so on. Or the key itself could have been intentionally enciphered.
- The message text or data could be changed in just as many different ways.

So even if we have the right key, we can only show that by checking every cipher we know in every way it could have been used.

Unknown Ciphers

Selecting among different ciphers is part of Shannon's 1949 Algebra of Secrecy Systems. In a modern computer implementation, we could select ciphers dynamically. The number of selectable transformations increases exponentially when several ciphers are used in sequence (multiple encryption). Considering Shannon's academic work in this area, the use of well-known standardized designs is, ironically and rather sadly, current cryptographic orthodoxy (see risk analysis).

The general mathematical model of ciphering is that of a keyed transformation (a mapping or function). Numerically, we can make the general model work for a system of multiple ciphers by allowing some "key" bits to select the cipher, with the rest of the key bits going to key that cipher. But in the adapted model, different parts of the key will have vastly different difficulties for the opponent. Finding the correct key within a cipher may be hard, yet could be much, much easier than finding the exact cipher actually being used. Differences in the difficulty of finding different key bits are simply glossed over in the adapted general model.

Somehow obtaining and breaking every cipher which possibly could have been used is a vastly larger problem than the relatively small increase in keyspace indicated by the number of possible ciphers. For example, if we think we have found the key and want to check it, on a known cipher that has essentially no cost and may take a microsecond. But if we want to check the key on an unknown cipher, we first have to obtain that cipher. That may require the massive ongoing cost of maintaining an intelligence field service to obtain copies of secret ciphers. Once the needed cipher is obtained, finding a practical break may take experts weeks or months, if a break is even found. Taken together, this is a vast increase in difficulty for the opponent per cipher choice compared to the difficulty per key choice within a single cipher.

Just as it may be impossible to try every 128-bit key at even a nanosecond apiece, it also may be impossible to keep up with a far smaller but continuing flow of new secret ciphers which take hundreds of billions of times longer to handle. This advantage seems to be exploited by NSA in keeping cipher designs secret (also see security through obscurity). Given the stark contrast of yet another real example which contradicts the current cryptographic wisdom, crypto academics continue to insist that standardizing and exposing the cipher design makes sense. Surely, exposing a cipher does support gratuitous analysis and help to expose some cipher weakness, but does not, in the end, give us a proven strong cipher. In the end, exposing the cipher may turn out to benefit opponents far more than users.

In practice, an individual attacker mainly must hope that the cipher to be broken is flawed. An attacker can collect ciphertext statistics and hope for some irregularity, some imbalance or statistical bias that will identify the cipher class, or maybe even a well-known design. An attacker can make plaintext assumptions and see if some key will produce those words. But enciphering guessed plaintext seems an unlikely path to success when every possible cipher, and every possible modification of that cipher, is the potential encryption source. All this is a very difficult problem, and far different than the normal academic analysis.

Other Weaknesses

Many academic attacks are essentially theoretical, involving huge amounts of data and computation. But even when a direct technical attack is *practical,* that may be the most difficult, expensive and time-consuming way to obtain the desired information. Other methods include making a paper copy, stealing a copy, bribery, coercion, and electromagnetic monitoring. No cipher can keep secret something which has been otherwise revealed. Information security thus involves far more than just cryptography, and a cryptographic system is more than just a cipher (see: cipher system). Even finding that information has been revealed does not mean that a cipher has been broken, although good security virtually requires that assumption. (Of course, when we can use only one cipher, we cannot change ciphers anyway.)

Unfortunately, we have no way to know how strong a cipher appears to our opponents. Even though the entire reason for using cryptography is a belief that our cipher has sufficient strength, science provides no basis for such belief. At most, cryptanalysis can give us only an *upper limit* to the strength of a cipher, which is not particularly helpful, and can only do that when a cipher actually can be broken. But when a cipher is *not* broken, cryptanalysis has told us nothing about the strength of a cipher, and unbroken ciphers are the only ones we use.

Cryptanalytic Contributions and Limits

The ultimate goal of cryptanalysis is *not* to break every possible cipher (that would be the end of an industry and also the end of new PhD's in the field). Instead, the obvious goal is *understanding* why some ciphers are weak, and why other ciphers seem strong. It is not much of a leap from that to expect cryptanalysts to work with, or at least interact with, cipher designers, with a *common goal* of producing better ciphers.

Unfortunately, cryptanalysis is ultimately limited by what can be done: there are no ciphering techniques which guarantee strength, and there is no test which tells us how weak an arbitrary cipher really is. Accordingly, exposing a particular weakness in a particular cipher may be about as much as cryptanalysis can offer, even if that means a deafening silence about similar designs, ciphers which have been repaired, or significant cipher designs which remain both unbroken and undiscussed.

Cryptanalyst

Someone who attacks ciphers with cryptanalysis. A "codebreaker." Often called the opponent by cryptographers, in recognition of the (serious) game of thrust and parry between these parties.

Crypto Controversies

In cryptography, some supposedly technical disputes just seem to go on and on. Some topics have led to many hundreds of postings on Usenet sci.crypt in various conversations across multiple years. Clearly, those are controversial topics, whether academics think so or not. But it is also controversial to point out alternatives which conflict with current cryptographic wisdom or techniques. Dissent and disagreement are how controversies start.

Apparent agreement among academics does not imply a lack of academic controversy, since many will side with the conventional wisdom, while others step back to consider the arguments. Since reality is not subject to majority rule, even universal academic agreement would not constitute a scientific argument, which instead requires facts and exposed logical reasoning. Controversy may even imply that academic cryptographers are unaware of the issue, or have not really considered it in a deep way. For if clear, understandable and believable explanations already existed, there would be little room for debate. Controversy arises when the given explanations are false, or obscure, or unsatisfactory.

Scientific controversy is less about conflict than exposing Truth. That happens by doing research, then taking a stand and supporting it with facts and scientific argument. Many of these issues should have indisputable answers or expose previously ignored consequences. Wishy-washy statements like "some people think this, some think that," not only fail to inform, but also fail to frame a discussion to expose the real answer.

Science Means Models

One aspect of science is the creation of quantitative models which describe or predict reality. Since poor models lead to errors in reasoning, a science reacts to poor predictions by improving the models. In contrast, cryptography reacts by making excuses about why the model really is right after all, or does not apply, or does not matter. Examples include:

- A common model for a conventional block cipher is "a family of permutations." But that family is huge, and in practice almost none of that family can be selected. A better model would be "a tiny subset of a family of permutations," which would lead immediately to questions about subset size and structure. Absent a realistic model, such questions rarely arise, and are even more rarely addressed.
- The one time pad (OTP) is often said to be "proven secure." But NSA has in fact broken OTP's in practice, as we know from their description of VENONA. In exactly what way can cryptography claim that "proven secure" has any practical meaning, when using a "proven secure" cipher can result in death by execution from cipher failure? If "proven secure" is intended to be only theoretical, where is the practical analysis we need?
- Cryptanalysis is sometimes said to be how we know our ciphers are strong, but that is false. The best cryptanalysis can do is find a problem, but not finding a problem does not make a cipher "strong," only "not known to be weak." That is not mere semantics, but instead sets absolute limits on cryptanalysis as a source of knowledge about strength. It also implies that cryptography must do something different just to get the knowledge of strength that most people think we already have. By not exposing this limitation, cryptography encourages students, buyers and users to form the conceptual model that mathematics can deliver guaranteed strength in practice when in reality that is almost never possible. And it is similarly impossible to know the probability of failure.

In my view, cryptography has presided over a fundamental breakdown in logic, perhaps created by awe of supposedly superior mathematical theory. Upon detailed examination, however, theoretical math often turns out to be inapplicable to the case at hand. Practical results which conflict with theory are ignored or dismissed, even though confronting reality is how science improves models. Demanding belief in conventional cryptographic wisdom requires people to think in ways which accept logical falsehood as truth, and then they apply that lesson. Reasoning errors have become widespread, accepted, and prototypes for future thought. Disputes in cryptography are commonly argued with logic fallacies, and may be "won" with arguments that have no force at all. Since experimental results are rare in cryptography, we cannot afford to lose reason, because that is almost all we have.

One Cipher is Not Enough

A major logical flaw in conventional cryptography is the belief that one good cipher is enough.

But since cipher strength occurs only in the context of our opponents, *how could we ever know* that we have a "good" cipher, or how "good" it is?

In particular:

1. There is no way to *measure* strength for an arbitrary practical cipher.
2. Theoretical strength "proofs" almost never apply in practice.

So if we cannot *measure* "good," and cannot *prove* "good," then exactly *how do we know* our ciphers are "good?" The answer, of course, is that we do not and can not know any such thing. In fact, *nobody* on our side can know that our ciphers are "good," no matter how well educated, experienced or smart they may be, because that is determined by our opponents in secret. Anyone who feels otherwise should try to put together what they see as a scientific argument to prove their point.

When conventional cryptography accepts a U.S. Government standard cipher as "good" enough, there is no real need:

- to research and develop fundamentally new cipher designs;
- to innovate ways to measure strength;
- to certify multiple ciphers;
- to allow users to choose their cipher;
- to change ciphers from time to time; and
- there is *certainly* no need for ever-so-costly multiple encryption.

Conventional cryptography encourages a belief in known cipher strength, thus ignoring both logic and the lessons of the past. That places us all at risk of cipher failure, which probably would give no indication to the user and so could be happening right now. That we have no indication of any such thing is not particularly comforting, since that is exactly what our opponents would want to portray, even as they expose our information.

When an ordinary person makes a claim, they can be honestly wrong. But when a trained expert in the field makes a claim that we know cannot be supported, and continues to make such claims, we are pretty well forced into seeing that as either professional incompetence or deliberate deceit. Encouraging people to use only one cipher by claiming they need nothing else is exactly what one would expect from an opponent who knows how to break the cipher. Maybe that is just coincidence.

A List of Crypto Controversies

Cryptographic controversies include:

- **AES:** AES is the new conventional block cipher standard. However, like most modern block ciphers, it uses only a breathtakingly small portion of the possible keyspace. And while that may not be a known weakness, it is at least disturbing.
- **BB&S:** Blum, Blum and Shub is the way we refer to a well known article which describes a very slow random number generator which is supposedly "provably secure." Current crypto texts generally present a simplified version, but deceptively use the same BB&S name. That simplified construction has a very rare weakness related to choosing the initial value, a weakness that can be fixed at modest cost. As a consequence, a designer who wants assurance that every known fixable weakness has been eliminated will use the original construction. Confrontation arises when someone insists that no competent designer could have such a goal.
- **Bijective Compression:** When random blocks or strings decompress into apparently grammatical text, it may be difficult to automatically distinguish the correct decryption from incorrect decryptions. The result is a potential increase in practical strength.
- **Block Cipher Definitions:** Ordinarily, the person wishing to display an insight gets to set up the definitions, but of course communication requires both the writer and reader to have the same definition. Various block cipher definitions exist.
- **CBC First Block Problem:** CBC requires an IV, typically sent with ciphertext. That IV is exclusive-ORed with the plaintext of the first block. So if opponents know the first block plaintext, and can intercept and change the IV, and the IV is sent as plaintext, the opponents can change the deciphered first block to any desired value. This is a form of man-in-the-middle attack. The usual advice is to use a MAC to authenticate the full deciphered message, and that works. But the fundamental problem is confidentiality, not authentication. A MAC is unnecessary if the IV is enciphered.
- **Cryptanalysis:** Cryptanalysis is sometimes thought of as the way we know the strength of a cipher, but that is wrong. At its best, cryptanalysis can find out that a cipher is weak, and then we do not use that cipher. But cryptanalysis which finds no break does not mean that the cipher is strong, and we will use it anyway.
- **Data Compression:** Data compression can increase the unicity distance while simultaneously decreasing the size of plaintext which must exceed that distance to be attacked.
- **Distinguisher:** A distinguisher typically is some sort of statistical test that will show an imperfect distribution in, for example, a conventional block cipher. But by itself, that does not show weakness, and is no form of a break at all.
- **Dynamic Transposition:** A class of unconventional block cipher with a particularly clear strength argument.
- **Entropy:** Shannon's entropy measures coding efficiency. It produces the same result whether a sequence is known in advance or not. It produces the same result whether a sequence is predictable or not. Entropy does not measure unpredictability.
- **Huge Block Cipher Advantages:** If we have a technology which allows the efficient construction of huge blocks, such blocks can have significant advantages.
- **Kerckhoffs' Requirements:** Everybody thinks they already know what Kerckhoffs wrote.
- **Known Plaintext:** Modern ciphers are sometimes thought to be invulnerable to known plaintext. But if we model a cipher as a mathematical function taking plaintext to ciphertext, known plaintext is how an attacker examines the function that the cryptographer wishes to hide.
- **Multiple Encryption:** The idea is to add redundancy and so avoid the single point of failure represented by a single cipher.
- **Old Wives' Tales:** Various delusions are widely accepted in the strange field of cryptography.
- **One Time Pad:** The one time pad is often held up as the only example of a proven unbreakable cipher. Unfortunately, only the *theoretical* version is proven unbreakable, and that version only protects theoretical data. In *practice* there is no proven secure one time pad, because the required assumptions cannot be provably achieved in practice. Unfortunately, the crypto texts cause every crypto newbie to come to a different and wrong conclusion. That is first a failure of the field to properly model reality, and next a willingness to deceive those with the least crypto training and experience.
- **Proof:** It is currently fashionable for cryptographic constructions to be claimed (and acclaimed) "provably secure." But for a proof to apply, every assumption it uses must be both known, and known to

be true. For a proof to be effective for a user, that user must be able to first *know* every required assumption, and next *show* that each has been achieved in practice. Since few if any cryptosystems actually allow that, for users, "provably secure" generally is just another crypto deception, especially for those with the least crypto training and experience.

- **Randomness Testing:** Randomness cannot be certified by test, just indicated. If absolute randomness is required, no test can tell us when we have it.
- **Really Random:** Predictable statistical randomness is easily created, but unpredictable real randomness cannot be generated by a deterministic machine. Although various forms of "entropy" are supposedly available in a computer system, upon closer examination we may find that little unpredictability remains.
- **Risk:** Ciphers may be the only product in modern manufacture which cannot be tested to their design goal. If the goal of a cipher is to keep secrecy, that can only be judged by opponents who work against us in secret and do not announce their results. Since we cannot know when our ciphers fail, we cannot begin to know the risk of cipher failure. Designers cannot know the outcome of their attempts to meet ciphering goals. That means there can be no expertise in cipher strength, by anyone, no matter how well educated or experienced. Even opponents only know their part of the truth.
- **Scalability:** There is, and probably can be, no test to measure the strength of an arbitrary cipher. All ciphers are completely outside manufacturing control with respect to strength. An alternative is to develop designs which can be implemented at tiny size, and then exhaustively tested. Tiny ciphers will not be strong, but they can be rigorously tested in ways that large ciphers cannot. Scalability is an enabling technology which provides insight to strength, something sorely needed but not supported by conventional cryptography.
- **Snake Oil:** Since there are no tests of cipher strength, certain rules of thumb have been used to indicate weakness. Those rules have problems.
- **Software Patent:** There are no software patents; there are just patents which apply to software. There are no pure algorithm patents, but patents always have described machines which implement algorithms. In particular, process claims apparently have been granted almost from the beginning of U.S. patents. (But see a patent lawyer in any real situation.)
- **Strength:** Crypto people talk about strength precisely the way that virgins talk about sex: with great enthusiasm and little understanding. There is and can be no expertise on cipher strength.
- **Term of Art:** Terms used by various professions often mean something different than the exact same phrase used on its own. That can lead to endless difficulty in understanding and discussion, especially for those with the least training and experience.
- **Threat Model:** Trying to estimate cipher strength by predicting attack possibilities is usually hopeless.
- **Trust:** The value of trust inherently depends upon having a relationship with consequences should trust fail.
- **Unpredictability:** One aspect of science is the construction of numerical models which "correctly predict" measurable outcomes. Similarly, cryptanalysis seeks to build models which "correctly predict" the unknown cipher key. The inability to construct a key-predicting scientific model is one way to see cipher strength.

In my view, cryptography often does not understand or attempt to address controversial issues in a scientific way. In areas where cryptography cannot distinguish between truth and falsehood, it cannot advance.

If anyone has any other suggestions for this list, please let me know.

Cryptographer

Someone who creates ciphers using cryptography.

Cryptographic Hash

A complex hash function in which deliberately producing any particular result is thought to be very difficult. (Also see one way hash and MAC.)

Cryptographic Mechanism

A process for enciphering and/or deciphering, or an implementation (for example, hardware, computer software, hybrid, or the like) for performing that process. See also cryptography and mechanism.

Cryptographic Random Number Generator

(CRNG). A random number generator (RNG) suitable for cryptographic use.

Uses might include:

- As the confusion sequence generator for a stream cipher.
- Expanding key material into a much-larger state for use by a much-larger CRNG.
- As part of the process for keying S-boxes, Latin squares, orthogonal Latin squares, polynomials, primes or other key-selected or key-constructed objects.
- As an internal part of a cipher (such as providing the shuffling sequence for Dynamic Transposition).
- Generating nulls for use in padding and message-length hiding, a desirable function which is not part of ciphers *per se* (see cipher system).
- Generating nonce values for use in protocols.

Requirements for such an RNG will vary depending upon use, but might include:

- Large internal state.
- Sufficient cycle length to "never" repeat in operation.
- Ease of keying.
- Few, if any, weak keys.
- To be externally unpredictable.

Cryptography

Greek for "hidden writing." The art and science of transforming (encrypting) information (plaintext) into an intermediate form (ciphertext) which secures information in storage or transit. Normally, security occurs as a result of having a vast number of different transformations, as selected by some sort of key. Then, if an opponent acquires some ciphertext, a vast number of different plaintext messages presumably could have produced that exact same ciphertext, one for each of the possible keys.

We normally assume that the opponents have a substantial amount of known plaintext to use in their work (see cryptanalysis). So the situation for the opponents involves taking what is known and trying to extrapolate or predict what is not known. That is similar to building a scientific model intended to predict larger reality on the basis of many fewer experiments. Since the whole idea is to make prediction difficult for the opponent, unpredictability can be called the essence of cryptography.

Cryptography is a part of cryptology, and is further divided into secret codes versus ciphers. As opposed to steganography, which seeks to hide the existence of a message, cryptography seeks to render a message unintelligible *even when the message is completely exposed*.

Cryptography includes at least:

- key generation,
- secrecy (*confidentiality*, or *privacy*, or *information security*, or *encryption* operations), and
- message authentication (integrity).

Cryptography may also include:

- nonrepudiation (the inability to deny sending a message),
- access control (*source* or *user* authentication), and
- availability (keeping security services available for use).

In practice, cryptography should be seen as a system or game which includes both *users* and *opponents*: True scientific measures of strength do not exist when a cipher has not been broken, so users can only *hope* for their cipher systems to protect their messages. But opponents may benefit greatly if users can be convinced to adopt ciphers which opponents can break. Opponents are thus strongly motivated to get users to believe in the strength of a few weak ciphers. Because of this, deception, misdirection, propaganda and conspiracy are inherent in the practice of cryptography. (Also see trust and risk analysis.)

And then, of course, we have the natural response to these negative possibilities, including individual *paranoia* and *cynicism*. We see the consequences of not being able to test cipher security in the *arrogance* and *aggression* of some newbies. Even healthy users can become *frustrated* and *fatalistic* when they understand cryptographic reality. Cryptography contains a full sea of unhealthy psychological states.

Great Expectations in Cryptography

For some reason (such as the lack of direct academic statements on the issue), some networking people who use and depend upon cryptography every day seem to have a slightly skewed idea about what cryptography can do. While they seem willing to believe that ciphers might be broken, they assume such a thing could only happen at some great effort. Apparently they believe the situation has been somehow assured by academic testing. But that belief is false.

Ciphers are like puzzles, and while some ways to solve the puzzle may be hard, other ways may be easy. Moreover, once an easy way is found, that can be put into a program and copied to every "script kiddie" around. The hope that every attacker would have to invest major effort to find their own fast break is just wishful thinking. And even as their messages are being exposed, the users probably will think everything is fine, just like we think right now. Cipher failure could be happening to us right now, because there will be no indication when failure occurs.

What are the chances of cipher failure? *We cannot know*! Ciphers are in that way different from nearly every other constructed object. Normally, when we design and build something, we measure it to see that it works, and how well. But with ciphers, we *cannot* measure how well our ciphers resist the efforts of our opponents. Since we have no way to judge effectiveness, we also cannot judge risk. Thus, we simply have no way to compare whether the cipher design is more likely to be weak than the user, or the environment, or something else. As sad as this situation may seem, it is what we have.

When compared to the alternative of blissful ignorance, it should be a great advantage to know that ciphers cannot be depended upon. First, design steps could be taken to improve things (although that would seem to require a widespread new understanding of the situation that has always existed). Next, we note that ciphers can at most reveal only what they try to protect: When protected information is not disturbing, or dangerous, or complete, or perhaps not even true, exposure becomes much less of an issue.

Keying in Cryptography

Modern cryptography generally depends upon translating a message into one of an astronomical number of different intermediate representations, or ciphertexts, as selected by a key. If all possible intermediate representations have similar appearance, it may be necessary to try all possible keys (a brute force attack) to find the key which deciphers the message. By creating mechanisms with an astronomical number of keys, we can make this approach impractical.

Keying is the essence of modern cryptography. It is not possible to have a strong cipher without keys, because it is the uncertainty about the key which creates the "needle in a haystack" situation which is conventional strength. (A different approach to strength is to make every message equally possible, see: Ideal Secrecy.)

Nor is it possible to choose a key and then reasonably expect to use that same key forever. In cryptanalysis, it is normal to talk about hundreds of years of computation and vast effort spent attacking a cipher, but similar effort may be applied to obtaining the key. Even one forgetful moment is sufficient to expose a key to such effort. And when there is only one key, exposing that key also exposes all the messages that key has protected in the past, and all messages it will protect in the future. Only the selection and use of a new key terminates insecurity due to key exposure. Only the frequent use of new keys makes it possible to expose a key and not also lose all the information ever protected.

Engineering in Cryptography

Cryptography is not an engineering science: It is not possible to know when cryptography is "working," nor how close to not-working it may be:

- There is no test that will give us the strength of an arbitrary cipher.
- There is no "materials science" that will tell us how much strength to expect, given the component parts.
- Academic strength proofs almost never apply in practice.
- We do not know who is attacking, nor what their resources might be, nor how much success they have had.
- There will be no exposure of failure from which we might develop even a handwave probability of weakness or risk.
- Since we can expect to not know of failure, no amount of use without seeing failure can develop trust.

(Also see: scientific method and threat model.)

Cryptography may also be seen as a zero-sum game, where a cryptographer competes against a cryptanalyst. We might call this the cryptography war.

Cryptography War

Cryptography may be seen as a dynamic *battle* between cryptographer and cryptanalyst or opponent. The cryptographer tries to produce a cipher which can retain secrecy. Then, when it becomes worthwhile, one or more cryptanalysts may try to penetrate that secrecy by attacking the cipher. Fortunately for the war, even after fifty years of mathematical cryptology, not *one* practical cipher has been accepted as proven secure in practice. (See, for example, the one-time pad.)

Note that the successful cryptanalyst must keep good attacks secret, or the opposing cryptographer will just produce a stronger cipher. This means that the cryptographer is in the odd position of never knowing whether his or her best cipher designs are successful, or which side is winning.

Cryptographers are often scientists who are trained to ignore unsubstantiated claims. But the field of cryptography often turns the scientific method on its head, because almost never is there a complete proof of cryptographic strength in practice. In cryptography, scientists accept the failure to break a cipher as an indication of strength (that is the ad ignorantium fallacy), and then demand substantiation for claims of *weakness*. But there *will be* no substantiation when a cipher system is attacked and broken for real, while continued use will endanger all messages so "protected." Evidently, the conventional scientific approach of requiring substantiation for claims is not particularly helpful for users of cryptography.

Since the scientific approach does not provide the assurance of cryptographic strength that users want and need, alternative measures become appropriate:

- It can be a reasonable policy to not adopt a widely-used cipher, since such ciphers provide the best target for attackers and also the best reward for cryptoanalytic success.
- Another reasonable policy is to change ciphers periodically, perhaps even on a message-by-message basis. This limits the extent of use of any weak cipher.
- Yet another reasonable policy is to use multiple encryption as a matter of course (see Algebra of Secrecy Systems). Using three different ciphers on every message protects the message even if one of the ciphers has been broken in secret.

Cryptology

The field of study covering all forms of message protection and exposure. Typically thought to include:

- steganography (methods which seek to conceal the presence of a message, such as patterns in graphics and secret inks)
- cryptography (methods which translate a message into an intermediate form intended to hide information even when completely exposed)
- cryptanalysis (methods which expose information hidden by cryptography)

Sometimes also said to include:

- message interception
- traffic analysis (the use of transmission data, such as time of origin, call signs and message length, as a basis for intelligence)

Cryptosystem

A term which might be used to describe the overall function of ciphering; see: cipher system.

It is especially important to consider the effect the underlying equipment has on the design. Even apparently innocuous operating system functions, such as the multitasking "swap file," can capture supposedly secure information, and make that available for the asking. Since ordinary disk operations generally do not even attempt to overwrite data on disk, but instead simply make that storage free for use, supposedly deleted data is, again, free for the asking. A modern cryptosystem will at least try to address such issues.

Crystal

1. In Physics, atoms arranged in a repeating structure.
2. In electronics, typically a small part of a thin slice of a much larger quartz crystal, intended to regulate or filter a particular frequency. Typically a thin slab of translucent mineral, perhaps a quarter of an inch square, usually with metal terminals plated on each large side. Frequency preference occurs because of mechanical resonance -- an actual flexing of the crystalline rock itself -- which "rings" at a particular frequency.

Quartz is a piezoelectric material, so a voltage across the terminals forces the quartz wafer to bend slightly, thus storing mechanical energy in physical tension and compression of the solid quartz. The physical mass and elasticity of quartz cause the wafer to mechanically resonate at a natural frequency depending on the size and shape of the quartz blank. The crystal will thus "ring" when the electrical force is released. The ringing will create a small sine wave voltage across electrical contacts touching the crystal, a voltage which can be amplified and fed back into the crystal, to keep the ringing going as oscillation.

Crystals are typically used to make exceptionally stable electronic oscillators (such as the clock oscillators widely used in digital electronics) and the relatively narrow frequency filters often used in radio.

It is normally necessary to physically grind a crystal blank to the desired frequency. While this can be automated, the accuracy of the resulting frequency depends upon the effort spent in exact grinding, so "more accurate" is generally "more expensive."

Frequency stability over temperature depends upon slicing the original crystal at precisely the right angle. Temperature-compensated crystal oscillators (TCXO's) improve temperature stability by using other components which vary with temperature to correct for crystal changes. More stability is available in oven-controlled crystal oscillators (OCXO's), which heat the crystal and so keep it at a precise temperature despite ambient temperature changes.

Crystal Oscillator

In electronics, an oscillator in which the frequency is regulated by a quartz crystal. Typically used to produce very regular clock signals for digital electronics. In digital use, the analog sine-wave signal of the internal oscillator is digitized, and the square-wave result becomes the clock. Also see: specification.

Sometimes suggested in cryptography as the basis for a TRNG, typically based on phase noise or frequency variations. But a crystal oscillator is deliberately designed for high frequency stability; it is thus the worst possible type of oscillator from which to obtain and exploit frequency variations. And crystal oscillator phase noise (which we see as edge jitter) is typically tiny and must be detected on a cycle-by-cycle basis, because it does not accumulate. Detecting a variation of, say, a few picoseconds in each 100nSec period of a typical 10 MHz oscillator is not something we do on an ordinary computer.

Another common approach to a crystal oscillator TRNG is to XOR many such oscillators, thus getting a complex high-speed waveform. (The resulting digital signal rate increases as the sum of all the oscillators.) Unfortunately, the high-speed and asynchronous nature of the wave means that setup and hold times cannot be guaranteed to latch that data for subsequent use. (Latching is inherent in, say, reading a value from a computer input port.) That leads to statistical bias and possible metastable operation. Futher, the construction is essentially linear and may power up similarly each time it is turned on.

Current

The measure of electron flow, in amperes. A current of one amp is one coulomb per second, or about 6.24 x 10^{18} electrons per second. Conventional current flow is in the direction *opposite* to the flow of electrons, because electrons have been assigned a negative charge. The movement of electrons is a negative flow, which is generally equivalent to a positive flow in the opposite direction.

Current is analogous to the amount of water *flow,* as opposed to *pressure* or voltage. A flowing electrical current will create a magnetic field around the conductor. A changing electrical current may create an electromagnetic field.

Cycle

1. Something which repeats over and over, like a wheel turning.
2. One full period of a repetitive signal.
3. In a FSM, a path which repeats endlessly. As opposed to an arc. In any FSM of finite size, every state sequence must eventually repeat or lead into a sub-sequence which repeats; every state eventually leads to a cycle.

In some RNG constructions, (e.g., BB&S and the Additive RNG) the system consists of multiple independent cycles, possibly of differing lengths. Since having a cycle of a guaranteed length is one of the main requirements for an RNG, the possibility that a short cycle may exist and be selected for use can be disturbing.

Cyclic Group

A simple group in which every element is produced by one of the elements, and that element is called a generator.

Cypher

In the U.S., an overly-cute misspelling of cipher.

Data

1. Values, especially when coded for electronic representation.
2. Facts, before being processed into "information." (The "information" then may be interpreted as "knowledge.")

Data Compression

The ability to represent data in forms which take less storage than the original. The limit to this is the amount of uniqueness in the data.

Sometimes people claim that they have a method to compress a file, and that they can compress it again and again, until it is only a byte long. Unfortunately, it is *impossible* to compress all possible files down to a single byte each, because a byte can only select 256 different results. And while each byte value might represent a whole file of data, only 256 such files could be selected or indicated.

Normally, compression is measured as the percentage size reduction; 60 percent is a good compression for ordinary text.

In general, compression occurs by representing the most-common data values or sequences as short code values, leaving longer code values for less-common sequences. Understanding which values or sequences are more common is the "model" of the source data. When the model is wrong, and supposedly less-common values actually occur more often, that same compression may actually *expand* the data.

Data compression is either "lossy," in which some of the information is lost, or "lossless" in which all of the original information can be completely recovered. Lossy data compression can achieve far greater compression, and is often satisfactory for audio or video information (which are both large and may not need exact reproduction). Lossless data compression must be used for binary data such as computer programs, and probably is required for most cryptographic uses.

Compression and Encryption

Compressing plaintext data has the advantage of reducing the size of the plaintext, and, thus, the ciphertext as well. Further, data compression tends to remove known characteristics from the plaintext, leaving a compressed result which is more random. Data compression can simultaneously *expand* the unicity distance and *reduce* the amount of ciphertext available which must exceed that distance to support attack. Unfortunately, that advantage may be most useful with fairly short messages. Also see: Ideal Secrecy.

One goal of cryptographic data compression would seem to be minimize the statistical structure of the plaintext. Since such structure is a major part of cryptanalysis, that would seem to be a major advantage. However, we also assume that our opponents are familiar with our cryptosystem, and they can use the same decompression we use. So the opponents get to see the structure of the original plaintext simply by decompressing any trial decryption they have. And if the decompressor cannot handle every possible input value, it could actually assist the opponent by identifying wrong decryptions.

When using data compression with encryption, one pitfall is that many compression schemes add recognizable data to the compressed result. Then, when that compressed result is encrypted, the "recognizable data" represents known plaintext, even when only the ciphertext is available. Having some guaranteed known plaintext for every message could be a very significant advantage for opponents, and unwise cryptosystem design.

It is normally impossible to compress random-like ciphertext. However, some cipher designs do produce ciphertext with a restricted alphabet which can of course be compressed. Also see entropy.

Bijective Compression

Another possibility is to have a data decompressor that can take any random value to some sort of grammatical source text. That may be what is sometimes referred to as bijective compression. Typically, a random value would decompress into a sort of nonsensical "word salad" source text. However, the statistics of the resulting "word salad" could be very similar to the statistics of a correct message. That could make it difficult to computationally distinguish between the "word salad" and the correct message. If "bijective compression" imposes an attack requirement for human intervention to select the correct choice, that might complicate attacks by many orders of magnitude. The problem, of course, is the need to devise a compression scheme that decompresses random values into something grammatically similar to the expected plaintext. That typically requires a very extensive statistical model, and of course at best only applies to a particular class of plaintext message.

An extension of the "bijective" approach would be to add random data to compressed text. Obviously, there would have to be some way to delimit or otherwise distinguish the plaintext from the added data, but that may be part of the compression scheme anyway. More importantly, the random data probably would have to be added between the compressed text in some sort of keyed way, so that it could not easily be identified and extracted. The keying requirement would make this a form of encryption. The result would be a homophonic encryption, in that the original plaintext would have many different compressed representations, as selected by the added random data. Having many different but equivalent representations allows the same message to be sent multiple times, each time producing a different encrypted result. But it is also potentially dangerous, in that the compressed message expands by the amount of the random data, which then may represent a hidden channel. Since, for encryption purposes, any random data value is as good as another, that data could convey information about the key and the user would never know. Of course, the same risk occurs in message keys or, indeed, almost any nonce.

Data Fabrication

The creation of false data. Intentional data fabrication for personal gain is a form of fraud. An issue of scientific integrity.

Data Falsification

The alteration of data, including selective omission. Intentional data falsification for personal gain is a form of fraud. An issue of scientific integrity.

Data Security

The extent to which data are secure. Protection of software and data from unauthorized modification, destruction, or disclosure. May or may not include cryptography.

dB

decibel.

DC

1. In cryptography, Differential Cryptanalysis.
2. In electronics, direct current: Electrical power which flows in one direction, more or less constantly. As opposed to AC, which frequently reverses direction.

Most electronic devices require DC -- at least internally -- for proper operation, so a substantial part of modern design is the "power supply" which converts 120 VAC wall power into 12 VDC, 5 VDC and/or 3 VDC as needed by the circuit and active devices.

Debug

The interactive analytical process of correcting bugs in the design of a complex system. A normal part of the development process.

Contrary to naive expectations, a complex system almost never performs as desired when first realized. Both hardware and software system design environments generally deal with systems which are *not* working. When a system *really* works, the design and development process is generally over.

Debugging involves identifying problems, analyzing the source of those problems, then changing the construction to fix the problem. (Hopefully, the fix will not itself create new problems.) This form of interactive analysis can be especially difficult because the realized design may not actually be what is described in the schematics, flow-charts, or other working documents: To some extent the real system is unknown.

The most important part of debugging is to understand in great detail exactly what the system is supposed to do. In hardware debugging, it is common to repeatedly reset the system, and start a known sequence of events which causes a failure. Then, if one really does know the system, one can probe at various points and times and eventually track down the earliest point where the implemented system diverges from the design intent. The thing that causes the divergence is the bug. Actually doing this generally is harder than it sounds.

Software debugging is greatly aided by a design and implementation process that decomposes complex tasks into small, testable procedures or modules, and then actually testing those procedures. Of course, sometimes the larger system fails anyway, in which case the procedure tests were insufficient, but they can be changed and the fixed procedure re-tested. Sometimes the hardest part of the debugging is to find some set of conditions that cause the problem. Once we have that, we can repeatedly run through the code until we find the place where the expected things do not occur.

Debugging real-time software is compounded by the difficulty of knowing the actual sequence of operations. This frequently differs from our ideal model of multiple completely independent software processes.

- One possibility is to build in some sort of "hidden screen" that displays important system data, like the amount of free memory, disk space, current buffer situation, etc.
- Another possibility is to create a real-time log of which procedures were entered when, on which thread, with their parameters. This code can be added to the start of the normal code in each routine, and enabled with a global flag.
- One might think to comment-out the logging code eventually, or to be able to conditionally compile that code away, but bugs generally do not give us a chance to re-compile with the debug code before they appear. We need to be able to set a system flag at runtime to turn data logging on and off.
- There are various ways to implement real-time logging:
    - One way might be to have some sort of high-speed link to another system which records that data for later analysis.
    - Alternately, it may be possible to have another thread which does the logging in the system itself, which should of course be as simple and reliable as possible.
    - Yet another approach might be to allocate a fairly-substantial memory buffer, and just write the log data sequentially into memory. In this case we want to stop logging as soon as possible after the bug occurs so critical data are not overwritten.
- When analyzing the data, check every possibility, no matter how unlikely: At gigahertz instruction rates, even very, very unlikely things can happen fairly often. Whether something is improbable or not is irrelevant if it is the bug.

Two of the better books on the debug process include:

Agans, D. 2002. *Debugging. The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems.* AMACOM.

Telles, M. and Y. Hsieh. 2001. *The Science of Debugging.* Coriolis.

Poster graphics with the 9 rules can be found at: http://www.debuggingrules.com/, but the shorthand form may seem somewhat arcane unless one already knows the intent. The best thing is to read the book. Failing that, however, I offer my interpretation, plus a few insights:

1. **Know, in great detail, exactly how the system is supposed to work.** If we do not know when things are right or wrong, how can we trace back to the first wrong thing?
2. **Find a situation which causes the system to misbehave.** Then we can reproduce it often and track it down. But if the problem is intermittent, it may be necessary to collect data on the whole system repeatedly until the problem is detected to get data worth analyzing.
3. **Don't waste much time theorizing; instead collect real data from the actual system until the data**

show the problem. Bugs tend to be odd or improbable, because the more obvious things will have been addressed already. It is often a waste of time to look at the "likely" sources of the problem, and difficult to imagine "unlikely" sources.
4. **Try to identify sections of the system that are working correctly.** If we can identify the problem with a particular section, we often can eliminate much of the system as the cause. Sometimes we can "comment out" calls to routines without changing the problem, and then know those routines are *not* a cause.
5. **Change at most one thing at a time.** This classic statement is well-known throughout experimental Science and Engineering, for if we change even two things, and the problem is affected, we do not know which change caused the effect. Although tedious, we must run the system and check for the problem after every single change. And if a change does not help, *change it back*, so that the design documents and printouts match what we really have. Collect proposed design or implementation changes for use *after* the bug is found.
6. **Log every debug action, in sequence, for later review.** Especially make copies of the system before changes. The copies and log entries should make it possible to undo any significant action, and to check back on symptoms or results from previous experiments.
7. **Actually check the real system and make sure all your assumed values are in fact what you think.** Just the fact that you are working on a bug means the system is not functioning as assumed. The documentation and listings may themselves be wrong.
8. **Describe the problem to someone else.** The simple act of organizing what we know so we can describe it may help to see the problem from another angle. Keep to the facts and avoid jumping to conclusions.
9. **Continue until you can fix the problem and then replace it again at will.** Only when we can control the bug effect can we be sure that the real problem has been identified. It is not sufficient to assume the problem has somehow "gone away" during debugging and not know how that happened.

Decade

A ratio of 10:1. Often a frequency ratio of 10:1. Also see octave.

Deception

A deliberate attempt to hide the truth, or to delude someone as to what the truth really is.

Decipher

The process which can reveal the information or plaintext hidden in message ciphertext (provided it is the correct process, with the proper key). The inverse of encipher.

Decoupling

In electronics, the concept of isolating circuit stages from a common power supply, and from each other. The basic concept may be useful in modern digital electronics. See bypass and amplifier.

Each stage in a typical radio circuit might use 100 ohm resistors from power, and then a bypass capacitor of perhaps 0.01uF to circuit ground. The RF signals in one stage are attenuated at the common power bus by the frequency-sensitive voltage divider or filter consisting of the resistor and the power supply capacitors. Any RF signals on the power supply bus are attenuated again by the resistor to the next stage and that bypass capacitor.

Since a stage current of 20mA could cause a 2V drop across the 100 ohm resistor, smaller resistors, or even chokes using ferrite beads, might be used instead.

Decryption

The general term for extracting information which was hidden by encryption.

Decibel

Ten times the base-10 logarithm of the ratio of two power values. Denoted by dB. One-tenth of a bel.

When voltages or currents are measured, power changes as the square of these values, so a decibel is twenty times the base-10 logarithm of the ratio of two voltages or currents.

For example, suppose we have an active filter with a voltage gain of 1 at 425Hz and a gain of 6 at 2550Hz:

```
  amplitude change in dB = 20 log10 (e2/e1) = 20 log10 6 = 15.56 dB
```

Decimal

Base 10: The numerical representation in which each digit has an alphabet of ten symbols, usually 0 through 9. Also see: binary, octal, and hexadecimal.

Decimation

Every tenth. In general, selecting every n^{th} element. Sometimes used to hide exploitable patterns produced by an LFSR or other random number generator.

Supposedly the name of the ancient Roman practice of lining up soldiers who had run from battle, and selecting every tenth one to be killed by those not selected.

Deductive Reasoning

In the study of logic, reasoning about a particular case from one or more general statements assumed true; a rigorous proof. (The general statements are premises in a logical argument, and are typically based on assumption or evidence). Necessarily limited to describing the consequences of known or assumed truth. In contrast to inductive reasoning. Also see: fallacy and especially belief.

Defined Plaintext

The information condition of an opponent being able to create plaintext which is then enciphered under the unknown key to produce ciphertext which is also available. The defined plaintext condition can be seen as a further restriction to, and thus a form of, the known plaintext condition. Also see: defined plaintext attack and ciphertext only.

Defined Plaintext Attack

Any attack which takes place under defined plaintext information conditions. Clearly, when the opponent already knows both the plaintext and the associated ciphertext, the only thing left to find is the key. Also see known plaintext attack and ciphertext only attack.

A defined plaintext attack typically needs to send a particular plaintext value to the internal cipher (thus "knowing" that value), and get the resulting ciphertext. Typically, a large amount of plaintext is needed under a single key. A cipher system which prevents any one of the necessary conditions also stops the corresponding attacks.

Many defined plaintext attacks are interactive, and so require the ability to choose subsequent plaintext based on previous results, all under one key. It is relatively easy to prevent interactive attacks by having a message key facility, changing message keys on each message, and by handling only complete messages instead of a continuing flow or stream that an opponent can modify interactively.

Known plaintext (and, thus, defined plaintext) attacks can be opposed by:

- limiting the amount of data enciphered under one key,
- changing keys frequently,
- using multiple encryption and so hiding the actual plaintext or ciphertext or both for each internal cipher, and
- enciphering each message under a different random message key which users cannot control. (And that is something we have every right to expect will be done by even the most basic cipher system.)

Defined plaintext attacks can be opposed by:

- allowing only authorized users to encipher data,
- using a block cipher operating mode (such as CBC) that changes the data before it is actually enciphered (although that would still be weak to interactive attacks).

Interactive defined plaintext attacks can be opposed by:

- ciphering only complete, pre-existing messages.

Most modern cipher systems will use a message key (or some similar facility), making defined-plaintext attacks generally more academic than realistic (also see break).

Degree

1. In algebra, for expressions in one variable, the highest power of that variable.
2. In geometry, a measure of angle, 1/360th of a full rotation.

Degrees of Freedom

In statistics, the number of completely independent values in a sample. The number of sampled values or observations or bins, less the number of defined or freedom-limiting relationships or "constraints" between those values.

If we choose two values completely independently, we have a DF of 2. But if we must choose two values such that the second is twice the first, we can choose only the first value independently. Imposing a relationship on one of the sampled value means that we will have a DF of one less than the number of samples, even though we may end up with apparently similar sample values.

In a typical goodness of fit test such as chi-square, the reference distribution (the expected counts) is normalized to give the same number of counts as the experiment. This is a constraint, so if we have N bins, we will have a DF of N - 1.

DeMorgan's Laws

In set theory and Boolean logic, the equivalence between between "sum of products" and "product of sums" expressions.

```
   NOT( A OR B)    = NOT(A) AND NOT(B)
   NOT( A AND B ) = NOT(A) OR  NOT(B)
```

so

```
   A OR  B = NOT( NOT(A) AND NOT(B) )
   A AND B = NOT( NOT(A) OR  NOT(B) )
```

Also see: Logic Function.

Depletion Region

In a crystalline semiconductor like a diode or bipolar transistor, the distance of ionization on both sides of a junction between P and N materials. Also called "depletion layer," "depletion zone," "junction region" and "space charge region." The "depletion" part of this is the lack of majority charge carriers in the area.

Normally, an N-type semiconductor has no net electrical charge, but does have an excess of electrons which are not in a stable bond. Similarly, on its own, P-type semiconductor also has no net charge, but has a surplus of *holes* or bond-positions which have no electrons. When the two materials occur in the same crystal lattice, there are opposing forces:

- Thermal energy may cause free electrons from the N material to find themselves in the P material where they can form a bond. Extra electrons add excess negative charge to formerly balanced atoms making them negative ions. Atoms to which the electrons originally belonged will have excess positive charge

as positive ions.
- As a consequence, a potential will build between the P and N materials, thus making it more difficult for N material electrons to travel into the P material, and less difficult for P material electrons to travel into the N material. At some potential, the thermal movement and accumulation of electrons between N and P balances out or is in equilibrium.

The result is a static potential or bias due solely to the material physics.

For diode conduction to occur, sufficient potential must be applied so that the depletion field effect is overwhelmed. The depletion field (or "junction voltage" or "barrier voltage") is typically about 0.6V in silicon.

DES

The conventional block cipher known as the U.S. Data Encryption Standard (DES). Recently replaced by AES.

DES is a 64-bit block cipher with a 56-bit key. However, a keyspace of 2^{56} keys is now too small for serious work. A keyspace of at least 80 key bits is now recommended, and AES jumped to 128 and 256 key bits. One possibility for DES is to form the multiple encryption known as Triple DES: three sequential cipherings by DES, each with an independent key. That should produce an expected strength of something like 112 bits, which is more than enough to defeat brute force attacks.

The mechanics of DES are widely available elsewhere. Here I note how one particular issue common to modern block ciphers is reflected in DES. A common academic model for conventional block ciphers is a "family of permutations." The issue is the size of the implemented keyspace compared to the size of the potential keyspace for blocks of a given size.

For 64-bit blocks and 56-bit keys, DES provides:
- 2^{56} keyed or emulated tables, out of about
- $2^{1,000,000,000,000,000,000,000}$ possibilities.

The obvious conclusion is that *almost none* of the keyspace implicit in the model is actually implemented in DES, and that is consistent with other modern block cipher designs. While that does not make modern ciphers weak, it is a little disturbing. See more detailed comments under AES.

Degenerate Cycle

In an FSM, a cycle with only one state, which thus does not "cycle" through different states, and so does not change its output value.

Design Strength

The keyspace; the effort required for a brute force attack.

Deterministic

A process whose sequence of operations is fully determined by its initial state. A mechanical or clockwork-like process whose outcome is inevitable, given its initial setting. Pseudorandom. See: finite state machine. As opposed to stochastic.

Conventional block ciphers are completely deterministic, unless they include a random homophonic feature. Computational random number generators also are deterministic (again, see: finite state machine). Some specially-designed physical devices produce nondeterministic sequences (see: really random).

Deterministic Finite Automata

The theoretical concepts of deterministic finite automata (DFA) are usually discussed as an introduction to lexical analysis and language grammars, as used in compiler construction. In that model, computation occurs in a finite state machine, and the computation sequence is modeled as a network, where each node represents a particular state value and there is exactly one arc from each node to one other node. Normal program execution steps from the initial node to another node, then another, step by step, until the terminal node is reached. In general, the model corresponds well both to software sequential instruction execution, and to synchronous digital hardware operations controlled by hardware clock cycles.

Within this theory, however, at least two different definitions for "nondeterministic" are used:

1. a FSM which can pass through multiple state values at a single step, and
2. a FSM which has more than one exit path from at least one node.

Such executions can be called "nondeterministic" in sense (1) that the number of execution cycles may vary, and sense (2) that the execution path may vary, from run to run. Unfortunately, this established model of FSM determinism can lead to confusion in cryptographic analysis.

Nondeterministic Cryptographic Computation

In cryptography, we are usually interested in "nondeterministic" behavior to the extent that it is unpredictable. One example of cryptographic nondeterministic behavior would be a really random sequence generator. Programs, on the other hand, including most random number generators, are almost always deterministic. Even programs which use random values to select execution paths generally get those values from deterministic statistical RNG's, making the overall computation cryptographically predictable.

One issue of cryptographic determinism is the question of whether user input or mechanical latencies can make a program "nondeterministic." To the extent that user input occurs at a completely arbitrary time, that should represent some amount of uncertainty or entropy. In reality, though, it may be that certain delays are more likely than others, thus making the uncertainty less than it might seem. Subsequent program steps based on the user input cannot increase the uncertainty, being simply the expected result of a particular input.

If hardware device values or timing occur in a completely unpredictable manner, that should produce some amount of uncertainty. But computing hardware generally is either completely exposed or strongly predictable. For example, disk drives can appear to have some access-time uncertainty based on the prior position of the read arm and rotational angle of the disk itself. But if the arm and disk position information is known, there is relatively little uncertainty about when the desired track will be reached and the desired sector read.

If disk position state was actually *unknowable*, we could have a source of cryptographic uncertainty. But disk position is *not* unknowable, and indeed is fairly well defined after just a single read request. Subsequent operations might be largely predictable. Normally we do not consider disk position state, or other computer hardware state, to be sufficiently protected to be the source of our security. In the best possible case, disk position state *might* be hidden to *most* opponents, but that hope is probably not enough for us to assign cryptographic uncertainty to those values.

Similar arguments pertain to most automatic sources of supposed computer uncertainty. Also see really random.

Deus ex Machina

Latin for: "god from the Machine," translated from the even older Greek. Originally, a device for playwrites to resolve even the most complex plot line, as in: "A machine with a god hidden inside appears and fixes everything."

More generally, the idea that results unimaginable to any normal person or impossible for any normal design are actually from a god inside and not the machine itself.

DH

Diffie-Hellman.

Dialectic

Discussion and reasoning by dialog. Logical argumentation. Also see rhetoric.

Dichotomy

Greek for "cut in two." In the study of logic and argumentation, the division of a topic into exactly two normally distinct and mutually-exclusive categories. The goal is to have a single clear distinction by which every item in the universe of discourse can be assigned to one class or the other. The power of the technique obviously depends upon the extent to which the above goal is achieved.

A true dichotomy adds tremendous power to analysis by identifying particular effects with particular categories. Finding one such effect thus identifies a category, which then predicts the rest of the effects of that category and the lack of the effects of the opposing category. And when only two categories exist, even seeing the *lack* of the effects from one category necessarily implies the other.

Dictionary Attack

Typically an attack on a secret password. A dictionary of common passwords is developed, and a brute force attack conducted by stepping through each word from the dictionary.

Differential Cryptanalysis

(DC). A form of attack which exploits the observation that a fixed bit difference between plaintext values sometimes produces a predicted bit difference between ciphertext values, depending upon the key. Typically used to expose the key-bits in the last round of a Feistel construction. Typically a "chosen plaintext" or defined plaintext attack, where the opponent can create vast amounts of the exact data to go into a cipher, and then see the resulting ciphertext. Also see Linear Cryptanalysis.

The basic idea of Differential Cryptanalysis is to first cipher some plaintext, then make particular changes in that plaintext and cipher it again. Particular ciphertext differences occur more frequently with some key values than others, so when those differences occur, particular keys are (weakly) indicated. With huge numbers of tests, false indications will be distributed randomly, but true indications always point at the same key values and so will eventually rise above the noise to indicate some part of the key.

The technique typically depends on exploiting various assumptions:

1. The cipher design uses S-boxes.
2. The cipher design has some causal relationship between key and S-box which expresses itself in detectable ciphertext differences.
3. The S-box contents are known, which allows probabilities to be calculated.
4. The cipher system allows the opponent to perform a defined plaintext attack.
5. The cipher system allows the opponent to generate and analyze sufficient text so that small difference probabilities can rise above random noise.

Typically, differential weakness can be eliminated at cipher design time by the use of keyed S-boxes.

Typically, differential attacks can be prevented at the cipher system level, by:

- Multiple encryption, which prevents known plaintext (and, thus, defined plaintext) information exposure for a component cipher.
- Limiting the amount of data enciphered under a single key (which should be done in any case).

Also see

- "Differential Cryptanalysis: A Literature Survey," locally, or @: "http://www.ciphersbyritter.com/RES/DIFFANA.HTM
- and the Differential Cryptanalysis discussion, locally, or @:

"http://www.ciphersbyritter.com/NEWS4/DIFRENTL.HTM

Differential Mode

In electronics, the difference signal between two conductors. On balanced line this is the desired signal, and common mode signals are intentionally ignored.

Diffie-Hellman

A public key scheme suitable for the private exchange of secret keys, which are then used for actual data ciphering.

Diffusion

Diffusion is the property of an operation such that changing one bit (or byte) of the input will change adjacent or near-by bits (or bytes) after the operation. In a block cipher, diffusion propagates bit-changes from one part of a block to other parts of the block. Diffusion requires mixing, and the step-by-step process of increasing diffusion is described as avalanche. Diffusion is in contrast to confusion.

Normally we speak of *data* diffusion, in which changing a tiny part of the plaintext data may affect the whole ciphertext. But we can also speak of *key* diffusion, in which changing even a tiny part of the key should change each bit in the ciphertext with probability 0.5.

Perhaps the best diffusing component is substitution, but this diffuses only within a single substituted value. Substitution-permutation ciphers extend diffusion beyond a single value by moving the bits of each substituted element to other elements, substituting again, and repeating. But this only provides guaranteed diffusion if particular substitution tables are constructed.

Another alternative for extending diffusion to other elements is to use some sort of Balanced Block Mixing. BBM constructions inherently have guaranteed diffusion which I call ideal mixing. Still another alternative is a Variable Size Block Cipher construction. Also see overall diffusion.

Digital

Pertaining to discrete or distinct finite values. As opposed to analog or continuous quantities. Values reasonably represented as Boolean or integer quantities, as opposed to values that require real numbers for accurate representation.

Digital Signature

A value which provides authentication in an electronic document. Typically the result from a cryptographic hash of the document.

Diode

The idealized model of an electronic device with two terminals which allows current to flow in only one direction.

In practice, real diodes do allow some "leakage" current in the reverse direction, and leakage approximately doubles for every 10degC increase in temperature.

Semiconductor junctions also have a "forward" voltage (typically 0.6V in silicon, or 0.3V in germanium and for Schottky devices in silicon) which must be exceeded for conduction to occur. This bias voltage is basically due to the semiconductor depletion region. The forward voltage has a negative temperature coefficient of about -2.5mV/degC, in either silicon or germanium.

Semiconductor junctions also have a dynamic resistance (for small signals) that varies inversely with current:

```
   r = 25.86 / I
```

```
where:
   r = dynamic resistance in ohms
   I = junction current (mA)
```

As a result of the forward voltage and internal resistance, power is dissipated as heat when current flows. Diodes are made in a range from tiny and fast signal devices through large and slow high power devices packaged to remove internal heat. There is a wide range of constructions for particular properties, including photosensitive and *light-emitting* diodes (LED's).

All real diodes have a reverse "breakdown" voltage, above which massive reverse conduction can occur. (See avalanche multiplication and Zener breakdown.) Real devices also have current, power, and temperature limitations which can easily be exceeded, a common result being a wisp of smoke and a short-circuit connection where we once had a diode. However, if the current is otherwise limited, diode breakdown can be exploited as a way to "regulate" voltage (although IC regulator designs generally use internal "bandgap" references for better performance).

All diodes break down, but those specifically designed to do so at particular low voltages are called *Zener* diodes (even if they mainly use avalanche multiplication). A semiconductor junction in reverse voltage breakdown typically generates good-quality noise which can be amplified and exploited for use (or which must be filtered out). A bipolar transistor base-emitter junction can be used instead of a Zener or other diode. Another noise generation alternative is to use an IC which has a documented and useful noise spectrum, such as the "IC Zener" LM336 (see some noise circuits locally, or @: http://www.ciphersbyritter.com/NOISE/NOISRC.HTM). Also see avalanche multiplication and Zener breakdown.

Distinguisher

In cryptanalysis, a function or process or attack which shows that a cipher or hash does not behave as the expected theoretical model would predict. Often presented in the context of conventional block cipher designs.

A distinguisher makes a contribution to cryptanalysis by showing that a model does not work for the particular cipher. The problem comes in properly understanding what it means to not model the reality under test. In many cases, a successful distinguisher is presented as a successful attack, with the stated or implied result being that the cipher is broken. But that goes beyond what is known. When a scientific model is shown to not apply to reality, that does not make reality "wrong." It just means that the tested model is not useful. Maybe the best implication is that a new model is needed.

Distinguishers generally come under the heading of "computational indistinguishability," and much of this activity occurs in the area of conventional block ciphers. One of the problems of that area is that many cryptographers interpret "block cipher" as an emulated, huge simple substitution (that is, a key-selected pseudorandom permutation). But it is entirely possible for ciphers to work on blocks yet not fit that model. Clearly, if a cipher does not really function in that way, it may be possible to find a distinguisher to prove it.

The real issue in cipher design is strength. The problem is that we have no general measure to give us the strength of an abstract cipher. But a distinguisher provides testimony about conformance to model, not strength. A distinguisher simply does not testify about weakness.

In normal cryptanalysis we start out knowing plaintext, ciphertext, and cipher construction. The only thing left unknown is the key. A practical attack must recover the key. If it does not, it is not a real attack after all.

Distribution

In statistics, the range of values which is a random variable, and the probability that each value or range of values will occur. Also the probability of test statistic values for the case "nothing unusual found," which is the null hypothesis.

If we have a *discrete* distribution, with a finite number of possible result values, we can speak of "frequency" and "probability" distributions: The "frequency distribution" is the expected *number* of occurrences for each possible value, in a particular sample size. The "probability distribution" is the *probability* of getting each value, normalized to a probability of 1.0 over the sum of all possible values.

Here is a graph of a typical "discrete probability distribution" or "discrete probability density function," which displays the probability of getting a particular statistic value for the case "nothing unusual found":

```
0.1|        ***
   |       *   *         Y = Probability of X
 Y |     **     **       y = P(x)
   | ****         ****
0.0 -------------------
             X
```

Unfortunately, it is not really possible to think in the same way about continuous distributions: Since continuous distributions have an infinite number of possible values, the probability of getting any *particular* value is zero. For continuous distributions, we instead talk about the probability of getting a value in some subrange of the overall distribution. We are often concerned with the probability of getting a particular value or below, or the probability of a particular value or above.

Here is a graph of the related "cumulative probability distribution" or "cumulative distribution function" (c.d.f.) for the case "nothing unusual found":

```
1.0|            ******
   |          **           Y = Probability (0.0 to 1.0) of finding
 Y |         *                 a value which is x or less
   |       **
0.0 -******-------------
             X
```

The c.d.f. is just the sum of all probabilities for a given value or less. This is the usual sort of function used to interpret a statistic: Given some result, we can look up the probability of a lesser value (normally called *p*) or a greater value (called *q* = 1.0 - *p*).

Usually, a test statistic is designed so that extreme values are not likely to occur by chance in the case "nothing unusual found" which is the null hypothesis. So if we *do* find extreme values, we have a strong argument that the results were not due simply to random sampling or other random effects, and may choose to reject the null hypothesis and thus accept the alternative hypothesis.

Usually the ideal distribution in cryptography is "flat" or uniform. Common *discrete* distributions include:

- binomial, and
- Poisson.

A common *continuous* distribution is:

- normal.

Also see the Ciphers By Ritter / JavaScript computation pages (locally, or @: http://www.ciphersbyritter.com/index.html#JavaScript).

Distributive

In abstract algebra, the case of a dyadic operation, which may be called "multiplication," which can be applied to equations involving another dyadic operation, which may be called "addition," such that: a(b + c) = ab + ac and (b + c)a = ba + bc.

Also see: associative and commutative.

Divide and Conquer

The general concept of being able to split a complexity into several parts, each part naturally being less complex than the total. If this is possible, the opponent may be able to solve all of the parts far easier than the supposedly complex whole. Often part of an attack.

This is a particular danger in cryptosystems, since most ciphers are built from less-complex parts. Indeed, a major role of cryptographic design is to combine small component parts into a larger complex system which cannot be split apart.

Dogma

A set of beliefs expected or required of those in a particular organization or field.

Domain

The set of all input values or arguments *x* which are allowed to a mapping or function *f(x)*. All numbers in the set *X* for which *f(x)* is defined. Also see range.

Double Shuffling

Doing the shuffle algorithm twice. Double shuffling prevents the resulting table contents from revealing the sequence used to construct those contents.

Typically, shuffling permutes the contents of a substitution table or block as a result of a sequence from a keyed random number generator (RNG). Essentially, shuffling allows us to efficiently key-select an arbitrary permutation from among all possible permutations, which is the ideal sort of balanced selection.

If shuffling is implemented so the shuffling sequence is used as efficiently as possible, simply knowing the resulting permutation should suffice to reconstruct the shuffling sequence, which is the first step toward attacking the RNG. While common shuffle implementations do discard some of the sequence, we can *guarantee* to use at least twice as much information as the table or block can represent simply by shuffling *twice*. Double-shuffling will not produce any more permutations, but it should prevent the mere contents of a permuted table or block from being sufficient to reconstruct the original shuffling sequence.

In a sense, double-shuffling is a sort of one-way information valve which produces a key-selected permutation, and also hides the shuffling sequence which made the selection.

Also see Dynamic Transposition.

DSA

Digital Signature Algorithm, published by NIST.

DSP

1. Digital Signal Processor. A type of microprocessor optimized for fast multiplication and value accumulation as often needed in signal processing operations like FFT.
2. Digital Signal Processing. The translation of analog signals into a stream of digital values, which can be analyzed and modified by mathematical computation.

DSS

Digital Signature Standard, based on DSA.

Due Care

A term of art in law: A legal standard of expected care or responsibility for safety owed to someone or society.

Typically the normal precautions of a reasonable person.

Due Diligence

A term of art in finance and law: An expectation of thorough investigation. It is expected that a buyer actually check the substance of claims and the assumptions involved for any transaction or investment opportunity.

Dyadic

Relating to *dyad*, which is Greek for dual or having two parts. In particular, a function with two inputs or arguments. Also see: arity, monadic, unary and binary.

Dynamic Keying

That aspect of a cipher which allows a key to be changed with minimal overhead. A dynamically-keyed block cipher might impose little or no additional computation to change a key on a block-by-block basis. The dynamic aspect of keying could be just one of multiple keying mechanisms in the same cipher.

One way to have a dynamic key in a block cipher is to include the key value along with the plaintext data (also see block code). But this is normally practical only with blocks of huge size, or variable size blocks. (See huge block cipher advantages.)

Another way to have a dynamic key in a block cipher is to add a confusion layer which mixes the key value with the block. For example, exclusive-OR could be used to mix a 64-bit key with a 64-bit data block.

Dynamic Substitution Combiner

The combining mechanism described in U.S. Patent 4,979,832. See

- U.S. Patent 4,979,832 locally, or @: http://www.ciphersbyritter.com/PATS/DYNSBPAT.HTM
- and the list of Dynamic Substitution info and articles locally, or @: http://www.ciphersbyritter.com/index.html#DynSubTech

The main goal of Dynamic Substitution is to provide a stream cipher combiner with reduced vulnerability to known-plaintext attack. In contrast, a stream cipher using a conventional additive combiner will immediately and completely expose the confusion sequence under known-plaintext attack. That gives the opponents the chance to attack the stream cipher internal RNG, which is the common stream cipher attack. But a Dynamic Substitution combiner hides the confusion sequence, thus complicating the usual attack on the stream cipher RNG.

Dynamic Substitution is a substitution table in which the arrangement of the entries changes during operation. This is particularly useful as a strong replacement for the strengthless exclusive-OR combiner in stream ciphers.

In the usual case, an invertible substitution table is keyed by shuffling under the control of a random number generator. One combiner input is used to select a value from within the table to be the result or output; that is normal substitution. But the other combiner input is used to select an entry at random, then the values of the two selected entries are exchanged. So as soon as a plaintext mapping is used for output, it is immediately reset to any possibility, and the more often any plaintext value occurs, the more often that particular transformation changes.

The arrangement of a keyed substitution table starts out unknown to an opponent. From the opponent's point of view, each table entry could be any possible value with uniform probability. But after the first value is mapped through that table, the just-used table entry or transformation is at least potentially exposed, and no longer can be considered unknown. Dynamic Substitution acts to make the used transformation again completely unknown and unbiased, by allowing it to again take on any possible value. Thus, the amount of information leaked about table contents is replaced by information used to re-define each just-used entry.

Also see Cloak2, Penknife, Dynamic Transposition, Balanced Block Mixing, Mixing Cipher, Mixing Cipher design strategy and Variable Size Block Cipher.

Dynamic Transposition

A block cipher which first creates an exact or approximate bit balance within each block, and then shuffles the bits within a block, each block being permuted independently from a keyed random number generator. See transposition and transposition cipher.

One goal of Dynamic Transposition is to leverage the concept of Shannon Perfect Secrecy on a block-by-block basis. In contrast to the usual *ad hoc* strength claims, Perfect Secrecy has a fundamental basis for understanding and believing cipher strength. That basis occurs when the ciphering operation can produce any possible transformation between plaintext and ciphertext. As a result, even brute force no longer works, because running through all possible keys just produces all possible block values. And, in contrast to conventional block ciphers, which actually implement only an infinitesimal part of their theoretical model, each and every Dynamic Transposition permutation can be made practically available.

One interesting aspect of Dynamic Transposition is a fundamental hiding of each particular ciphering operation. Clearly, each block is ciphered by a particular permutation. If the opponent knew which permutation occurred, that would be useful information. But the opponent only has the plaintext and ciphertext of each block to expose the ciphering permutation, and a vast plethora of different permutations each take the exact same plaintext to the exact same ciphertext. (This is because wherever a '1' occurs in the ciphertext, any plaintext '1' would fit.) As a consequence, even known-plaintext attack does not expose the ciphering permutation, which is information an opponent would apparently need to know. The result is an unusual block cipher with an unusual fundamental basis in strength.

Also see

- the latest article: "Dynamic Transposition Revisited Again" locally, or @: http://www.ciphersbyritter.com/ARTS/DYNTRAGN.HTM
- the conversation "Dynamic Transposition Ciphering" locally, or @: http://www.ciphersbyritter.com/NEWS5/REDYNTRN.HTM
- my *Cryptologia* article "Transposition Cipher with Pseudo-Random Shuffling: The Dynamic Transposition Combiner locally, or @: http://www.ciphersbyritter.com/ARTS/DYNTRAN2.HTM

Also see Dynamic Substitution Combiner, Balanced Block Mixing, Mixing Cipher, Mixing Cipher design strategy and Variable Size Block Cipher.

Ebers-Moll Model

A numerical model of bipolar transistors.

In the simple model of bipolar transistor operation, base-emitter current is multiplied by transistor h_{FE} or beta (B), thus producing amplified collector-emitter current:

$$I_c = h_{FE} * I_b$$

I_c = collector current
h_{FE} = the Forward common-Emitter h-parameter, or beta
I_b = base current

And while that is a reasonable rule of thumb, the simple model is not very accurate.

A far more accurate computational model is Ebers-Moll:

```
I_c = I_s[ e**(V_be / V_t) - 1 ]
V_t = kT / q

I_c  = collector current
I_s  = saturation current (reverse leakage)
e    = the base of natural logs
V_be = voltage between base and emitter

k = Boltzmann's constant (1.380662 * 10**-23 joule/deg K)
T = temperature (deg. Kelvin)
q = electron charge (1.602189 * 10**-19 coulombs)
```

In Ebers-Moll the collector current is a function of base-emitter *voltage*, not current. Unfortunately, base-emitter voltage V_{be} itself varies both as a function of collector current (delta V_{be} = 60mV per power of ten collector current), and temperature (delta V_{be} = -2.1mV per deg C). V_{be} also varies slightly with collector voltage V_{ce} (delta V_{be} ~ -0.0001 * delta V_{ce}), which is known as the Early effect.

ECB

The electronic codebook operating mode.

ECC

1. Error Correcting Code.
2. Elliptic Curve Cipher.

ECDSA

Elliptic Curve DSA.

EDE

Encipher-Decipher-Encipher. A form of multiple encryption, where the same cipher is used in both encipher and decipher modes, with different keys (also see Pure Cipher). Usually, a form of Triple DES.

Efficiency

In a process using a resource to produce a result, the ratio of the result to the resource.

In cryptography two aspects of efficiency concern keys. Keys are often generated from unknowable bits obtained from really-random generators. When unknowable bits are a limited resource there is motive both to decrease the amount used in each key, and to increase the amount being generated. However, both of these approaches have the potential to weaken the system. In particular, insisting on high efficiency in really-random post-processing can lead to reversible processing which is the exact opposite of the goal of unknowable bits.

Extra unknowable bits may cover unknown, undetected problems both in key use (in the cipher), and in key generation (in the randomness generator). Because there is so much we do not know in cryptography, it is difficult to judge how close we are to the edge of insecurity. We need to question the worth of efficiency if it ends up helping opponents to break a cipher. A better approach might be to generate high quality unknowability, and use as much of it as we can.

Electric Field

The fundamental physical force resulting from the attraction of opposing charges.

Electromagnetic Field

The remarkable self-propagating physical field consisting of energy in synchronized and changing electric and magnetic fields. Energy in the electric or potential field collapses and creates or "charges up" a magnetic field.

Energy in the magnetic field collapses and "charges up" an electric field. This process allows physical electrical and magnetic fields which normally are short-range phenomena to "propagate" and thus carry energy over relatively large distances at the speed of light. Light itself is an electromagnetic field or *wave*. Other examples include "radio" waves (including TV, cell phones, etc.), and microwave cooking.

It is important to distinguish between a long-distance propagating electromagnetic field and simpler and more range-limited independent electric and magnetic fields.

It is unnecessary to consider how a field has been generated. Exactly the same sort of magnetic field is produced either by solid magnets or by passing DC current through a coil of wire making an electromagnet. A field from an electromagnet is not necessarily an electromagnetic field in the sense of a propagating wave; it is just another magnetic field.

Changing magnetic fields can be produced by forcing magnets to rotate (as in an alternator) or changing the current through an electromagnet. Typical dynamic magnetic field sources might include AC motor clocks, mixing motors, fans, or even AC power lines. It would be extremely difficult for low-frequency changes or physical movement to generate a propagating electromagnetic field.

Radio frequency voltage is the basis of most radio transmission. Radio antenna designs convert RF power into synchronized electric and magnetic fields producing a true electromagnetic field which can be radiated into space.

It is important to distinguish between the expanding or "radiating" property of an electromagnetic field, as opposed to the damaging ionizing radiation produced by a radioactive source.

As far as we know -- and a great many experiments have been conducted on this -- electromagnetic waves are not life-threatening (unless they transfer enough power to dangerously heat the water in our cells). The belief that electromagnetic fields are not dangerous is also *reasonable,* since light itself is an electromagnetic wave, and life on Earth developed in the context of the electromagnetic field from the Sun. Indeed, plants actually use that field to their and our great benefit.

Electromagnetic Interference

(EMI). Various forms of electromagnetic or radio waves which interfere with communication. This can include incidental interference, as from a running AC/DC motor, and interference from otherwise legitimate sources (as when a mobile transmitter is operated close to and thus overloads a receiver).

Reducing emitted EMI and dealing with encountered EMI is an issue in most modern electronic design. Also see TEMPEST and shielding.

Electronic

Having to do with the control and use of physical electrons, as electrical potential or voltage, electrical flow or current, and generally both, or power. See circuit, conductor, component, hardware, system and schematic diagram.

Electrostatic Discharge

ESD. In electronics, the instantaneous discharge of high voltage static electricity through invisible sparks or arcs due to breakdown of air insulation. Static charge can accumulate to very high voltages (but relatively low power) in normal activities, such as walking or sliding across a chair seat. ESD is significant in that the high voltage involved can break down the tiny PN junctions in semiconductor components even though the power involved is tiny. This breakdown can cause outright failure or -- worse -- degraded or unreliable operation.

To prevent ESD, we "simply" prevent the accumulation of static electricity, or prevent discharge through a sensitive device. Some approaches include the use of high-resistance ESD surfaces to keep equipment at a known potential (typically "ground"), conductive straps to connect people to the equipment (or "ground") before they touch it, and ample humidity to improve static discharge through air. Other measures include the use of "ESD shoes" to ground individuals automatically, the use of metalized insulated bags, and improved ESD protection in the devices themselves.

Unless grounded, two people are rarely at the same electrical potential or voltage, so handing a sensitive board or device from one person to another can complete a circuit for the discharge of static potential. That could be prevented by shaking hands before giving a board to another person, or by placing the board on an ESD surface to be picked up.

Electronic Codebook

(ECB). Electronic Codebook is an operating mode for block ciphers. Presumably the name comes from the observation that a block cipher under a fixed key functions much like a physical codebook: Each possible plaintext block value has a corresponding ciphertext value, and vise versa.

ECB is the naive method of applying a block cipher, in that the plaintext is simply partitioned into appropriate size blocks, and each block is enciphered separately and independently. When we have a small block size, ECB is generally unwise, because language text has biased statistics which will result in some block values being re-used frequently, and this repetition will show up in the raw ciphertext. This is the basis for a successful codebook attack.

On the other hand, if we have a large block (at least, say, 64 bytes), we may expect it to contain enough unknowable uniqueness or "entropy" (at least, say, 64 bits) to prevent a codebook attack. In that case, ECB mode has the advantage of supporting independent ciphering of each block. That, in turn, supports various things, like ciphering blocks in arbitrary order, or the use of multiple ciphering hardware operating in parallel for higher speeds.

Modern packet-switching network technologies often deliver raw packets out of order. The packets will be re-ordered eventually, but having out-of-sequence packets can be a problem for low-level ciphering if the blocks are not ciphered independently.

Also see Balanced Block Mixing, All or Nothing Transform and the "Random Access to Encrypted Data" conversation (locally, or @: http://www.ciphersbyritter.com/NEWS4/ECBMODE.HTM).

EMI

Electromagnetic Interference.

Encipher

The process which will transform information or plaintext into one of plethora of intermediate forms or ciphertext, as selected by a key. The inverse of decipher.

Encryption

The general term for hiding information in secret code or cipher.

Enemy

See opponent.

Engineering

1. The modeling activity associated with designing new constructions or devices to behave as predicted. Often directed toward achieving a known goal at minimal cost or risk. Also the use of a specialized body of knowledge to analyze and predict the behavior of a complex system.
2. The product definition and specification activities associated with new design and construction.
3. The testing activity of assuring that materials and constructions meet specifications.

4. The inventive activity of creating novel and unexpected designs, sometimes to the point of being beyond the understanding of conventional wisdom. Also the resulting accumulation of a body of knowledge; see patents.
5. The experimental activity of measuring designs, improving on those results and creating better, more optimal designs than those currently known, even with comprehensive theory. Sometimes the use of new devices in old systems.
6. The heuristic activity of building the best designs and constructions possible in absence of comprehensive theory, often by trial-and-error; see Software Engineering.
7. The planning and control activity associated with the coordination of multiple designers and constructors and their needed equipment and resources, potentially expensive and long-lead-time construction materials, across time, in the implementation of large complex systems; see Software Engineering, system design and risk management.

Ensemble Average

In statistics, the average of symbols at a particular position in a sequence, across all sequences. The probability of each symbol at a given place in a sequence. Also see ergodic process.

Entropy

A technical term with at least three distinct and confusing meanings in cryptography:

1. **The Shannon measure of coding efficiency**. Given the non-zero probability (*p*) for each symbol or value (*i*) in random variable *X*, we can calculate the real value *average information rate* (*H*).

   ```
   H(X) = -SUM( $p_i$ $\log_2$ $p_i$ )
   ```

   H is in bits per symbol when the log is taken to base 2. Also called "communications entropy."

2. **The idea of disorder from physics**: Eventually, everything breaks into disorder. Eventually, the universe will die the "heat death" of evenly-distributed energy. While not completely incompatible with Shannon entropy, "disorder" can mean whatever is convenient since it has no specific measure. In contrast, Shannon entropy deals with information and bits, not matter.

3. **Mystical unpredictability**. The Shannon entropy computation **can *measure*** the information rate, but it **cannot *distinguish*** between *predictable* and *unpredictable* information. Using "entropy" to imply "unpredictable" is simply inconsistent with the Shannon information theory computation.

Note that all meanings can be casually described as our uncertainty as to the value of a random variable. But in cryptography, mere statistical uncertainty is **not** the same as the unpredictability required for cryptographic strength. (Also see Old Wives' Tale.)

Classical Introductions

In the original literature, and even thereafter, we do not find what I would accept as a precise word-definition of information-theoretic entropy. Instead, we find the development of a specific numerical computation which Shannon names "entropy."

Apparently the term "entropy" was taken from the physics because the *form* of the computation in information theory was seen to be similar to the *form* of "entropy" computations used in physics. The "entropy" part of this is thus the *formal* similarity of the computation, instead of a common underlying idea, as is often supposed.

The meaning of Shannon entropy is both implicit in and limited by the specific computation. Fortunately for us, the computation is relatively simple (as these things go), and it does not take a lot of secondary, "expert" interpretation to describe what it does or means. We can take a few simple, extreme distributions and easily calculate entropy values to give us a feel for how the measure works.

Basically, Shannon entropy is a measure of coding efficiency in terms of information bits per communicated bit. It gives us a measure of optimal coding, and the advantage is that we can quantify how much we would gain or lose with a different coding. But no part of the computation addresses the context required for "uncertainty" about what we could or could not predict. Exactly the same values occur, giving the same entropy result, whether we can predict a sequence or not.

> "Suppose we have a set of possible events whose probabilities of occurrence are $p_1, p_2, ..., p_n$. These probabilities are known but that is all we know concerning which event will occur. Can we find a measure of how much 'choice' is involved in the selection of the event or of how uncertain we are of the outcome?" -- Shannon, C. E. 1948. A Mathematical Theory of Communication. *Bell System Technical Journal.* 27:379-423.
>
> "In a previous paper the entropy and redundancy of a language have been defined. The entropy is a statistical parameter which measures, in a certain sense, how much information is produced on the average for each letter of a text in the language. If the language is translated into binary digits (0 or 1) in the most efficient way, the entropy *H* is the average number of binary digits required per letter of the original language." -- Shannon, C. E. 1951. Prediction and Entropy of Printed English. *Bell System Technical Journal.* 30:50-64.
>
> "[Even if we tried] all forms of encoding we could think of, we would still not be sure we had found the best form of encoding, for the best form might be one which had not occurred to us." "Is there not, in principle at least, some statistical measurement we can make on the messages produced by the source, a measure which will tell us the minimum average number of binary digits per symbol which will serve to encode the messages produced by the source?" -- Pierce, J. R. 1961. *Symbols, Signals and Noise.* Harper & Row.
>
> "If we want to understand this entropy of communication theory, it is best first to clear our minds of any ideas associated with the entropy of physics." ". . . the literature indicates that some workers have never recovered from the confusion engendered by an early admixture of ideas concerning the entropies of physics and communication theory." -- Pierce, J. R. 1961. *Symbols, Signals and Noise.* Harper & Row.

Entropy and Predictability

When we have a sequence of values from a random variable, that sequence may or may not be predictable. Unfortunately, there are virtually endless ways to predict a sequence from past values, and since we cannot test them all, we generally cannot know if a sequence is predictable or not (see randomness testing). So until we can predict a sequence, we are "uncertain" about each new value, and from that point of view, we might think the "uncertainty" of the sequence is high. That is how all RNG sequences look at first. It is not until we actually can predict those values that we think the "uncertainty" is low, again from our individual point of view. That is what happens when the inner state of a statistical RNG is revealed. So if we expect to interpret entropy by what we can predict, the result necessarily must be both contextual and dynamic. Can we seriously expect a simple, fixed computation to automatically reflect our own changing knowledge?

In practice, the entropy computations use actual sequence values, and will produce the same result whether we can predict those values or not. The entropy computation uses simple frequency-counts reflected as probabilities, and that is all. No part of the computation is left over for values that have anything to do with prediction or human uncertainty. Entropy simply does not discriminate between information on the basis of whether we can predict it or not. Entropy does not measure how unpredictable the information is. In reality, entropy is a mere statistical measure of information rate or coding efficiency. Like other statistical measures, entropy simply ignores the puzzles of the war between cryptographers and their opponents the cryptanalysts.

Application

The entropy(1) computation can be seen as one of many possible measures of randomness, with the ideal being a flat or uniform distribution of values. (Obviously, anything other than a flat distribution will be, to some extent, predictable.) Also see: linear complexity and Kolmogorov-Chaitin complexity.

Entropy(1) is useful in coding theory and data compression, but requires a knowledge of the probabilities of each value which we usually know by sampling. Consequently, in practice, we may not *really* know the "true" probabilities, and the probabilities may change through time. Furthermore, calculated entropy(1) does not detect any underlying order that might exist between value probabilities, such as a correlation, or a linear relationship, or any other aspect of cryptographically-weak randomness.

The limit to the unpredictability in any deterministic random number generator is just the number of bits in the state of that generator, plus knowledge of the particular generator design. Nevertheless, most RNG's will have a very high calculated entropy(1) in bits-per-symbol, because each symbol or value is produced with about the same probability, despite the sequence being ultimately predictable. Accordingly, a high entropy(1) value does **not** imply that a source really *is* random, or that it produces unknowable randomness at any rate, or indeed have any relationship at all to the amount of unknowable cryptographic randomness present, if any.

On the other hand, it is very possible to consider pairs of symbols, or triples, etc. The main problem is practicality, because then we need to collect exponentially more data, which can be effectively impossible. Nevertheless, if we have a trivial toy RNG such as a LFSR with a long single cycle, and which outputs its entire state on each step, it may be possible to detect problems using entropy(1). Although a random generator should produce any possible next value from any state, we will find that each state leads into the next without choice or variation. But we do not need entropy(1) to tell us this, because we know it from the design. On the other hand, cryptographic RNG's with substantial internal state which output only a small subset of their state are far too large to measure with entropy(1). And why would we even want to, when we already know the design, and thus the amount of internal state, and thus the maximum entropy(3) they can have?

A value similar to entropy(1) can be calculated by population estimation methods (see augmented repetitions). Also see the experimental correspondence in entropy values from noise generator characterization tests: "Experimental Characterization of Recorded Noise" (locally, or @: http://www.ciphersbyritter.com/NOISE/NOISCHAR.HTM).

Some fairly new and probably useful formulations have been given which include the term "entropy," and so at first seem to be other *kinds* of entropy. However, the name "entropy" came from a *formal* similarity to the computation in physics. To the extent that new computations are *less* similar to the physics, they confuse by including the term "entropy."

Equation

A mathematical statement that two expressions have the same value, for any possible values which replace the variables in the expressions. The statement may be true or false. Also see: root.

Equivocation

1. A logic fallacy.
2. A measure of uncertainty.

> "[The] set of *a posteriori* probabilities describes how the cryptanalyst's knowledge of the message and key gradually becomes more precise as enciphered material is obtained. This description,

however, is much too involved and difficult for our purposes. What is desired is a simplified description of that approach to uniqueness of the possible solutions."

". . . a natural mathematical measure of this uncertainty is the conditional entropy of the transmitted signal when the received signal is known. This conditional entropy was called, for convenience, the equivocation."

". . . it is natural to use the equivocation as a theoretical security index. It may be noted that there are two significant equivocations, that of the key and that of the message. These will be denoted by $H_E(K)$ and $H_E(M)$ respectively. They are given by:

$$H_E(K) = \mathrm{Sum}_{E,K}[\ P(E,K) \log P_E(K)\ ]$$
$$H_E(M) = \mathrm{Sum}_{E,M}[\ P(E,M) \log P_E(K)\ ]$$

in which *E*, *M* and *K* are the cryptogram, message and key and

- $P(E,K)$ is the probability of key *K* and cryptogram *E*
- $P_E(K)$ is the *a posteriori* probability of key *K* if cryptogram *E* is intercepted
- $P(E,M)$ and $P_E(M)$ are the similar probabilities for message instead of key.

"The summation in $H_E(K)$ is over all possible cryptograms of a certain length (say *N* letters) and over all keys. For $H_E(M)$ the summation is over all messages and cryptograms of length *N*. Thus $H_E(K)$ and $H_E(M)$ are both functions of *N*, the number of intercepted letters."

-- Shannon, C. E. 1949. Communication Theory of Secrecy Systems. *Bell System Technical Journal.* 28: 656-715.

Also see: unicity distance, Perfect Secrecy, Ideal Secrecy and pure cipher.

Ergodic

See ergodic process.

Ergodic Process

In statistics and information theory, a type of Markov process. A particularly "simple" and easily modeled stationary (homogenous) stochastic (random) process (function) in which the temporal average is the same as the ensemble average. In general, a process in which no state is prevented from re-occurring. Ergodic processes are the basis for many important results in information theory, and are thus a technical requirement before those results can be applied.

Here we have all three possible sequences from a **non**-ergodic process: **across** we have the average of symbols through time (the "temporal average"), and **down** we have the average of symbols in a particular position over all possible sequences (the "ensemble average"):

```
A B A B A B ...   p(A) = 0.5, p(B) = 0.5, p(E) = 0.0
B A B A B A ...   p(A) = 0.5, p(B) = 0.5, p(E) = 0.0
E E E E E E ...   p(A) = 0.0, p(B) = 0.0, p(E) = 1.0
^         ^
|   ...   |
|         +----   p(A) = 0.3, p(B) = 0.3, p(E) = 0.3
|                 ...
+--------------   p(A) = 0.3, p(B) = 0.3, p(E) = 0.3
```

(Pierce, J. 1961. *Symbols, Signals and Noise.*)

When a process is ergodic, every possible ensemble average is equal to the time average. As increasingly long sequences are examined, we get increasingly accurate probability estimates. But when a process is non-ergodic, the measurements we take over time from one or a few sequences may not represent all possible sequences. And measuring longer sequences may not help. Also see entropy.

Error Correcting Code

A data coding which adds redundant information to the data representation, so that if data are changed in storage or transit, some amount of damage can be corrected. Also see: error detecting code and block code.

Error Detecting Code

A data coding which adds redundant information to the data representation, so that if data are changed in storage or transit, the damage can be detected with high probability, but not certainty. Often a hash like CRC. Other examples include: checksum and parity. Also see: error correcting code and block code.

ESD

Electrostatic Discharge.

Even Distribution

Uniform distribution.

Evidence

1. Information used in reasoning toward proof.
2. Information that tells us what happened. As opposed to risk, which addresses what *might* happen in the future.

Although most hard sciences can depend upon experimental measurement to answer basic questions, cryptography is different. Often, issues must be argued on lesser evidence, but science rarely addresses kinds of evidence, or how conclusions might be drawn.

Exclusive-OR

A Boolean logic function which is also mod 2 addition. Also called XOR. Generalized beyond bits to arbitrary values in the Latin square combiner.

Expectation

In statistics, after sampling a random variable many times, the average value over all samples. Consequently, the value we "expect" to see.

Exposure

In risk analysis, the statistical expectation of loss from a particular hazard. The product of event probability and the loss due to that event.

Expression

A sequence of symbols denoting mathematical operations, variables and constants. Also see: factor, term and equation.

Extraordinary Claims

A quote attributed to Carl Sagan is that: "extraordinary claims require extraordinary evidence." Apparently this comment repeatedly occurred in a long-running battle between the famous astronomer and UFO believers. Also see claim and burden of proof.

It is easy to sympathize with the quote, but it is also easily misused: For example, what, exactly, is being "claimed"? Is that stated, or do we have argument by innuendo? And just who determines what is "extraordinary"? And what sort of evidence could possibly be sufficiently "extraordinary" to convince someone who has a contrary bias?

In a scientific discussion or argument, a distinction must be made between what is actually *known* as fact and what has been merely *assumed* and *accepted* for lo, these many years. Often, what is needed is not so much evidence, as the reasoning to expose conclusions which have drawn far beyond the evidence we have. Even if well-known "experts" have chosen to believe overdrawn conclusions, that does not make those conclusions correct, and also does not require new evidence, let alone anything "extraordinary."

Extractor

In a cryptographic context, an extractor is a mechanism which produces the inverse effect of a combiner. This allows data to be enciphered in a combiner, and then deciphered in an extractor. Sometimes an extractor is exactly the same as the combiner, as is the case for exclusive-OR.

F_q

In math notation, a finite field of order q.

F_q*

In math notation, the multiplicative group of the finite field of order q.

(F_q,+)

In math notation, the additive group of the finite field of order q.

Factor

1. A part of a mathematical expression which is multiplied against other parts. As opposed to term.
2. The attempt to find the primes which make up the *factors* of a composite or non-prime value.

Factorial

The *factorial* of natural number *n,* written **n!**, is the product of all integers from 1 to *n.* See factor.

See the factorials section of the "Base Conversion, Logs, Powers, Factorials, Permutations and Combinations in JavaScript" page (locally, or @: http://www.ciphersbyritter.com/JAVASCRP/PERMCOMB.HTM#Factorials).

Failure

In a system, when a required operation is not delivered to specification, due to some fault.

Failure Modes and Effects Analysis

(FMEA). An analytical process used in fault analysis or risk analysis to organize the examination of component or subprocess failure and consequential results.

The system under analysis is considered to be a set of components or black box elements, each of which may fail. By considering each component in turn, the consequences of a failure in each particular component can be extrapolated, and the resulting costs or dangers listed. For each failure it is generally possible to consider alternatives to minimize either the probability or the effect of such failure. Things that might be done include:

- improve component quality so that the usual failure modes no longer apply, or at least occur less frequently;
- use multiple *redundant* components organized such that any single component failure does not produce a system failure; and
- simplify the design so that fewer unreliable components are used.

Also see risk, risk management and fault tree analysis.

Fallacy

In the philosophical study of logic, sometimes apparently-reasonable arguments that do not maintain truth, and often lead to false conclusions. A fallacious argument does not mean that a conclusion is wrong, only that the conclusion is not supported by that argument. Although most authors agree on the classical fallacies, there is no single name for various modern fallacies, and different authors give different organizations. There can be no exhaustive list. Also see: inductive reasoning, deductive reasoning, proof, rhetoric, and propaganda. Common fallacies include:

A. **Fallacies of Insufficient Evidence**
    A. **Mere claims** with no supporting evidence at all.
    B. **Claims** without acceptable supporting evidence.
    C. ***ad ignorantium*** ("Appeal to Ignorance") -- a belief which is claimed to be true because it has not been proven false.
    D. **Hasty Generalization**
    E. **Unrepresentative Sample**
    F. **Card Stacking** -- a deliberate withholding of evidence which does not support the author's conclusions.

B. **Fallacies of False Cause (*non causa pro causa*)**
    A. ***post hoc ergo propter hoc*** ("after this therefore because of this")
    B. ***reductio ad absurdum*** -- a form of argument which assumes a premise is false then shows that leads to a contradiction. The fallacy is in assuming that a particular one of multiple premises is necessarily false.
    C. **Joint Effect** -- the assumption of a single cause, when there are multiple causes
    D. **Composition** -- the implication that what is true of the parts must also be true of the whole.
    E. **Division** -- the implication that what is true of the whole must be true of its parts.

C. **Fallacies of Irrelevance (*ignoratio elenchi*)** -- ignoring the question
    A. ***ad hominem*** ("Name Calling").
    B. ***ad populum*** ("Plain Folks") -- an appeal to the prejudices and biases of the audience.
    C. ***ad misericordiam*** ("Appeal to Pity")
    D. ***ad nauseam*** ("Repetition") -- repetition to the point of disgust.
    E. ***ad verecundiam*** ("Appeal to Awe") -- a flawed or unrepresentative testimonial from a supposed authority. The source may not be an authority, or may be biased, or mistaken, or quoted out of context. Different authorities may have different opinions. The "awe" part of this may have us accept a mere conclusion, perhaps based on fuzzy or even false reasoning, when we should examine the facts and the argument and come to a conclusion on our own (especially see belief).
        - ("**Inappropriate Authority**") -- a testimonial from someone with expertise in a different field.
    F. ***tu quoque*** ("You Did It Too").
    G. ***ad baculum*** ("Appeal to force") -- e.g., threats.
    H. **Red Herring** -- a new irrelevant topic, sometimes deliberately intended to throw the discussion off track (supposedly named for the practice of using fish to throw tracking dogs off the scent).
        - **Dictionary** -- my term for ignoring the original topic, and instead arguing the words used to describe it, to the exclusion of the underlying idea.
    I. **Opposition** ("Guilt by Association") -- to condemn an idea because of who is for it.
    J. **Genetic** -- attacking the source of the idea, rather than the idea itself.
    K. **Bandwagon** -- everybody agrees, so it must be true
    L. **Straw Man** -- an extreme, exaggerated or just different and weaker argument introduced to imply that the original argument was absurd, faulty or weak.
    M. **Appeal to Tradition** -- since it is one of our ancient beliefs, we all know it is true.

D. **Fallacies of Ambiguity**
    A. **Equivocation** -- the use of a word in different senses (different meanings) in the same argument.

        B. **Amphiboly** -- sentences whose gramatical structure admit more than one interpretation.
        C. **Accent** -- some sentences have different meanings depending on which word is stressed.
        D. **False Analogy** -- the two things being compared are in fact different in a way which affects the supposedly shared property
        E. **Quoting Out Of Context** -- language often requires the surrounding context to distinguish the particular meaning the author intended.
        F. **Argument By Innuendo** -- directing the reader to an unwarranted and usually false conclusion without stating the conclusion directly so it can be challenged, or hiding the argument itself behind a mere claim, so the argument cannot be challenged.

    E. **Fallacies of the Misuse of Logic**
        A. ***petitio principii*** ("Begging the Question") -- assuming the truth of a proposition which needs to be proven. (Here "beg" means "improperly take for granted.")
            - ***circulus in probando*** ("Circular Argument") -- a conclusion which just restates a premise.
        B. ***non sequitur*** ("Does Not Follow") -- the stated conclusion does not follow from the evidence supplied.
        C. ***plurimum interrogationum*** ("Multiple Question" or "Complex Question") -- e.g., "When did you stop beating your wife?"
        D. **Garbled Syllogism** -- an illogical argument phrased in logical terms.
        E. **Either-Or** -- assuming a question has only two sides.
        F. **Accident** ("Dicto Simpliciter") -- demanding mathematical correctness from a rule of thumb. The "accident" is the occurrence of a low-probability random value which does not properly reflect the whole. The fallacy is any attempt to use a non-representative value to contradict a generally correct result.
        G. **Special Pleading** ("Double Standard") -- claiming an exception to a rule-of-thumb based on irrelevant issues

Fast Walsh Transform

(FWT). (Also Walsh-Hadamard transform, WHT.) When applied to a Boolean function, a fast Walsh transform is essentially a correlation count between the given function and each Walsh function. Since the Walsh functions are essentially the affine Boolean functions, the FWT computes the unexpected distance from a given function to each affine function. It does this in time proportional to *n* log n, for functions of *n* bits, with *n* some power of 2. Also see Boolean function nonlinearity.

If two Boolean functions are *not* correlated, we expect them to agree half the time, which we might call the "expected distance." When two Boolean functions *are* correlated, they will have a distance greater or less than the expected distance, and we might call this difference the unexpected distance or UD. The UD can be positive or negative, representing distance to a particular affine function or its complement.

It is easy to do a fast Walsh transform by hand. (Well, I say "easy," then always struggle when I actually do it.) Let's do the FWT of function *f* : (1 0 0 1 1 1 0 0). First note that *f* has a binary power length, as required. Next, each pair of elements is modified by an "in-place butterfly"; that is, the values in each pair produce two results which replace the original pair, wherever they were originally located. The left result will be the two values added; the right will be the first less the second. That is,

```
   (a',b') = (a+b, a-b)
```

So for the values (1,0), we get (1+0, 1-0) which is just (1,1). We start out pairing adjacent elements, then every other element, then every 4th element, and so on until the correct pairing is impossible, as shown:

```
   original      1    0    0    1    1    1    0    0
                 ^---^    ^---^    ^---^    ^---^

      first      1    1    1   -1    2    0    0    0
                 ^--------^         ^--------^
```

```
                         ^-------^        ^-------^

      second       2   0   0   2   2   0   2   0
                   ^---------------^
                       ^---------------^
                           ^---------------^
                               ^---------------^


       final       4   0   2   2   0   0  -2   2
```

The result is the unexpected distance to each affine Boolean function. The higher the absolute value, the greater the "linearity"; if we want the *non*linearity, we must subtract the absolute value of each unexpected distance from the expected value, which is half the number of bits in the function. Note that the range of possible values increases by a factor of 2 (in both positive and negative directions) in each sublayer mixing; this is information expansion, which we often try to avoid in cryptography.

Also see: "Walsh-Hadamard Transforms: A Literature Survey," locally, or @: http://www.ciphersbyritter.com/RES/WALHAD.HTM. and the "Active Boolean Function Nonlinearity Measurement in JavaScript" page, locally, or @: http://www.ciphersbyritter.com/JAVASCRP/NONLMEAS.HTM.

The FWT provides a strong mathematical basis for block cipher mixing such that all input values will have an equal chance to affect all output values. Cryptographic mixing then occurs in butterfly operations based on balanced block mixing structures which replace the simple add / subtract butterfly in the FWT and confine the value ranges so information expansion does not occur. A related concept is the well-known FFT, which can use exactly the same mixing patterns as the FWT.

Fault

In a system, the cause of an operation failure. However, the exact same fault may not cause a failure at other times. Faults can occur from:

- specification,
- design,
- implementation,
- documentation,
- operation,
- environment, and/or
- use.

Faults can be

- *excluded* by design,
- *removed* during implementation by verification and testing, and
- *avoided* or *tolerated* during operation.

Fault Tolerance

The goal of making a system operate within specification (without failure) in the presence of faults. Measured by coverage.

Fault tolerance is achieved by eliminating each and every single point of failure in the system. Improving the reliability can benefit the overall system, but is not the same as fault tolerance.

Fault Tree Analysis

(FTA). An analytical process used in fault analysis or risk analysis to organize the examination of particular undesirable outcomes. (See tree analysis.)

First, the various undesired outcomes are identified. Then each sequence of events which could cause such an outcome is also identified.

If it is possible to associate an independent probability with each event, it may be possible to compute an overall probability of occurrence of the undesirable outcome. Then one can identify the events which make the most significant contributions to the overall result. By minimizing the probability of those events or reducing their effect, the overall probability of the negative outcome may be reduced.

Various system aspects can be investigated, such as unreliability (including the effect of added redundancy), system failure (e.g., the causes of a plane crash), and customer dissatisfaction. A potential advantage is efficiency when multiple faults seem to converge in a particular node, since it may be possible to modify that one node to eliminate the effect of many faults at once. However, that also would be contrary to a "defense in depth" policy of protecting all levels, wherever a fault might occur or propagate.

Also see risk, risk management, failure modes and effects analysis and attack tree.

FCSR

Feedback with Carry Shift Register. A sequence generator analogous to a LFSR, but separately storing and using a "carry" value from the computation.

Feedback

Returning the output back to the input. In electronics, a major tool in circuit design. Used with analog amplifiers, *negative* feedback can improve linearity (reduce distortion). In contrast, *positive* feedback can increase amplification and is the fundamental basis for oscillation. Feedback is also used in digital systems such as LFSR's and most other RNG's. to create long but predictable sequences. In cryptography, feedback is used in autokey stream ciphers to continually add ciphertext complexity to the state of the confusion or running key RNG.

Feistel

Horst Feistel, a senior employee of IBM in the 60's and 70's, responsible for the Feistel construction used in the early Lucifer cipher and then DES itself. Awarded a number of important crypto patents, including: 3,768,359 and 3,768,360 and 4,316,055.

Feistel Construction

The Feistel construction is the widely-known method of mixing used in some block ciphers including the classic DES. One generalization is the Unbalanced Feistel Network, and one alternative is Balanced Block Mixing.

Normally, in a Feistel construction, the input block is split into two parts, one of which drives a transformation whose result is exclusive-OR combined into the other block. Then the "other block" value feeds the same transformation, whose result is exclusive-OR combined into the first block. This constitutes 2 of perhaps 16 "rounds."

```
L           R
|           |
|--> F --> +     round 1
|           |
+ <-- F <--|     round 2
|           |
v           v
L'          R'
```

One advantage of the Feistel construction is that the transformation does not need to be invertible. To reverse any particular layer, it is only necessary to apply the same transformation again, which will undo the changes of the original exclusive-OR.

A disadvantage of the Feistel construction is that diffusion depends upon the internal transformation. There is no guarantee of overall diffusion, and the number of rounds required is often found by experiment.

Fenced DES

A block cipher with three layers, in which the outer layers consist of fencing tables, and the inner layer consists of DES used as a component. For block widths over 64 bits, Balanced Block Mixing technology assures that any bit change is propagated to each DES operation.

Also see the Fenced DES section of the main page locally, or @: http://www.ciphersbyritter.com/index.html#FencedTech. Also see: "A Keyed Shuffling System for Block Cipher Cryptography," locally, or @: http://www.ciphersbyritter.com/KEYSHUF.HTM.

Fencing

Fencing is a term-of-art which describes a layer of substitution tables. In schematic or data-flow diagrams, the row of tiny substitution boxes stands like a picket fence between the data on each side.

Fencing Layer

A fencing layer is a variable size block cipher layer composed of small (and therefore realizable) substitutions. Typically the layer contains many separate keyed substitution tables. To make the layer extensible, either the substitutions can be re-used in some order, or in some pre-determined sequence, or the table to be used at each position selected by some computed value.

Fencing layers are also used in other types of cipher.

FFT

Fast Fourier Transform. A numerically advantageous way of computing a Fourier transform. Basically a way of transforming information between amplitude values sampled periodically through time, and amplitude values sampled across complex frequency. The FFT performs this transformation in time proportional to n log n, for some n a power of 2.

While exceedingly valuable, the FFT tends to run into practical problems in use which can require a deep understanding of the process:

- An FFT assumes that the waveform is stationary and thus repetitive and continuous, which is rarely the case.
- Sampling a continuous wave can create spurious "frequency" values related to the sampling and not the wave itself.
- The range of possible values increases by a factor of 2 (in both positive and negative directions) in every sublayer mixing; this is information expansion, which we often try to avoid in cryptography.
- An FFT transforms the sampled wave into specific precise frequency components, not frequency bands: in most signals other frequencies will dominate and may not be represented as expected in FFT results.

The FFT provides a strong mathematical basis for block cipher mixing in that all input values will have an equal chance to affect all output values. But an ordinary FFT expands the range of each sample by a factor of two for each mixing sub-layer, which does not produce a conventional block cipher. A good alternative is Balanced Block Mixing, which has the same general structure as an FFT, but uses balanced butterfly operations based on orthogonal Latin squares. These replace the simple add / subtract butterfly in the ordinary FFT, yet confine the value ranges so information expansion does not occur. Another concept related to the FFT is the fast Walsh-Hadamard transform (FWT), which can use exactly the same mixing patterns as the FFT.

Field

In abstract algebra, a commutative ring with more than one element, a unity (multiplicative identity) element, and a corresponding multiplicative inverse for every nonzero element. (This means we can divide without loss.) Also see: group.

In general, a field supports the four basic operations (addition, subtraction, multiplication and division), and satisfies the normal rules of arithmetic. An operation on any two elements in a field is a result which is also an element in the field. The real numbers and complex numbers are examples of infinite fields.

Fields of finite order include rings of integers modulo some prime. Here are multiplication tables under mod 2, mod 3 and mod 4:

```
       0  1           0  1  2          0  1  2  3

   0   0  0       0   0  0  0      0   0  0  0  0
   1   0  1       1   0  1  2      1   0  1  2  3
                  2   0  2  1      2   0  2  0  2
                                   3   0  3  2  1
```

In a field, each element must have an inverse, and the product of an element and its inverse is 1. This means that every non-zero row and column of the multiplication table for a field must contain a 1. Since row 2 of the mod 4 table does not contain a 1, the set of integers mod 4 is not a field. This is because 4 is not a prime.

The order of a field is the number of elements in that field. The integers mod prime *p* (Z/p) form a finite field of order *p*. Similarly, mod 2 polynomials will form a field with respect to an irreducible polynomial, and will have order 2^n, which is a very useful size.

FIFO

First In, First Out. A form of queue which accumulates values as they occur and holds them until they are consumed in the order they occurred. As opposed to LIFO, which is also known as a stack.

Filter

In electronics, typically a device or circuit intended to reduce some frequencies in preference to other frequencies. Filters can be low-pass, high-pass, or bandpass. Filtering also occurs in power supplies, where rectified AC is smoothed into DC output. Also see: voltage divider.

For example, suppose we have an active filter with a voltage gain of 1 at 425Hz and a gain of 6 at 2550Hz:

```
amplitude change in dB = 20 log10 (e2/e1) = 20 log10 6 = 15.56 dB
octaves = log2 (f2/f1) = log2 (2550/425) = 2.58 octaves
decades = log10 (f2/f1) = log10 (2550/425) = 0.778 decades
dB/octave = 15.56 / 2.58 = 6dB/octave
dB/decade = 15.56 / 0.778 = 20dB/decade
```

The value 6dB/octave or 20dB/decade is a clasic indication of single-stage RC filtering.

Finite Field

A Galois field, denoted GF(x): A mathematical field of non-infinite order. As opposed to an *infinite field,* like the reals and complex numbers.

- In a finite field, every nonzero element *x* can be squared, cubed, and so on, and at some power will eventually become 1. The smallest (positive) power *n* at which $x^n = 1$ is the order of element *x*. This of course makes *x* an "*nth root of unity,*" in that it satisfies the equation $x^n = 1$.
- A finite field of order *q* will have one or more *primitive* elements *a* whose order is *q*-1 and whose powers cover all nonzero field elements.
- For every element *x* in a finite field of order *q*, $x^q = x$.

Also denoted F_q for a finite field of order *q*. Also see: characteristic.

Finite State Machine

FSM. The general mathematical model of computation consisting of a finite amount of storage or state, transitions between state values, and output as some function of state. Given full knowledge of the "state machine" (that is, the next-state and output functions), plus the initial state values, the resulting sequence of states and outputs is defined absolutely. This is completely predictable deterministic computation.

In particular, a finite collection of states *S*, an input sequence with alphabet *A*, an output sequence with alphabet *B*, an output function *u(s,a)*, and a next state function *d(s,a)*.

Other than really random generators for nonce or message key values, all the computations of cryptography are finite state machines and so are completely deterministic. Much of cryptography thus rests on the widespread but unproven belief that the internal state of a cryptographic machine (itself a FSM) cannot be deduced from a substantial amount of known output, even when the machine design is completely defined.

Flat Distribution

Uniform distribution.

Flip-Flop

A class of digital logic component which has a single bit of state with various control signals to effect a state change. There are several common versions:

- **Latch**: the output follows the input, but only while the clock input is "1"; lowering the clock prevents the output from changing.
- **SR FF**: Set / Reset; typically created by cross-connecting two 2-input NAND gates, in which case the inputs are complemented: a "0" on the S input forces a stable "1" state, which is held until a "0" on the R input forces a "0".
- **D or "delay" FF**: senses the input value at the time of a particular clock transition.
- **JK FF**: the J input is an AND enable for a clocked or synchronous transition to "1"; the K input is an AND enable for a clocked transition to "0"; and often there are S and R inputs to force "1" or "0" (respectively) asynchronously.

Flow Control

The control aspect of a digital communications system which temporarily stops a data source from sending data. This allows the data sink to "catch up" or perform other operations without losing data.

Ultimately, flow control is one of the most important aspects of a communications network. Inside the network, however, protocols may simply send and re-send data until those particular data are acknowledged.

Formal Proof

In mathematics, a proof which does not depend in any way at all upon the meanings of the terms. Each unique term can be replaced by an arbitrary unique symbol without affecting the truth of the statement. The logic structure of a formal proof is thus susceptible to mechanical verification. As opposed to an informal proof.

Fourier Series

An infinite series in which the terms are constants (A, B) multiplied by sine or cosine functions of integer multiples (n) of the variable (x). One way to write this would be:

$$f(x) = A_0 + \mathrm{SUM}\ (A_n \cos nx + B_n \sin nx)$$

Alternately, over the interval [a, a+2c]:

f(x) = a_0 + SUM (a_n cos(n PI x/c) + b_n sin(n PI x/c))
a_n = 1/c INTEGRAL[a,a+2c](f(x) cos(n PI x/c) dx)
b_n = 1/c INTEGRAL[a,a+2c](f(x) sin(n PI x/c) dx)

Fourier Theorem

Under suitable conditions any periodic function can be represented by a Fourier series. (Various other "orthogonal functions" are now known.)

The use of sine and cosine functions is particularly interesting, since each term (or pair of terms) represents a single frequency oscillation with a particular amplitude and phase. So to the extent that we can represent an amplitude waveform as a series of sine and cosine functions, we thus describe the frequency spectrum associated with that waveform. This frequency spectrum describes the frequencies which must be handled by a circuit to reproduce the original waveform. This illuminating computation is called a Fourier transform.

Fourier Transform

The Fourier transform relates amplitude samples at periodic discrete times to amplitude samples at periodic discrete frequencies. There are thus two representations: the amplitude vs. time waveform, and the amplitude vs. complex frequency (magnitude and phase) spectrum. Exactly the same information is present in either representation, and the transform supports converting either one into the other. This computation is efficiently performed by the FFT.

In a cryptographic context, one of the interesting ideas of the Fourier transform is that it represents a thorough mixing of each input value to every output value in an efficient way. On the other hand, using the actual FFT itself is probably impractical for several reasons:

- **Range Expansion.** An FFT significantly expands the range of each input variable, and thus requires more storage (for exact results) than the original values.
- **Floating Point.** The use of sine and cosine functions virtually implies floating point results, which is not ideal for cryptography.

The basic idea of efficiently combining each value with every other value is generalized in cryptography as Balanced Block Mixing. BBM structures can be applied in FFT-like patterns, and can support a wide range of keyed, non-expanding, nonlinear, and yet reversible transformations.

Frequency

In general, the number of repetitions or *cycles* per second. Specifically, the number of repetitions of a sine wave signal per second: A signal of a single frequency is a sine wave of that frequency. Any deviation from the sine waveform can be seen as components of other frequencies, as described by an FFT. Now measured in Hertz (Hz); previously called cycles-per-second (cps).

Typically, audio frequencies range from 20Hz to 20kHz, although many designs try to be flat out to 100kHz. Video baseband frequencies range from DC up to something like 3MHz or 5 MHz. Common names for radio frequency (RF) ranges with generally similar properties are:

ELF	3Hz..300Hz	naval strategic commands
VF	300Hz..3kHz	telephone voice
VLF	3kHz..30kHz	radionavigation, military
LF	30kHz..300kHz	time/freq std (WWVB 60kHz)
MF	300kHz..3MHz	AM broadcast band
HF	3MHz..30MHz	shortwave, ham bands, CB
VHF	30MHz..300MHz	FM broadcast band, TV
UHF	300MHz..3GHz	TV, cell phones, satellite
SHF	3GHz..30GHz	satellite
EHF	30GHz..300GHz	

FSM

Finite State Machine.

Function

A rule which takes zero or more independent variables or input parameters or arguments (in a defined domain) and produces a single result or output (in a defined range). A mapping; sometimes specifically confined to numbers.

FWT

Fast Walsh Transform.

(G,*)

In math notation, a group under multiplication.

Gain

The amplitude change due to amplification. A negative gain is in fact a *loss.*

Galois Field

Finite field. First encountered by the 19-year-old student Evariste Galois, in 1830 France, a year or so before dying in a duel. Denoted GF(x).

Game Theory

A branch of mathematical analysis concerned with the worth of opposing strategies in the context of a competition or generalized "game."

Garble

To distort or make unintelligible, or the distorted or unintelligible result. In cryptography, deciphered plaintext which is unintelligible, in part or whole, typically due to use of a wrong key, transmission errors, or possible attack.

Gate

1.A digital logic component which implements a simple logic function, possibly with a complemented output. Some common Boolean logic gates include:

- AND
- OR
- Exclusive-OR
- NAND -- AND with output complement
- NOR -- OR with output complement
- Exclusive-NOR -- Exclusive-OR with output complement
- NOT -- the complement (an inverter)

2. That part of a field effect transistor (FET) which controls current flow.

Gaussian

The often-found, usual, or normal statistical distribution.

GCD

Greatest Common Divisor. In number theory, the largest factor common to two positive integers. For integers *x* and *y*, written as gcd(*x*,*y*) or just (*x*,*y*). Also see: congruence, relatively prime and math notation.

Geffe Combiner

A stream cipher combiner for mixing two RNG's.

Geffe, P. 1973. How to protect data with ciphers that are really hard to break. *Electronics*. January 4. 99-101.

For conventional stream ciphers, which are basically just an RNG and exclusive-OR, the name of the game is to find a strong RNG. Most RNG designs are essentially linear and easily broken with just a small amount of the produced sequence. The Geffe combiner was an attempt to combine two RNG's and produce a stronger sequence than each. But that was not to be.

Also see: "The Story of Combiner Correlation: A Literature Survey" (locally, or @: http://www.ciphersbyritter.com/RES/COMBCORR.HTM#Geffe73) and "MacLaren-Marsaglia Again" (locally, or @: http://www.ciphersbyritter.com/NEWS5/MACLAR.HTM).

Geiger-Mueller Tube

An electronic radioactivity detector which produces a strong voltage pulse when an ionizing-radiation event is detected. While a Geiger-Mueller tube cannot measure the strength of an event, it does makes some events obvious with little or no additional amplification or processing. The number of events per unit of time then represents a "count" of the ionizing radiation being encountered. Often made with a thin mica window which allows alpha particles to be sensed.

Typically a cylindrical tube with an outer conductive shell (the cathode), a wire (the anode) in the center, and filled with a gas like argon at low pressure. Depending on the tube involved, a positive bias of perhaps 500 or 600 volts above the cathode is applied to the anode through a resistance of perhaps 1 to 10 Megohms. We would expect both tube temperature and applied voltage to affect detection sensitivity to some extent.

When an ionizing event like a gamma ray interacts with an atom of the internal gas, a fast electron may be ejected from the shell of the atom. As the ejected electron encounters other atoms, it may cause other electrons to be ejected, producing a cascade or "avalanche" of gas ions along their paths. If the full distance between cathode and anode becomes ionized, a strong current pulse or arc will occur. Presumably, many weaker or wrongly positioned events occur which do not form an arc and are not sensed. However, even unsensed events may have short-term and localized effects on sensitivity that may be hard to quantify.

After the initial pulse (which discharges the interelectrode capacitance), current for the arc flows through the anode resistance, which should cause the applied voltage to drop below the level which would sustain an arc. After the arc ends, an internal trace gas, such as an alcohol, may help to "quench" the ionization which could cause the same arc to reoccur. During the avalanche and quench period, the tube cannot detect new events. Meanwhile, the anode voltage climbs back toward the operational level (charging the interelectrode capacitance) until another sufficiently-strong and properly-placed ionizing event occurs.

Often cited in cryptography as the ultimate really random generator.

Generator

In mathematics, one of the elements of a cyclic group which in some way produces all the other elements in the group.

GF(x)

A Galois field. See: math notation.

GF(2)

The Galois field of two elements: 0 and 1. See: math notation.

GF(2^n)

The Galois field or finite field of 2^n elements.

Typically we have mod 2 polynomials with results reduced "modulo" an irreducible or generator polynomial of degree *n*. This is analogous to creating a field from the integers modulo some prime *p*. Unfortunately, a block size of *n* bits would imply an order of 2^n, which is not prime. But we can get the block size we want using mod 2 polynomials.

For example, consider $GF(2^4)$ with the generator polynomial $x^4 + x + 1$, or 10011, which is a degree-4 irreducible. First we multiply two elements as usual:

```
          1 0 1 1
        * 1 1 0 0
        ---------
                0
              0
      1 0 1 1
    1 0 1 1
   --------------
    1 1 1 0 1 0 0
```

Then we "reduce" the result modulo the generator polynomial:

```
                        1 1 0
              ---------------
    1 0 0 1 1 ) 1 1 1 0 1 0 0
                1 0 0 1 1
                ---------
                  1 1 1 0 0
                  1 0 0 1 1
                  ---------
                    1 1 1 1 0
                    1 0 0 1 1
                    ---------
                        1 1 0 1
                       =========
```

So, if I did the arithmetic right, the result is the remainder, 1101. I refer to this as arithmetic "mod 2, mod p".

An irreducible is sufficient to form a finite field. However, some special irreducibles are also primitive, and these create "maximal length" sequences in LFSR's.

GF(2)[x]

The ring of polynomials in x whose elements are in the set GF(2). The mod 2 polynomials.

GF(2)[x]/p(x)

The field (with respect to addition, multiplication, and division) of polynomials in x modulo irreducible polynomial p(x) composed of the Galois field {0,1}.

Goodness of Fit

In statistics, a test used to compare two distributions. For nominal or "binned" measurements, a chi-square test is common. For ordinal or ordered measurements, a Kolmogorov-Smirnov test is appropriate.

Goodness-of-fit tests can *at best* tell us whether one distribution **is** or **is not** the same as the other, and they say even *that* only with some probability. It is important to be very careful about experiment design, so that, almost always, "nothing unusual found" is the goal we seek. When we can match distributions, we are obviously able to state exactly what the experimental distribution should be and is. But there are *many* ways in which distributions can differ, and simply finding a difference is *not* evidence of a specific effect. (See null hypothesis.)

Gray Code

An ordering of binary code values such that stepping through the sequence of values changes exactly one bit with each step. There are many such codings, for example:

Dec	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Greek Alphabet

Web page HTML does not have a specific representation for Greek symbols. However, if the user has a font which does have Greek symbols, it is easy to tell the browser to use that font. The obvious choice is the "Symbol" font, since it is normally present and widely available. As usual, an ASCII character will select a shape to display, but now the shape set is taken from the "Symbol" font. For example:

```
<FONT FACE = "Symbol">A</FONT> displays Α and
<FONT FACE = "Symbol">a</FONT> displays α;
<FONT FACE = "Symbol">G</FONT> displays Γ and
<FONT FACE = "Symbol">g</FONT> displays γ.
```

Alpha	A	Α	a	α
Beta	B	Β	b	β
Gamma	G	Γ	g	γ
Delta	D	Δ	d	δ
Epsilon	E	Ε	e	ε
Zeta	Z	Ζ	z	ζ
Eta	H	Η	h	η
Theta	Q	Θ	q	θ
Iota	I	Ι	i	ι
Kappa	K	Κ	k	κ
Lamda	L	Λ	l	λ
Mu	M	Μ	m	μ
Nu	N	Ν	n	ν
Xi	X	Ξ	x	ξ
Omicron	O	Ο	o	ο
Pi	P	Π	p	π
Rho	R	Ρ	r	ρ
Sigma	S	Σ	s	σ
Tau	T	Τ	t	τ
Upsilon	U	Υ	u	υ
Phi	J	ϑ	j	φ
Chi	C	Χ	c	χ
Psi	Y	Ψ	y	ψ
Omega	W	Ω	w	ω

Ground

In electronics, the concept of a common reference, against which voltage can be detected or measured. Also known as "earth."

1. A chassis or circuit common point.
2. A metallic connection buried in dirt.

Perhaps the earliest common use of a ground was in the electric telegraph, which came into use around the time of the U.S. Civil War. A battery voltage was switched onto a common wire using a telegraph key, and electromagnets up and down the wire responded by making a *click*. When the key was released, the electromagnets would release and make a *clack*, the time between click and clack being the dot or dash of Morse code. But for current to flow in the circuit, there had to be a return path. One way to do that was to string two wires for each circuit. However, it was found that a metal surface in the earth, such as a rod driven into the ground, can contact, within a few ohms of resistance, the same reference as used by everybody else. So, especially for small signals, the return path can be through the actual dirt itself, thus saving a lot of copper and making a system economically more viable.

The original concept of radio was to launch and collect signals from the air, as referenced to the common ground. What actually happens is the propagation of an electromagnetic wave, which can be detected without a common reference. But a ground can play an important role in an antenna system, especially at lower RF frequencies.

In the past, the usual ground reference was the copper cold water pipe which extended in the earth from the home to the city water main. In many homes, this reference was carried throughout the home on substantial copper pipe with soldered connections. Unfortunately, the introduction of nonconductive plastic water pipe, while convenient and cheap, also has eliminated an easy ground reference.

Power distribution, with a massive appetite for copper, is a natural application for one-wire connections, but in this case there are surprising and dangerous complexities. Nowadays, at the AC socket, we have both a protective ground wire which connects directly to some ground, and also a return power path, which is connected to ground at some point.

Ideally, the metal chassis or case of anything connected to the power lines should connect to the protective ground. Ideally, if protected equipment shorts out and connects live power to the case, that will blow the equipment fuse or even a power box circuit breaker, instead of electrocuting the operator. Even more ideally, a ground fault interrupter (GFI) can detect even a small amount of protective current flow and open an internal breaker. However, the protective ground system itself is generally tested at most once (upon installation), and if it goes bad under load, we will not know until bad things happen. While GFI's do have a "test" button, most ordinary equipment does not.

As different amounts of AC current flow about the home or building, wire resistance causes the voltage between the two AC socket grounds to vary, which is the origin of a ground loop. But ground loops are not limited to power circuits, and can present serious problems in instrumentation and audio systems.

Ground Loop

The condition where a voltage exists between different ground points in the same signal system, all of which points are supposed to be exactly the same. There are various causes:

- Ground loop hum is sometimes caused or *induced* by transformer-like coupling between AC power wires and the associated safety ground wire. This source of hum will vary dynamically according to the current flowing in those particular AC wires.
- Ground loop hum also can be caused by AC current flow in the safety ground wire, current which ideally should not be present. Any current flowing in the safety ground always develops a small voltage difference across the resistance of that wire. The source could be some amount of leakage from equipment, perhaps capacitive coupling inside a transformer, or any of a wide range of component failure possibilities.
- Ground loop signal coupling also can occur in the case of unbalanced signal interconnections. The usual culprit is single-conductor shielded wire using "RCA connectors." If a substantial signal is transported

some distance by unbalanced cable into a relatively low resistance load, there will be some signal current flow on the return shield. However, if the RCA connectors connect to chassis ground, and both pieces of equipment are safety grounded (as required), and plugged into different outlets, there will also be some return signal current flow in the safety ground. The result will be a small signal voltage between the outlet grounds, which may then impose itself on other unbalanced interconnections.

The simple ground model would have us believe that there is no resistance in the ground, which is of course false. Even sending a small signal from one amplifier to another on an unbalanced line implies that some current will flow on that line, and that same current also flows through the ground connections (often, a "shield" conductor). Thus, the voltage across different parts of ground will vary dynamically depending on the ground resistance, which can cause a cross-coupling between independent unbalanced channels. Even if that coupling is tiny, when working with tiny signals, it may matter anyway, especially in a TEMPEST context, or when working with signals in a receiver.

The ground loop problem is inherent in unbalanced signal lines. The same effects occur inside circuitry, but then the problem is under the control of a single designer or manufacturer; most problems occur when interconnecting different units. In general, ground loop cross-coupling effects are minimized by reducing ground resistance and increasing input or load resistance. Alternately, broadcast audio systems use balanced line interconnections that do not need ground as a part of the signal path. Balanced lines also tend to "cancel out" common-mode noise picked up by cables on long runs.

Consumer equipment generally uses unbalanced lines where the signal is referenced to some ground, typically on "RCA connectors." But because of ground loops, different equipment can have different references, thus introducing power line hum into the signal path.

Various responses are possible, but the one which is *not* possible is to open or disconnect the safety ground. The safety ground is there to protect life and should never be subverted. 3-prong to 2-prong AC adapters should never be used when equipment has 3-wire plugs. Because sound systems are interconnected, a system isolated from safety ground allows a failure on even one remote piece of equipment to electrify the entire system, and that is breathtakingly dangerous. Better alternatives always exist.

- Probably the easiest approach is to make sure that all equipment has grounded power plugs, and that those are inserted into a single power strip, instead of different wall outlets. 3-wire extensions should be used if necessary. This will tend to keep all pieces at the same ground level, a relative zero, even if that varies in an absolute sense. This is a "star" grounding system.
- Maybe the next thing is to directly connect each equipment signal ground to each other signal ground, using large copper wire. Large wire is not used to handle large currents, but instead to reduce interconnection resistance, and thus minimize the voltage effect of any leakage current. The "single power strip" approach should have done that already, but some equipment may not use chassis ground as a signal reference.
- The most direct solution to audio ground loops with existing equipment is to add external audio isolation transformers in each signal line between equipment. However, audio isolation transformers can be expensive, two are required for stereo, and they generally have worse specifications than the equipment which they interconnect. They also may require known load impedances, which typically are not otherwise required by consumer equipment, and so may need a resistor across the secondary (the side connected to the input of the next equipment). Transformers need no power and last essentially forever.
- Instead of audio isolation transformers, it is possible to buy or build active op amp unbalanced-to-balanced and balanced-to-unbalanced converters. These minimize common-mode noise from cable pickup (although less well, and perhaps *much* less well than a transformer), but do not really isolate ground. The advantage is in not requiring a shield ground, or in allowing the shield path to be open, while ignoring ground noise. Op amp balanced line converters can have an excellent frequency response, but, unlike transformers, must be powered.
- Another possibility is to use *power* isolation transformers, which are large and also expensive, but which do not affect signal quality. Sometimes a center-tap on the secondary can be grounded to the following

equipment, thus producing balanced AC power that can be helpful in reducing hum. If the problem is AC leakage into safety ground from some piece of equipment, isolating the power can prevent current flow into the safety ground.

- A possible fix for some problems is to insert a small (say, 10 ohm to 100 ohm) resistor in series with the audio shield at one end of each cable. If the problem was excessive current flow in the shield, the added resistor will prevent that, and the small resistance should have little effect on the signal going to a high-impedance input. But the resistor can worsen electromagnetic shielding and so cause pickup and interference on those lines (see shielding).
- It is also possible to open the shield connections entirely, thus depending on the AC safety ground for signal coupling. Using the AC ground as a signal reference is normally a bad idea, but this response has no cost, is relatively easy to try, and may improve things occasionally, although perhaps temporarily. Opening the shield is common when working with balanced lines that inherently ignore signals on the AC ground.
- When the original source of the problem is the CATV cable, two back-to-back 75 ohm to 300 ohm baluns may isolate the shield ground. But some cheap "transformers" have no transformer inside and so do not isolate. The user should use an ohmmeter to verify that the input shield does not connect to the output shield.

The pervasive nature of ground loops is a good reason to use isolated balanced lines. It is also a reason to use optical digital interconnections, which inherently isolate the ground references in different pieces of equipment.

Group

In abstract algebra, a nonempty set *G*, and a closed dyadic (two-input, one-output) operation with associativity, an identity element and inverse elements. Whatever the operation is, we may choose to call it "multiplication" and denote it with * as usual, the group denoted (G,*). Closure means that if elements (not necessarily numbers) *a, b* are in *G*, then *ab* (that is, *a*b*) is also in *G*. Also see: cyclic group.

In a group consisting of set G and closed operation * :

1. **The operation * is associative:** (ab)c = a(bc)
2. **There is a single identity element e which works with all elements:** for e and any a in G, ea = ae = a
3. **There a corresponding inverse for every element:** for any a in G, there is an a^{-1} in G such that $a^{-1}a = e = aa^{-1}$

The integers under addition (Z,+) form a group, as do the reals (R,+). A set with a closed operation which is just associative is a semigroup. A set with a closed operation which is both associative and has an identity is a monoid. A ring has a second dyadic operation which is distributive over the first operation. A field is a ring where the second operation forms an abelian group.

Used throughout cryptography, but particularly related to Pure Cipher.

Hadamard

See Walsh-Hadamard transform.

Hamming Distance

A measure of the difference or "distance" between two binary sequences of equal length; in particular, the number of bits which differ between the sequences. This is the weight or the number of 1-bits in the exclusive-OR of the two sequences.

Hardware

The physical realization of computation. Typically, the electronic digital logic, power supply, and various electro-mechanical components such as disk drives, switches, and possibly relays which make up a computer or other digital system. As opposed to software. See system design and debug.

Fundamental distinctions exist between hardware and software:

- By itself, software does not function. Only hardware can function. The best software can do is to present a list of operations for hardware to perform, when and if hardware gets around to performing operations. Hardware always does all computation, which ultimately limits the efficiency of computation.
- Hardware computation is not limited to the software concept of sequentiality: In hardware, each individual cipher layer can have separate computation hardware in a classic example of pipelineing. During each computation period, each of the multiple layers can be computed simultaneously, and adding layers does not reduce the data rate. That allows cipher computations like balanced block mixing to occur at a constant *block rate*, independent of block size, by adding hardware computation layers. Such a computation will have a latency for each added layer, but the computation time per byte goes down in proportion to block size.

Hash

A classic computer operation which forms a fixed-size result from an arbitrary amount of data. Ideally, even the smallest change to the input data will change about half of the bits in the result. Often used for table look-up, so that very similar language terms or phrases will be well-distributed throughout the table. Also often used for error-detection, and, known as a message digest, authentication. Also see salt.

Error Detection

For error detection, a hash of message data will produce a particular hash value, which then can be included in the message before it is sent (or stored or enciphered). When the data are received (or read or deciphered), the message is hashed again, and the result should match the included value. If the hash is different, something has changed, and the usual solution is to request the data be sent again. But the hash value is typically much smaller than the data, so there *must* be "many" different data sets which will produce that same value, which is called hash "collision." Because of this, "error detection" inherently cannot detect all possible errors, and this is independent of any linearity in the hash computation.

An excellent example of a hash function is a CRC operation. CRC is a linear function without cryptographic strength, but does have a strong mathematical basis which is lacking in *ad hoc* methods. Strength is not needed when keys are processed into the state or seed used in a random number generator, because if either the key or the state becomes known, the keyed cipher has been broken already. Strength is also not needed when a hash is used to accumulate uncertainty in data from a really random generator, since the hash construction cannot expose unknowable randomness anyway.

Cryptographic Hashing

In contrast, a cryptographic hash function such as that used for authentication must be "strong." That is, it must be "computationally infeasible" to find two input values which produce the same hash result. Otherwise, an opponent could produce a different message which hashes to the correct authentication value. In general, this means that a cryptographic hash function should be nonlinear overall and the hash state or result should be 256 bits or more in size (to prevent birthday attacks).

Sometimes a cryptographic hash function is described in the literature as being "collision free," which is a misnomer. A collision occurs when two different texts produce exactly the same hash result. Given enough texts, collisions will of course occur, precisely because any fixed-size result has only so many possible code values. The intent for a cryptographic hash is that collisions be hard to find (which implies a large internal state), and that particular hash values be impossible to create at will (which implies some sort of nonlinear construction).

Reversibility

A special cryptographic hash is *not* needed to assure that hash results do not expose the original data: When the amount of information hashed is substantially larger than the internal state or the amount of state ultimately exposed, many different data sequences will all produce the exact same hash result (again, "collision"). The inability to distinguish between the data sequences and so select "the" original is what makes a hash one way. This applies to all "reasonable" hash constructions independent of whether they are "cryptographic" or not. In fact, we can better guarantee the collision distributions when we have a relatively simple linear hash than if we must somehow analyze a complex ad hoc cryptographic hash.

On the other hand, when *less* information is hashed than the amount of revealed state, the hashing may be reversible, even if the hash is "cryptographic." And, again, that is independent of the strength of the hash transformation.

Distribution Flattening

Currently, almost all of cryptography is based on complex but deterministic (and, thus, at least potentially solvable) operations like ciphers and hashes. Because of the occasionally disastrous effectiveness of cryptanalysis, every cipher system has need of at least a few absolutely unpredictable values, which can be described as really random. Really random values have various uses, including message keys and protocol nonces. Generally, such values are obtained by attempting to detect or sample some molecular or atomic process, such as electrical noise.

For most cryptographic use, values should occur in a uniform distribution, so that no value will be predictable (by an opponent) any more than any other value. Unfortunately, few measurable molecular or atomic processes have a uniform distribution. As a consequence, some deterministic processing must be applied to somehow "flatten" the non-uniform distribution.

In statistics, and with real number values, it is common to simply compute an inverse and multiply. Unfortunately, that depends upon knowing the original distribution very well, but in practice the sampled distributions from quantum levels are not ideal and do vary.

No simple, fixed, integer value transformation can compensate for a distribution bias where some values appear more often than they should. Bias is a property of a set of values, not individual items, so treating individual values similarly seems unlikely to correct the problem. On the other hand, a transformation of *multiple* values, like block ciphering, can go a long way. Because a cipher block generally holds 64 bits or more worth of sample values, we might never see two identical plaintext blocks, and thus never produce a bias in the ciphertext. With a block cipher, any bias in the sample values tends to be hidden by the multiple values in a block, although at substantial expense.

Perhaps the most common way to flatten a distribution is to hash multiple sample values into a result for use. Using a CRC hash as an example, we can model a CRC operation as something like a large, fast modulo. Now, when the CRC is initialized to a fixed value, a particular input sequence always produces the same result, just like any other deterministic operation, including a cryptographic hash. So when inputs repeat, results repeat, and that carries the bias from the input to the output, even if only a subset of result bits are used. The worst possible situation would be to "hash" each sample value independently into a smaller result, since then the most frequent sample values would transfer the bias directly into the results. Normally, though, if we hash enough sample values at the same time, we expect the input sequence to "never" repeat, so the results should be almost completely corrected. Thus, the issue is not just having more input than output, but also having enough input so that any particular input string will "never" recur.

An improvement is to initialize the CRC to a random starting state before each hash operation. Because of the random initialization, any remaining bias (as in particularly frequent or infrequent values) will be distributed among all possible output values. When using a CRC for hashing, a separate random value is not required, since a random value is already in the CRC state as a result of the previous hash. Thus, what is required is simply to *not* initialize the CRC to a fixed value before each hash operation. For other hashes, the previous result could be hashed before new sample values.

To assure that the hash is not reversible, the hash operation must be overloaded; that is, at least twice as much information must be hashed as the size of the hash result or the amount exposed. And when bias must be corrected, a factor of 2.5 or more may be a better minimum. Reasonable choices might include a 16-bit CRC with 40 bits of normally-distributed input data, or a 32-bit CRC with 80 bits of input.

Hazard

Something which may cause a negative outcome. Also see risk.

Heuristic

1. Computation using a rule of thumb. Sometimes the opposite of an algorithm.
2. Learning by experiment, common sense and trial-and-error instead of analysis and proof. Also see scientific method.

In most sciences, the main point of a mathematical model is to predict reality, and experimentation is how we know reality. Experimentation thus sets the values that the mathematical model must reproduce. When there is a real difference between experiment and model (both being competently evaluated), the model is wrong.

In general, experimentation cannot know every possible parameter, or try every possible value, and so cannot assure us that something never happens, or that every possibility has been checked. That sort of thing typically requires a proof, but such proof is always based on the *assumption* that the mathematical model is sufficient and correct. Because experimentation often collects all measurable data, it is generally *better* than proof at finding unexpected happenings or relationships.

In cryptography, ciphers are basically approved for use by *experiments* which find that various attacks do not succeed. Absent various assumptions (such as: no other attacks are possible, and every approach has been fully investigated) that does not even begin to approach what we would consider actual proof of strength. Nevertheless, those results apparently are sufficient for the field of cryptography to place real users and real data at risk.

Since experimentation is the basis for all real use of cryptography, it does seem odd that experimentation is often scorned in mathematical cryptography.

Hex

Hexadecimal.

Hexadecimal

Base 16. The numerical representation in which each digit has an alphabet of sixteen symbols, generally 0 through 9, plus A through F, or "a" through "f".

Each hex value represents exactly four bits, which can be particularly convenient. Also see: binary, octal, and decimal.

Hidden Markov Model

A Markov chain with an unknown number of hidden states.

Hold Time

In electronic digital logic, the amount of time a signal voltage must be present and stable *after* the occurrence of a clock for the signal to be guaranteed to be recognized by all devices over all allowed variations in device processing, power supply, signal level, temperature, etc. Also see: setup time.

Homomorphism

In abstract algebra, a mapping @ from group *G* into group *H* which "preserves the group operation."

For group *G* with operation #, and group *H* with operation %; for mapping @ from group *G* to group *H*; given *a*, *b* in *G*: The result of the group *G* operation on *a* and *b*, when mapped into group *H*, must be the same as first mapping *a* and *b* into *H*, and then performing the group *H* operation:

```
   @(a # b) = @a % @b
```

A homomorphic mapping (the map @ from group *G* into group *H*) need not be one-to-one.

Given a homomorphism from group *G* into group *H* and mapping @ from *G* to *H*:

- If *e* is the identity of *G*, then *e* mapped into *H* is the identity of *H*.
- For *a* in *G*, the inverse of *a* in *G*, when mapped into *H*, is the same as first mapping *a* into *H* and finding the inverse in *H*.
- If *G* is abelian and the mapping from *G* to *H* is onto, then *H* is abelian.

Also see automorphism and isomorphism.

Homophonic

Greek for "the same sound." The concept of having different letter sequences which are pronounced alike. In cryptography, a cipher which translates a single plaintext symbol into any one of multiple ciphertext symbols which all have the same meaning. Also see polyphonic, polygraphic, monographic and block code.

Homophonic Substitution

A type of substitution in which an original symbol is replaced by any one of multiple unique symbols. That generally implies ciphertext expansion. Intended to combat the property of simple substitution in which the most-frequent symbols in the plaintext always produce the most-frequent symbols in the ciphertext.

A form of homophonic substitution is available in a large block cipher, where a homophonic selection field is enciphered along with the plaintext (also see block coding, balanced block mixing and huge block cipher advantages). Any of the possible values for that field naturally will produce a unique ciphertext. After deciphering any of those ciphertexts, the homophonic selection field could be deleted, and the exact same plaintext recovered. When the homophonic selection is really random, it adds a non-deterministic aspect to the cipher. Note that the ability to produce a multitude of different encipherings for exactly the same data is related to the concept of a key, especially dynamic keying, and the use of a salt in hashing. Also see the "The Homophonic Block Cipher Construction" conversation (locally, or @: http://www.ciphersbyritter.com/NEWS3/HOMOPHON.HTM).

HTTP Status Codes

HTTP/1.1 web page server responses to browser requests (RFC 2616):

```
   1xx  Information                 4xx  Browser request error
        100  Continue                    400  Bad request
        101  Ack protocol change         401  Unauthorized
   2xx  Success                         402  Payment required
        200  OK                          403  Forbidden
        201  Resource created            404  Not found
        202  Request accepted            405  Method not allowed
        203  OK, non-authoritative       406  Not acceptable
```

```
    204  Empty                           407  Proxy authentication required
    205  Content reset                   408  Request timed out
    206  Partial content                 409  Conflict with state of resource
3xx  Redirection                       410  Resource no longer available
    300  Multiple locations              411  Length value required
    301  Moved permanently               412  Precondition failed
    302  Found in <location>             413  Requested file too large
    303  Use <location>                  414  Requested address too long
    304  Not modified                    415  Requested data on unsupported media
    305  Must use <proxy>                416  Requested range not satisfiable
    306  Unused                          417  Expectation failed
    307  Temp <location>             5xx  Server error
                                                    500  Internal error
                                                    501  Not implemented
                                                    502  Bad gateway
                                                    503  Service unavailable
                                                    504  Gateway timeout
                                                    505  HTTP version not supported
```

Huge Block Cipher Advantages

A "huge block cipher" is my term for a conventional block cipher with a maximum block size substantially larger than the common 64-bit, 128-bit or 256-bit block. Most huge block ciphers will be scalable, and so may select a wide range of blocks, including tiny blocks, but also including 256-byte, 512-byte or even 4K byte blocks. These sizes can be practical, given FFT-like networks of Balanced Block Mixing operations. Mixing Ciphers can be made to select block width in power-of-2 steps at ciphering time. (See Mixing Cipher Design Strategy.)

Various advantages can accrue from huge blocks (although not all simultaneously):

1. The possibility of using naive ECB mode, which means:
   a. No need for plaintext randomization (e.g., CBC mode).
   b. No ciphertext expansion from an IV.
   c. An ability to cipher and decipher blocks out of order.
2. The opportunity to add an "extra data" field of reasonable size to the plaintext block while retaining reasonable overall efficiency. That means ciphertext expansion, but the data could be more important when used for:
   a. Per-block Dynamic Keying
   b. Per-block Authentication
   c. Homophonic operation
3. Potentially much higher speed in hardware.

When we have a huge block of plaintext, we may expect it to contain enough (at least, say, 64 bits) uniqueness or entropy to prevent a codebook attack, which is the ECB weakness. In that case, ECB mode has the advantage of supporting the independent ciphering of each block. This, in turn, supports various things, such as the use of multiple ciphering hardware operating in parallel for higher speeds.

As another example, modern packet-switching network technologies often deliver raw packets out of order. The packets will be re-ordered eventually, but we cannot start deciphering until we have a full block in the correct order. But we might avoid delay if blocks are ciphered independently.

A similar issue can make per-block authentication very useful. Typically, authentication requires a scan of plaintext, and then some structure to transport the authentication value with the ciphertext, much like a common error detecting code. The problem is that all the data which are to be authenticated in one shot, which often means the whole deciphered file, must be buffered until an authentication result is reached. We certainly cannot use the data until it is authenticated. So we have to buffer all that data as we get it, but cannot use *any* of it until we get it *all* and find that it checks out. We can avoid that overhead and latency by using per-block authentication.

To implement per-block authentication, we use a keyed cryptographic RNG which produces a keyed sequence of values. Both ends produce the same keyed sequence by using the same key. We place a different random value in each block sent, and then compare that to the result as each block is received. This is very much like a per-block version of a Message Key.

Normally, in software, the more computation there is, the longer it takes, but that is not necessarily true in hardware, if we are willing to build or buy more hardware. In particular, we can pipeline hardware computations so that, once we fill the pipeline (a modest latency), we get a full block result on every computation period (e.g., on every clock pulse). Thus, huge blocks can be much faster in hardware than software: the larger the block, the larger the data rate, for any given hardware implementation technology.

Also see All or Nothing Transform and Balanced Block Mixing.

Also see:

- "Huge Block Size Discussion," locally, or @: http://www.ciphersbyritter.com/NEWS/HUGEBLK.HTM
- the "Large Block DES" on-line development project, locally, or @: http://www.ciphersbyritter.com/NEWS/LBDNEWS.HTM

Hybrid

Something of mixed origin. In cryptography, typically a cipher system containing both public key and secret key component ciphers. Typically, the public key system is used only to transport the key for the secret key cipher. It is the secret key cipher which actually enciphers and protects the data.

Hypothesis

1. In the study of logic, an assumption made for the purpose of evaluating the resulting conclusions. Also see: thesis, analysis, synthesis and argumentation.
2. In science and statistics, a proposed model of reality. As opposed to a speculation or belief. Normally, an hypothesis proposes a relationship between an *independent* variable (the cause) and a *dependent* variable (the effect), and thus relates one measurable quantity to another. Experimentation then quantifies that relationship, or fails to do so. Also see: null hypothesis and alternative or research hypothesis.

In both logic and science, some hypotheses are better than others. In logic, hypotheses are the same if they have the same formal structure. But in science, hypotheses with the exact same structure may or may not be appropriate in particular contexts. A scientific hypothesis must be:

- **Testable:** it must make "predictions" that can be verified experimentally or by comparison to factual data, and
- **Falsifiable:** it must allow experiments or comparisons to confirm or especially to disprove the hypothesis.

Hypotheses structured so that we can only develop evidence for the question by investigating an essentially unlimited number of possibilities are *untestable*. Hypotheses in which *no experiment of any kind* can disprove the question are *unfalsifiable*, and are best seen as mere beliefs. Hypotheses which apply, say, to all matter, are unprovable absent testing on all matter, but any one (repeatable) experiment *could* disprove the question. Most scientific hypotheses are structured so that they can be proven false or *confirmed* by experiment, but cannot ever be proven true.

Many scientific questions are formally unproven in the sense that they address the operation of all matter across time, only a tiny subset of which can be sampled experimentally. But most scientific issues also admit *quantifiable* experimentation which makes it possible to *bound* the interpretation of reality. Quantifiable experimentation makes it possible to *compare* different trials of the same experiment and see if things work about the same each time, with each material, and in each place. Of course, getting precisely similar values in each case may just be a coincidence, which is why it is not proof. But each trial does add to a growing mass of evidence for an overall similarity which, while not proof, does provide both statistical and factual support.

Cipher Strength

The usual logic of scientific experimentation is not available in cryptography, in that cryptography has no general, *quantifiable* test of strength. We only know the strength of a cipher when an actual attack is found (and then only know the strength under that particular attack); until then we know nothing at all about strength. Until an attack is found, there is no experimental strength value, so cryptanalytic experiments cannot develop bounds on strength. Nor are factual and comparable values developed. As a result, cryptographic proof appears to require what Science knows cannot be done: a testing of every possibility simply to show that none of them work.

For example, in cryptography, we may wish to assert that a certain cipher is strong, or unbreakable by any means. (We can insert "practical" with little effect.) A cipher is unbreakable when no possible attack can break it. So to prove that, we apparently first must identify every possible attack, and then check each to see if any succeed on the cipher under test. But not only do we *not* know every possible attack, it seems unlikely that we *could* know every possible attack, or even how many there are. Thus, the assertion of strength seems *unprovable*.

Even if we *could* know every possible attack, we still have problems: Since attacks are classified as *approaches* (rather than algorithms), it seems necessary to phrase each in ways guaranteed to cover every possible use. Yet it seems unlikely that we could be guaranteed to know every possible ramification of even one approach. Without comprehensive algorithms, it is hard to see how we could provably know that *any* particular approach could not work. So again the hypothesis of cipher strength seems *unprovable* and unhelpful.

On the other hand, for each well-defined attack algorithm, we probably *can* decide whether that would break any particular cipher. So it is at least conceivable that we *can* have proof of strength against particular explicit attack algorithms. Unfortunately, in most cases, attack approaches must be modified for each individual cipher before an appropriate algorithm is available, and failure might just indicate a poorly-adapted approach. So algorithmic tests are not particularly helpful.

Random Sequences

Similar issues occur with respect to random bit sequences: A sequence is *random* if no possible technique can extrapolate future bits from all past ones, succeeding other than half the time. Again, we do not and probably can not know every possible extrapolation technique, and as the set of "past" bits grows, so do the number of possible techniques. Not knowing each possible technique, we certainly cannot check each one, making the hypothesis of sequence randomness seemingly *unprovable*.

Fortunately, we do have some defined algorithms for statistical randomness tests. Thus, what we *can* say is that a particular test has found no pattern, and that test can be repeated by various workers on various parts of the sequence for confirmation. What that does **not** do is build evidence for results from *other* tests, and the number of such tests is probably unbounded. So, again, the hypothesis of randomness seems *unprovable*.

Scientific Strength

Even if we *could* develop cryptographic proof to the same level as natural law, that probably would not be useful. People already believe ciphers are strong, even *without* supporting evidence. Belief *with* supporting evidence would, of course, be far preferable. But what is *really* needed is not more belief, but actual affirmative proof of strength. And that is beyond what science can provide even for ordinary natural law.

Why is lack of absolute proof acceptable in science and not in cryptography? In science, the issue is not normally whether an effect exists, but instead the exact nature of that effect. When an apple falls, we see gravity in action, we know it exists, and then the argument is how it works in detail. Even when science pursues an unknown effect, it does so numerically in the context of experiments which *measure* the property in question.

When a cipher operates, all we see is the operation of a computing mechanism; we cannot see secrecy or security to know if they even exist, let alone know how much we have. Security cannot be measured because the appropriate context is our opponents, and they are not talking. While we may know that *we* could not penetrate security, that is completely irrelevant unless we know that the same applies to our opponents.

Does all this mean cryptography is hopeless? Well, absolute proof of absolute security seems unlikely. But we *can* seek to manage the risk of failure, particularly because we cannot know how large that risk actually is. For example, airplanes are designed with layers of redundancy specifically to avoid catastrophic results from single subsystem failures (see single point of failure). In cryptography, we could seek to not allow the failure of any single cipher to breach security. It would seem that we could approach that by multiple encryption and by dynamically selecting ciphers, which are parts of the Shannon Algebra of Secrecy Systems.

IDEA

The secret key block cipher used in PGP. Designed by James Massey and Xuejia Lai in several installments, called PES, IPES and IDEA. It is round-based, with a 64-bit block size, a 128-bit key, and no internal tables.

The disturbing aspect of the IDEA design is the extensive use of *almost* linear operations, and no nonlinear tables at all. While technically *non*linear, the internal operations seem like they might well be linear *enough* to be attacked.

Ideal Mixing

My term for a form of block mixing which guarantees that for a value change on any one input, *every* output will change. When each output is connected to a substitution table or S-box, all of the boxes will be active. See Balanced Block Mixing.

Ideal Secrecy

The strength delivered by even a simple cipher when each and every plaintext is equally probable and independent of every other plaintext. This is plaintext balance, which is sometimes approached by whitening, although producing true independence remains a problem.

> "With a finite key size, the equivocation of key and message generally approaches zero, but not necessarily so. In fact, it is possible for . . . $H_E(K)$ and $H_E(M)$ to not approach zero as N approaches infinity."
>
> "An example is a simple substitution on an artificial language in which all letters are equiprobable and successive letters independently chosen."
>
> "To approximate the ideal equivocation, one may first operate on the message with a transducer which removes all redundancies. After this, almost any simple ciphering system -- substitution, transposition, Vigenere, etc., is satisfactory."
>
> -- Shannon, C. E. 1949. Communication Theory of Secrecy Systems. *Bell System Technical Journal.* 28:656-715.

There are various examples:

- The use of CBC mode in DES: By making every plaintext block equally probable, DES is greatly strengthened against codebook attack. (Unfortunately, the *public* block randomization provided by CBC does not hide plaintext statistics, and so does not hinder brute force attack.)

- The transmission of random message key values: To the extent that every value is equally probable, even a very simple cipher is sufficient to protect those values.
- The use of a keyed simple substitution *of the ciphertext* to add strength, as used in the Penknife stream cipher design.
- The use of data compression to reduce the redundancy in a message before ciphering: This of course can only *reduce* language redundancy. (Also, many compression techniques send pre-defined tables before the data and so are not suitable in this application.)

Also see: pure cipher, Perfect Secrecy, unicity distance and balance.

Identity
An equation which is true for every possible value of the unknowns.

Identity Element
Given a dyadic operation which combines any two elements of a set, an element *e*, which when combined with any other element *x*, is just that element *x* itself.

i.i.d.
In statistics: Independent, Identically Distributed. Generally related to the random sampling of a single distribution.

Imaginary Number
Those numbers which, when paired with reals, form the complex numbers. The imaginaries are reals multipled by the square root of -1, which represents a 90 degree or Pi/2 rotation on a Cartesian grid.

Impedance
In electronics, the effective contribution of both reactance and resistance. Impedance is the magnitude of the complex number composed of scalar resistance and scalar reactance, the square-root of the sum of the squares of the two values.

Impedance Matching
In electronics, the basic idea that there is an advantage to matching a *source* or *generator* impedance to the *load*, typically by transformer.

Consider what it would mean to make the load match the impedance of a generator: As we decrease the load impedance toward the generator impedance, more current will flow into the load and more power will be transferred. But as we deliver more power, more power is also dissipated in the generator, and thus simply lost to heat. We deliver the maximum possible power to a load when the load has the same impedance as the generator, but then we lose as much power in the generator as we manage to transfer. An efficiency of 50 percent is generally a bad idea for any signal, low-level, high-level or power.

As the load impedance is decreased below the generator impedance, now *less* power is delivered to the load, and even *more* power is dissipated as heat in the generator. The limit is what happens when a power amplifier output is shorted. In practice, adding speakers in parallel to an amplifier output can so reduce the load impedance to cause the amplifier to overheat or self-protect or fail.

Audio Matching

In most audio work, there is little desire to "match" impedances. Normally, signals are produced by low-impedance sources (e.g., preamplifier outputs) for connection to high-impedance inputs (e.g., amplifier inputs). When transformers are used, they often create balanced lines, receive balanced signals, and provide ground loop isolation.

Impedance matching tends to be of the most concern for electro-mechanical devices. The fidelity of sensitive mechanical sensors like phonograph cartridges and professional microphones can be affected by their loads. For best performance, it is important to present the correct load impedance for each device, which is a serious impedance matching requirement. However, nowadays this is often accomplished trivially with an appropriate load resistor across a high-impedance input to an amplifier or preamp. In contrast, loudspeakers, which are also electro-mechanical, are almost universally "voltage driven" devices. Speakers are specifically designed to be driven from an source having an impedance much lower than their own.

One old application somewhat like matching is the classic "input transformer": These devices take a low-level low-impedance signal to a larger signal (typically ten times the original, or more), but then necessarily at a much, much higher impedance. When used with an amplifier that has a high input impedance anyway, an input transformer can deliver a greater signal, without the noise of a low-level amplification stage. However, bipolar transistor input stages generally want to see a low impedance source for best noise performance, so the advantage seems limited to somewhat noisier FET and tube input circuits.

Good input transformers, with a wide and flat frequency response, are *very* expensive and can be surprisingly sensitive to nearby AC magnetic fields. Although a thin metal shield is sufficient to protect against RF fields, sheet metal steel will not diminish low-frequency magnetic fields much at all. Mu metal shields may help, although distance is the usual solution.

Power Matching

In power transformers, losing as much power to heat as we deliver would be absolutely ridiculous. We may transform AC power to get the voltage we need, but we do not "match" the equipment load to the impedance of the AC line.

RF Matching

In radio frequency (RF) work, impedance matching *is* needed to properly use coaxial cables. Using source and load impedances appropriate for the coax minimizes "standing waves" on that coax. Standing waves increase current at voltage minima and thus increase signal loss, even for low-level signals. Standing waves also can cause voltage breakdown at voltage maxima, which may include transmitter tuning capacitors or output transistors. And almost any passive filter will require a known load impedance.

Impossible

Something which absolutely cannot happen, under any condition, over any amount of time. Making information exposure "impossible" is often a goal in cryptography. But it is not uncommon for something believed "impossible" to turn out to be not even all that improbable. Also see: scientific method and proof.

Improbable

Unlikely. Something believed to occur very infrequently. For example, modern cryptography is based on the use of keys. With any finite number of keys, it is always possible for an opponent to accidentally choose the correct one and so expose an enciphered message. (But that is not a break unless one can do it at will and with less effort than a brute-force search.) Choosing the correct key by accident is made "improbable" by having many keys, but no matter how many keys there are, the possibility of exposure still exists. Thus, proving that a cipher has astronomical numbers of keys still does not prove the cipher will protect information in every possible case.

Independent

In statistics, events are independent when the occurrence of one event does not change the probability of another. Simple independence can be argued from first principles, or measured with simple probability experiments. In complex reality, however, it may be difficult to know which of many different events may combine to influence another. Also see correlation and rule of thumb.

Independent Variable

The input to a function; a parameter to the function.

Inductive Reasoning

In the study of logic, reasoning by generalizing multiple examples into a single overall idea or statement, sometimes by analogy. While often incorrect, inductive reasoning does provide a way to go beyond known truth to new statements which may then be tested by observation and experiment. In contrast to deductive reasoning. Certain types of inductive reasoning can be assigned a correctness probability using statistical techniques. Also see: argument, fallacy, proof and scientific method.

Inductor

A basic electronic component which acts as a reservoir for electrical power in the form of current. An inductor acts to "even out" the current flowing through it, and to "emphasize" current changes across the terminals. An inductor conducts DC and opposes AC in proportion to frequency. Inductance is measured in Henrys: A voltage of 1 Volt across an inductance of 1 Henry produces a current change of 1 Ampere per second through the inductor. A current change of 1 Ampere per second through an ideal 1 Henry inductor generates a constant 1 Volt across the inductor.

If we know the inductance *L* in Henrys and the frequency *f* in Hertz, the inductive reactance X_L in Ohms is:

```
XL = 2 Pi f L
Pi = 3.14159...
```

Separate inductors in series are additive. However, turns on the same core increase inductance as the square of the total turns. Two separate inductors in parallel have a total inductance which is the product of the inductances divided by their sum.

An inductor is typically a coil or multiple turns of conductor wound on a magnetic or *ferrous* core, or even just a few turns of wire in air. Even a short, straight wire has inductance. Current in the conductor creates a magnetic field, thus "storing" charge. When power is removed, the magnetic field collapses to maintain the current flow; this can produce high voltages, as in automobile spark coils.

An inductor is one "winding" of a transformer.

The simple physical model of a component which is a simple inductance and nothing else works well at low frequencies and moderate impedances. But at RF frequencies and modern digital rates, there is no "pure" inductance. Instead, each inductor has a series resistance and parallel capacitance that may well affect the larger circuit.

Also see capacitor and resistor.

Inference

In the study of logic, drawing conclusions from given premises. There are various types:

- Deductive
- Inductive
- Probabilistic -- drawing a likelihood from known frequencies within the domain of interest
- Statistical -- drawing a likelihood from results of experiments in a generally uncharacterized domain.

Informal Proof

In mathematics, a proof which *does* depend upon the meanings of the terms, and thus is difficult or impossible to verify by logic machine. Informal proofs depend upon specific interpretations for their terms, and so depend upon being interpreted in a particular context; outside of that context the proof will not apply. Specifically defining the appropriate context is an important part of developing informal proofs. As opposed to a formal proof.

Many proofs are informal. Because they cannot be mechanically verified, these proofs may have unseen problems, and often do develop and change over time. See: Method of Proof and Refutations.

Information

In information theory, a measure of one's freedom of choice when one selects a message. Measured as the logarithm of the number of available choices. Also see entropy.

Injective

One-to-one. A mapping f: *X* -> *Y* where no two values *x* in *X* produce the same result *f(x)* in *Y*. A one-to-one mapping is invertible for those values of *X* which produce unique results *f(x)*, but there may not be a full inverse mapping g: *Y* -> *X*.

Innuendo

The *hint* of a negative conclusion which is never stated directly. See: Argument By Innuendo.

Insulator

A material in which electron flow is difficult or impossible. Classically air or vacuum, or wood, paper, glass, ceramic, plastic, etc. As opposed to a conductor and semiconductor.

As a rule of thumb, a cubic centimeter (cc) of a solid has about 10^{24} or 1E24 atoms. A good insulator like quartz has only about 10 free electrons per cc., which implies that only about one atom in 10^{23} (1E23) has a broken bond (at room temperature and modest voltage). This gives a massive resistance to current flow of about 10^{18} (1E18) ohms across a centimeter cube.

Integer

An element in the set consisting of counting numbers: 1, 2, 3, ..., their negatives: -1, -2, -3, ..., and zero, and denoted Z. Alternately, the natural numbers: 0, 1, 2, ..., and their negatives: 0, -1, -2, ...

The integers are closed under addition and multiplication, but not under division. (Z,+,*) is thus an infinite ring, but not a field. However, a finite field, denoted Z/p, can be created by doing operations modulo a prime.

Integrity

1. An objective of cryptography. The idea that information is what it appears to be: uncorrupted (unchanged) information from the source.
2. Adherence to a code of (typically) moral or intellectual values. Also see: data fabrication and data falsification.

Intellectual Property

The idea that the creation of intellectual objects (such as paintings, writings, and inventions) is sufficiently worthwhile to be given legal protection against unrestricted replication. Writers and inventors can thus profit from their own work, although only if someone wishes to buy that work: Legal protection is not a direct financial grant. See the United States Patent and Trademark Office (PTO) (http://www.uspto.gov/), and the United States Copyright Office (in the Library of Congress) (http://lcweb.loc.gov/copyright/) sites.

In the United States, the basis of intellectual property law is the Constitution, in Article 1, Section 8 (Powers of Congress):

> "Congress shall have power . . . To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries."

Intellectual property generally includes:

- **Trade Secrecy** is the right to keep formlas or inventions secret, and nevertheless profit from the resulting products or use. In the U.S., this is generally state law.
- **Trademarks** typically are symbols used on marketed goods to identify a particular maker. In the U.S., an issue of federal law.
- **Copyright** is a long-term grant by which the owner of a creative work can recover damages and penalties when others reproduce that work. Note that copyright protects written works as a particular fixed *expression* of an idea, but not the idea itself. In the U.S., this is federal law.
- **Patent** is a much shorter-term grant by which the owner of an invention can recover damages and penalties when others *make*, *sell*, or *use* unlicensed copies of the invention. A patent protects the essential functioning of an invention, and thus can protect the idea itself, in many different implementations. Because of international "harmonization" efforts, U.S. patent law generally *does* correspond to the patent laws of many other countries, even with respect to software. Of course, U.S. Patents apply only in the U.S.

  In the U.S., there are three types of patent:
  - **Plant Patents**;
  - **Design Patents**, which protect the ornamental physical shape of an object, but do *not* protect functioning or operation; and
  - **Utility Patents**, the conventional patent we think of in a technical context, which do protect the *functioning* of something which may be realized in many different forms, all of which would be protected. Utility patents require a complex and detailed application which must be argued, corrected and approved, plus application fees, issue fees and periodic maintenance fees.

Intermediate Block

In the context of a layered block cipher, the data values produced by one layer then used by the next.

In some realizations, an intermediate block might be wired connections between layer hardware. In the context of a general purpose computer, an intermediate block might represent the movement of data between operations, or perhaps transient storage in the original block.

Interval

In statistics, measurements in which the numerical value has meaning. Also see: nominal, and ordinal.

Into

A mapping f: *X* -> *Y* which only partially covers Y. An inverse mapping g: *Y* -> *X* may not exist if, for example, multiple elements *x* in *X* produce the same *f(x)* in *Y*.

```
+----------+          +----------+
|          |   INTO   | Y        |
|    X     |          |   +----+ |
|          |    f     |   |f(X)| |
|          |   --->   |   +----+ |
+----------+          +----------+
```

Invention

A useful yet unobvious novel combination. Also see patent.

Inverse

A mapping or function *g(y)* or $f^{-1}(y)$, related to some function *f(x)* such that for each *x* in *X*:

```
   g( f(x) ) = x = f^-1( f(x) ).
```

Only functions which are one-to-one can have an inverse.

Inverter

1. In electronics, a device which converts low-voltage DC battery power into high-voltage AC power.
2. The logic function which performs the NOT operation; a logic "gate" which outputs the compliment of the binary value on the input.

Invertible

A mapping or function which has an inverse. A transformation which can be reversed.

Involution

A type of mapping which is a self inverse. Function *f*: X -> Y, where, for all *x* in *X*:

```
   f( f(x) ) = x.
```

(Contrast with: bijection.)

A cipher which takes plaintext to ciphertext, and ciphertext back to plaintext, using the exact same operation.

Irreducible

A polynomial only evenly divisible by itself and 1. The polynomial analogy to integer primes. Often used to generate a *residue class field* for polynomial operations.

A polynomial form of the ever-popular "Sieve of Eratosthenes" can be used to build table of irreducibles through degree 16. That table can then be used to check any potential irreducible through degree 32. While slow, this can be a simple, clear validation of other techniques.

Also see primitive polynomial.

Iterated Block Cipher

Typically, a block cipher in which weak ciphering operations are repeatedly applied in rounds. Also see product cipher.

Isormorphism

In abstract algebra, the special case of a homomorphism which is also a one-to-one mapping. Mapping @ from group G to group H where every value in G is represented by a value in H.

IV

"Initial value," "initializing value" or "initialization vector." An external value needed to start cipher operations. Most often associated with the CBC block cipher operating mode.

Usually the main purpose of an IV is to randomize or whiten the plaintext data. As a result, plaintext (and, thus, ciphertext) repetition is made less likely, thus greatly reducing exposure to codebook attack.

Generally, an IV must be accompany the ciphertext, and so always expands the ciphertext by the size of the IV.

While it is often said that IV values need only be random-like or unpredictable, and need not be confidential, in the case of CBC mode, that advice can lead to man-in-the-middle attacks on the first plaintext block. If a MITM opponent knows the usual content of the first block, they can change the IV to manipulate that block (and only that block) to deliver a different address, or different dollar amounts, or different commands, or whatever. And while the conventional advice is to use a MAC at a higher level to detect changed plaintext, that is not always desirable or properly executed. But the CBC first-block problem is easily solved at the CBC level simply by enciphering the IV and otherwise keeping it confidential, and that can be reasonable even when a MAC will be used later.

Sometimes, iterative or repeated ciphering under different IV values can provide sufficient added keying to perform the message key function (e.g., the "iterative stream cipher" in a cipher taxonomy).

Jitter

Variations in the period of a repetitive signal. These period variations also can be seen as frequency or phase variations.

Often discussed with respect to oscillator signals. Oscillator jitter is commonly due to the small amounts of analog noise inherent in the physics of electronic circuitry, which thus affects the analog-to-digital conversion which indicates the start of each new period. This is unpredictable variation, but generally very tiny, bipolar around some mean frequency, and varies on a cycle-by-cycle basis. It cannot be accumulated over many cycles for easier sensing.

A different form of jitter occurs when a digital system uses two or more independent oscillators or clocks which are not synchronized. In this case, one signal may slide early or late with respect to the other, until an entire cycle is lost or skipped. But all this will be largely deterministic, based on the frequencies and phases of the different clocks. To a large extent, this is something like two brass gears rolling together, with a particular tooth on the smaller gear appearing a predictable number of teeth later on the larger gear.

Jitterizer

A particular cryptographic mechanism intended to complicate the sequence produced by a linear random number generator by deleting elements from the sequence at pseudo-random. See the articles:

- "The Cloak2 Cipher Design" locally, or @: http://www.ciphersbyritter.com/CLO2DESN.HTM
- and "A Keyed Shuffling System for Block Cipher Cryptography" locally, or @: http://www.ciphersbyritter.com/KEYSHUF.HTM

The name "jitterizer" was established in section 5.5 of my 1991 *Cryptologia* article: "The Efficient Generation of Cryptographic Confusion Sequences" (locally, or @: http://www.ciphersbyritter.com/ARTS/CRNG2ART.HTM#Sect5.5) and is taken from the use of an oscilloscope on digital circuits, where a signal which is not "in sync" is said to jitter. Mechanisms designed to restore synchronization are called "synchronizers," so mechanisms designed to cause jitter can legitimately be called "jitterizers."

Johnson Noise

Thermal noise. The voltage developed across any resistance due to the random thermal motion of charge carriers.

Just Semantics

See mere semantics.

Karnaugh Map

A graphical technique for minimizing the complexity or the number of logic function gates.

KB

Kilobyte. 2^{10} or 1024 bytes.

Kb

Kilobit. 2^{10} or 1024 bits.

Kerckhoffs' Requirements

General cryptosystem requirements formulated by Auguste Kerckhoffs in 1883:

> Kerckhoffs, Auguste. 1883. La cryptographie militaire. *Journal des sciences militaires*. IX(1):5–38, IX(2):161–191.

Various texts and papers are rather casual about what Kerchkhoffs supposedly wrote. Fortunately, the original article (in the original French) is available on-line for comparison:

- See: http://www.petitcolas.net/fabien/kerckhoffs/index.html
- Part I: http://www.petitcolas.net/fabien/kerckhoffs/la_cryptographie_militaire_i.htm
  or http://www.petitcolas.net/fabien/kerckhoffs/crypto_militaire_1.pdf
- Part II: http://www.petitcolas.net/fabien/kerckhoffs/la_cryptographie_militaire_ii.htm

Also see: http://en.wikipedia.org/wiki/Auguste_Kerckhoffs

1. **"The system should be practically, if not theoretically, unbreakable."**
   (Unfortunately, and to a large extent forgotten or coarsely ignored by conventional cryptography, is the truth that there *are* **no** actual, implemented, realized systems which are "theoretically unbreakable" (see, for example, one-time pad, proof, strength and cryptanalysis). And almost universally ignored is the fact that cryptography has no test or measure to show or even testify that a cipher is "practically unbreakable" (see, for example, cryptanalysis): Just because our guys cannot break it does not imply that the opponents are similarly limited. But there does seem little point in using a known breakable cipher, so we make do as best we can.)
2. **"Compromise of the system details should not inconvenience the correspondents."**
   (This requirement is often cited as a basis for stating that the security of a cryptosystem must depend only on the key, and not on the secrecy of any other part of the system. That is fine as far as it goes, but in systems which select among different ciphers *using a key*, while the general concept of selection would of course be known to an opponent, the actual cipher selected by the key would *not*.

   Systems which select among an ever-increasing number of ciphers can even make it difficult for an opponent to know the full set of possible ciphers. For the opponents, being forced to find, obtain and analyze a continuing flow of new secret ciphers is vastly more expensive than simply trying another key value in a known cipher. Forcing the opponent to *pay* (in effort) to acquire each of many cipher designs is not a bad idea. While having many possible ciphers does not guarantee strength, it should increase the cost of attacks and thus potentially change the balance of power between user and attacker.

   Kerckhoffs second requirement is also understood to discount secret ciphers, as in security through obscurity. We of course want to use only ciphers we can continue to use securely even when the cipher has been fully exposed. But we certainly can use ciphers that *start out* secret, even if we understand that eventually they will become exposed.

   Note that the issue of secret ciphers is not stated directly by Kerckhoffs, but is instead extrapolated from what he wrote. What Kerckhoffs *really says* is that cipher exposure should not cause "inconvenience." But to the extent that "inconvenience" is an issue, various *other* ramifications appear that the crypto texts studiously ignore:
   - When we have only one cipher, any newly-found weakness will require system-wide upgrades and be a major "inconvenience" for every user. We can thus take Kerckhoffs second requirement

as demanding the ability to easily replace current ciphers with new ones. Then if weakness is found in some particular cipher, we simply use something else. The same cryptography which loudly proclaims Kerckhoffs requirements as a basis for modern design deliberately ignores the "inconvenience" of cipher failure.

- Each new cipher should have as much cryptanalysis as we can afford. But if we could really trust even a heavily-cryptanalyzed standard cipher, we would never need to change ciphers in the first place. It is sadly "inconvenient" that no such trust is possible, either in new ciphers, or in old ones. Any cipher can have some sort of hidden weakness we have not yet found. To combat unknown weakness we can use multiple encryption with three ciphers in sequence. A good understanding of the Kerckhoffs requirements would seem to demand *something* beyond current cryptographic practice.
- Cryptographic orthodoxy has all of us use a single cipher for years on end. Naturally, our opponents will know which cipher we use, and try to break it. Opponents can well afford expensive and lengthy analysis, because success means they can expose data from anyone and everyone, for perhaps the next decade. Since we cannot know whether they have succeeded, our continuing use of the same cipher is a terrible risk. The risk is that of a single point of failure, a single cipher which, if broken, exposes everything. A less "inconvenient" alternative is to use a system of multiple ciphers, as proposed in Shannon's 1949 Algebra of Secrecy Systems. In a modern computer implementation we might dynamically select a cipher for use from among a continuously increasing set of ciphers. That would distribute protected information from different users and different times under wide variety of different cipherings, which would reduce the benefit opponents would get from breaking any particular one. Distributing new ciphers is not an issue: To the extent that we can securely transfer a new key, we can also securely transfer the name of the new cipher, or even the actual code.)

3. **"The key should be rememberable without notes and easily changed."**
   (This is still an issue. Hashing allows us to use long language phrases, but the best approach may someday be to have both a hardware key card *and* a key phrase.)
4. **"The cryptogram should be transmissible by telegraph."**
   (This is not very important nowadays, since even binary ciphertext can be converted into ASCII or base-64 for transmission if necessary.)
5. **"The encryption apparatus should be portable and operable by a single person."**
   (Software encryption approaches this ideal.)
6. **"The system should be easy, requiring neither the knowledge of a long list of rules nor mental strain."**
   (Software encryption has the *potential* to approach this, but often fails to do so. We might think of the absolute requirement for certifying public keys, which is still often left up to the user, and thus often does not occur.)

Key

The general concept of protecting things with a "lock," thus making those things available only if one has the correct "key." In a cipher, the ability to select a particular one of many possible transformations between a plaintext message and the corresponding ciphertext. By supporting a vast number of different key possibilities (a large keyspace), we hope to make it impossible for someone to decipher the message by trying every key in a brute force attack (but see key problems).

In cryptography we have various kinds of keys, including a User Key (the key which a user actually remembers), which may be the same as an Alias Key (the key for an alias file which relates correspondent names with their individual keys). We may also have an Individual Key (the key actually used for a particular correspondent); a Message Key (normally a random value which differs for each and every message); a Running Key (the confusion sequence in a stream cipher, normally produced by a random number generator); and perhaps other forms of key as well (also see key management).

In general, the value of a cryptographic key is used to initialize the state of a cryptographic mechanism such as some form of RNG. The RNG then may be used to create a sequence which eventually becomes a running key for a stream cipher, or perhaps for a Dynamic Transposition block cipher. Alternately, the RNG may be used to shuffle and, thus, key, S-boxes for other forms of block cipher. Any particular cipher system is likely to have a sequence of keys, starting with a human-friendly keyphrase, through a hashed version of that phrase, through other files and selections, until some ultimate value is used to protect data.

Ideally, a key will be an arbitrary equiprobable selection among a huge number of possibilities (also see balance). This is the fundamental strength of cryptography, the "needle in a haystack" of false possibilities. But if a key is in some way *not* a random selection from a uniform distribution, but is instead biased, the most-likely keys can be examined first, thus reducing the complexity of the search and the effective keyspace.

In most cases, a key will exhibit diffusion across the message; that is, changing even one bit of a key should change every bit in the message with probability 0.5. A key with lesser diffusion may succumb to some sort of divide and conquer attack.

For practical security, it is not sufficient to simply *have* a large keyspace, it is also necessary to *use* that keyspace. Because changing keys can be difficult, there is often great temptation to assign a single key and then use that key forever. But if that key is exposed, not only are the current messages revealed, but also all other messages both past and future, and this is true for both public-key and secret-key ciphers. Using only one key makes that key as valuable as all the information it protects, and it is probably impossible to secure any key that well, especially if it is frequently used. Humans make mistakes, people change jobs and loyalties, and employees can be intimidated, tempted or blackmailed.

It is important to change keys periodically, thus decisively ending any previous exposure. Secret-key systems can make this fairly invisible by keeping an encrypted alias file and automatically translating a name or channel identifier into the current key. This supports the easy use of many secret keys, and the invisible update of those keys. Then only the keyphrase for the alias file itself need be rememebered, and that keyphrase could be changed at will. Old keys could be removed from the alias file periodically to reduce the amount of information exposed by that file, and so minimize the consequences of exposure. See: key reuse.

Even with support from the cipher system, key file maintenance is always serious and potentially complex. The deciphered alias file itself should never appear either on-screen or as a printout; it should be edited automatically or indirectly. Thus, some security officer probably needs to be in charge of updating and archiving the alias files.

Secret versus Public Keys

For some unknown reason, some authors claim that secret key ciphers are essentially impractical. That of course flies in the face of many decades of extensive actual use of secret-key ciphers in the military. The claim seems to be that secret-key ciphers require vastly more keys to be set up and managed than public key ciphers. There is an argument there, as we shall see, but it is not a good one.

Public-key ciphering generally requires an entity beyond the ciphering parties simply to function. This is a public key infrastructure (PKI), of which the main element is a certification authority (CA). The CA distributes authenticated keys for use, and so must be set up and supported and protected as long as new keys or even mere key authentication is needed. But even with a CA, public-key misuse can lead to undetectable man-in-the-middle attacks (MITM), where the opponent reads the messages *without* having to break any cipher at all. With a secret key cipher at least the opponent has to actually break a cipher, which is thought to be hard.

Users can *always* give secret information to others. It is not the cryptography which allows exposure, since either end can give a copy to someone else no matter how the original was sent. The role of cryptography is limited to providing protected communication (or storage), and cannot prevent exposure by either user. Sharing secret information with someone else inherently implies a certain degree of trust.

In a secret-key cipher, a user at each end has exactly the same key. If only two users have a key, and one user receives a message which that key deciphers, the message can only have been sent by the other user who has that key. But there are various issues:

- **Can we afford for another user to have "our" key?** A secret key just represents a particular communications path. A user can easily manage thousands of keys, and create new ones at will. Keys are not a scarce resource.
- **Can we trust another user to have "our" key?** The whole point of having a key is to get secret information to that other user who we trust to have that information. Nobody cares about the key *per se*, the issue is what the key can expose. And if we trust the other user not to expose the information itself, what sense does it make to not trust them with the key whose main effect is simply to expose that information?
- **Can we trust another user to not expose "our" key?** The main thing exposing a key can do is to expose the secret messages. But the other user *already has* the messages, and could be sharing them, with or without sharing the key. Sharing the key has not changed the problem. There is nothing magic about a cipher system, whether secret key or public key, that changes the fact that we trust the other end with secret information, and they might not be worthy of that trust.
- **Suppose the other end is untrustworthy and shares the key with someone who pretends to be that user.** Even without sharing the key, the other user can accept outside messages and send them along, misleading us as to the source. Even without keys and cryptography, another party can present outside work as their own. The key is not the problem. The problem is whether we trust the other end, and if so, how much. Cryptography cannot solve that problem.

The remaining issue seems to be that, if everybody has to talk privately and independently of everyone else, then everybody needs a different key for everybody else. Public-key systems seem to make that easier by allowing the senders to share the public key to a single user. Ideally, fewer keys need be created. But actually *getting* public keys is only "easy" *after* a CA is established, funded, operating, and even then only if we can live with trusting a CA. Similar structures could be built to easily distribute secret keys, and may be particularly appropriate in a distributed, hierarchical business situation.

In practice, we do *not* need to talk to *everybody* else, just a small subset. And many interactions are with representatives from a common group, each of whom has access to the same secret information anyway. The group might need a different key for each "client" user, but everyone in the group could use the key for that client. Business groups might have to handle millions of keys, but public-key technology does not solve the problem, because *somebody* has to authenticate all those keys. If we hide that function in the CA, then we have to fund and trust the CA.

Suppose we need to communicate, privately and independently, with *n* people:

- With a secret-key cipher we need to transfer *n* different keys. This would happen in some inconvenient "out of band" way, such as a personal meeting, phone call, fax, letter, delivery service, or as a part of a business hierarchy. Usually this depends upon having some sort of prior or business relationship, with the usual introductions, meetings, and communication paths.
- With a public-key cipher we *also* need *n* keys (actually, *n* pairs). Since the keys are public, presumably they should be easy to get from a list on some server somewhere (provided somebody sets up and maintains such a server). But we cannot simply trust a web page since it may be hacked, nor can we trust email that may really be from an opponent. Each key must be *authenticated*, or we risk *vastly* easier MITM attacks exposing our messages. Key authentication can be conveniently done "in band," but only with a CA with whom we have already arranged trust. Otherwise we have to call the other guy on the phone, see if it is who we expect, and confirm a hash of the key, again, "out of band." And we cannot simply use a phone number or address from the web, since that may call the opponent instead. Absent a CA, the main advantage here seems to be that contents of the phone call (or letter or fax) need not be secret.

One other possibility is a "web of trust." In this structure, people attest to trusting someone who has a key for someone else. But even if we assume that could ever work to cryptographic levels of certainty, validating a key is only half the issue. The other part is whether we can reasonably hope to trust our secret information to someone we do not know. Public-key ciphers do not solve that problem.

An odd characteristic of public-key cryptography is that, normally, if we encipher a message to a particular user, we cannot then decipher that same message. In a business context that may be an auditing problem, since the business can only read and archive *incoming* messages. Absent a special design, public-key cryptography may make it impossible to document what offer was actually sent.

For those who would never consider using the short keys needed by secret-key ciphers, note that public keys must be much longer than private keys of the same strength, because a valid public key must have a very restricted form that most key values will not have. In practice, a public-key cipher almost certainly will just set up the keys for an internal secret-key cipher which actually protects the data, so the final key size will be small anyway.

Public-key technology is a tool that offers certain advantages, but those are not nearly as one-sided as people used to believe. Secret key cipher systems were functioning well in practice long before the invention of public-key technology.

Also see:

- Key Archives,
- Key Authentication,
- Key Distribution Problem,
- Key Loading,
- Key Loss,
- Key Management,
- Key Problems,
- Key Reuse,
- Key Selection,
- Key Storage,
- Key Transport and
- Keyspace.

Key Archives

Archival storage of keys. As usual, keys for secret key ciphers, and private keys for public key ciphers, all must be stored in encrypted form.

Keys should be changed periodically. But, in a corporate setting it is likely that the corporation will want to be able to review old messages which were encrypted under old keys. That either requires archiving a plaintext version of each message, or archiving the encrypted version, plus the key to decrypt it (see key storage).

Obviously, key archives could be a sensitive, high-value target, so that keeping keys and messages on different machines may be a reasonable precaution.

Key Authentication

Assurance that a key has not been changed or exchanged during delivery.

In secret key ciphers, key authentication is *inherent* in secure key distribution.

In public key ciphers, public keys can be exposed and delivered in unencrypted form. But someone who uses the wrong key may unknowingly have "secure" communications with an opponent, as in a man-in-the-middle attack. It is thus absolutely crucial that public keys be authenticated or certified as a separate process. Normally this implies the need for a Certification Authority (CA).

Also see: message authentication and user authentication.

Key Creation

The creation of a new key. Keys for secret key ciphers, and private keys for public key ciphers must be stored in encrypted form.

A corporation may seek to limit the ability for users to create their own new keys, so that corporate authorities can monitor all business communications. That of course implies that the corporation takes on the role of creating and distributing new keys, and probably also maintains a key archive as well as a message archive (see key storage).

Key Distribution Problem

The problem of distributing keys to both ends of a communication path, especially in the case of secret key ciphers, since secret keys must be transported and held in absolute secrecy. Also the problem of distributing vast numbers of keys, if each user is given a separate key. Also see: key reuse.

Although this problem is supposedly "solved" by the advent of the public key cipher, in fact, the necessary public key validation is almost as difficult as the original problem. Although public keys can be *exposed,* they must represent who they claim to represent, or a "spoofer" or man-in-the-middle can operate undetected.

Nor does it make sense to give each individual a separate secret key, when a related group of people would have access to the same files anyway. Typically, a particular group has the same secret key, which will of course be changed when any member leaves. Typically, each individual would have a secret key for each group with whom he or she associates.

Key Loading

Accepting a key and placing it in local key storage for later use. Keys for secret key ciphers, and private keys for public key ciphers, all must be transported and stored in encrypted form.

For public key ciphers, the key to be loaded will be in plaintext form and need not be deciphered. Similarly, a public key database may be unencrypted, since all the public keys are exposed anyway. So adding a public key can be just as simple as adding any other data.

For secret key ciphers, the key to be loaded will have been transported encrypted under some other key. And the user key database will be encrypted under the keyphrase for that particular user. Accordingly, a new key must first be decrypted and then encrypted under the user keyphrase. One problem here is that we want to minimize the amount of time any secret key exists in plaintext form. Of course keys will be in plaintext form during use, in which case we decipher the key only in program memory, and then zero that storage as soon as possible.

It seems desirable to avoid deciphering the entire key database simply to insert a single new key. One workable possibility is to add simple structure to the cipher itself so that cipherings can be concatenated *as ciphertext* (as long as they are ciphered under the same key). Then the new key can be enciphered on its own, and simply concatenated onto the key storage file.

Key Loss

The loss of the key necessary to decrypt encrypted messages, or needed to create new encrypted messages.

The more valuable the messages, the more serious the risk from loss of the associated keys. Such loss might occur by equipment failure, accident, or even deliberate user action.

In the business case, key loss can be mitigated by maintaining corporate key archives, and distributing key files to users. Without such archives (or some alternate way to recover), a single user equipment failure could result in the loss of critical keys and business documents, which would virtually guarantee the end of encryption in that environment.

Key Management

Typically, the facilities of a cipher system which support the use of keys, either for secret key ciphers or public key ciphers or both. All keys for secret key ciphers, and the secret private keys for public key ciphers, must be stored in encrypted form.

Key management facilities for users typically include:

- key loading (acquiring a key from someone),
- key storage (retaining keys for use),
- key selection (selecting the correct key),
- key re-use (minimizing security effects from using the same key for repeated messages),
- message archives (key management can support the archival storage of messages in their original encrypted form),
- key loss response ("My computer ate my key file! Now what?").

Additional key management facilities may be made available only to corporate security officers:

- key creation (creating a random key),
- key transport (moving a key to the users),
- key archives (supporting encrypted message archives)

Key Problems

Limits on the ability of cryptographic keys to solve security problems. Such limits are suggested by the well-known problems we have with the house keys we use everyday:

- We can lose our keys.
- We can forget which key is which.
- We can give a key to the wrong person.
- Somebody can steal a key.
- Somebody can pick the lock.
- Somebody can go through a window.
- Somebody can break down the door.
- Somebody can ask for entry, and unwisely be let in.
- Somebody can get a warrant, then legally do whatever is required.
- Somebody can burn down the house, thus making everything irrelevant.

Even absolutely perfect keys cannot solve all problems, nor can they guarantee privacy. Indeed, when cryptography is used for communications, generally at least two people know what is being communicated. So either party could reveal a secret:

- By accident.
- To someone else.
- Through third-party eavesdropping.
- As revenge, for actions real or imagined.
- For payment.
- Under duress.
- In testimony.

When it is substantially less costly to acquire the secret by means other then a technical attack on the cipher, cryptography has pretty much succeeded in doing what it can do. Unfortunately, once an attack has been found and implemented as a computer program, the incremental cost of applying that attack may be quite small.

Key Reuse

The ability of a cipher system to re-use a key without reduced security. One of the expected requirements for a cipher system.

Obviously, we may send many messages to a particular recipient. For security reasons we do not want to use the same key for any two messages, and we also do not want to manually establish that many keys. One approach is the two-stage message key, where a random value is used to encrypt the message, and only that random value is encrypted by the main key. In hybrid ciphers, a public key component transports the random message key. Thus, the stored keys are used only to encrypt a relatively small random value or nonce, and so are very well hidden.

One way to create the random nonce would be to bit-permute a vector of half-1's and half-0's, by shuffling twice. That way, even if the cipher failed and the nonce was exposed, the keyed generator creating the message keys would be protected (see Dynamic Transposition) so that the opponent will have to repeat the previous break, which may demand both time and luck.

Key Schedule

Typically, the modified key used in each round of an iterated block cipher. There is ample motive to make each key modification simple, so that each ciphering round can occur quickly, but that has resulted in strongly-related round keys which were targeted for attack.

Key Selection

A stored key is selected for use. Normally, we could not do this by inspection, because each key is a long random number, generally indistinguishable by humans from among all other long random numbers. In any case, we do not want to reveal it even to the user. Accordingly, the key storage system would typically include a name or nickname or email address field to identify the correct key.

Under my alias file implementation, selecting the right key for use is done by entering the alias nickname for the desired person, contract, project or group. That could be the email address for the appropriate channel; those with multiple email addresses could have the same key listed under multiple aliases. When the email address is used as the alias, the desired email address can be automatically found in the message header, and the correct key automatically used.

Similarly, stored keys should have a start date, and multiple keys for the same channel will be distinguished by that date. By checking the date of the message to be decrypted, the key which was correct as of that date could be automatically selected and used, again making most key selection automatic, even for archived messages.

In practice, it is common to select the wrong key, and then the message cannot be read. But if the message is just being accumulated somewhere for later use, we may mistakenly discard the original ciphertext before making sure we have the deciphered plaintext. Accordingly, a required feature of a cipher system is that using the wrong key be detected and announced. Presumably this will be done with some sort of error detecting code such as a CRC of the plaintext, although some cases may demand a keyed MAC.

Key Storage

Storage of keys for later use. Keys for secret key ciphers and private keys for public key ciphers must be stored in encrypted form.

For public keys, the key database can be unencrypted, and possibly even part of a larger database system.

For secret keys, the database must be encrypted, and should be as simple as possible for security reasons. One possible form is what I call an alias file. Each key is given a textual *alias*, which then becomes an efficient way to identify and use a particular long random key. Typically, an alias would be a nickname for a person, project, or work group, or an email address. By allowing the user to specify a short name instead of a key, long and random keys can be selected and used efficiently.

The "database" part of this could be as simple as an encrypted list of entries, with each entry having a few simple textual fields such as alias id, key value, and start date. By ordering the entries by start date, the system could search from the front of the list for the first entry matching the alias field and having a start date before the current date.

Although often honored in the breach, periodic key changes are a security requirement. We do of course assume that the cipher system will encrypt the data for each message with a random key in any case. But if everything always starts with the same key, then anyone getting that key will have access to everything, which makes that key an increasingly valuable target. To compartmentalize, and limit that risk, we must change keys periodically, even public and private keys.

A start date field supports periodic key change in a way largely invisible to the user. A routine corporate key file update would add new keys at the start of the file, with future start dates, thus not affecting the use of the current keys at all. Then, when the new date arrives, the new keys would be selected and used automatically, making the key update process largely invisible to the user and far more practical than usually thought possible.

In contrast, an end date seems much less useful, not the least because it involves a prediction of the future as to when the key may become a problem. Moreover, presumably the intended response to such a date is to stop key use, but if a new key has not been distributed, that also stops business operation, which is just not smart. So a start date is needed, but an end date is not.

Corporate key policies would produce new key files for users from time to time, with future keys added and unused keys stripped out. That also would be an appropriate time for the user to implement a new passphrase.

Also see message archives.

Key Transport

The movement of keys from creator to user. Keys for secret key ciphers and private keys for public key ciphers must be transported in encrypted form.

Public key transport may at first seem fairly easy. Public keys do not need to be encrypted for transport, and anyone may see them. However, they absolutely must be certified to be the exact same key the sender sent. For if someone replaces the sent key with another, subsequent messages can be exposed without breaking any cipher (see man in the middle attack). The usual solution suggested is a large, complex, and expensive certification infrastructure that is often ignored. (See PKI.)

Secret key transport first involves encrypting the secret key under a one-time keyphrase or random nonce. The resulting message is then hand-carried (on a floppy or CDR) or otherwise sent (perhaps by overnight mail or package courier) to the other end. Then the keyphrase is sent by a different channel (perhaps by phone or fax) to decrypt the key. Of course, if the encrypted message is intercepted and copied, and then the second channel intercepted as well, the secret key would be exposed, which is why hand-delivery is best. Fortunately, most people who are working together do meet occasionally and then keys can be exchanged. As soon as the transported secret key is decrypted it should immediately be re-encrypted for secure storage. That can and should be done without ever exposing the key itself.

Keyed Substitution

Two substitution tables of the same size with the same values can differ only in the ordering or permutation of the values in the tables. A huge keying potential exists: The typical "n-bit-wide" substitution table has 2^n elements, and $(2^n)!$ ("two to the nth factorial") different permutations or key possibilities. A single 8-bit substitution table has a keyspace of 1648 bits.

A substitution table is keyed by creating a *particular* ordering from each different key. This can be accomplished by shuffling the table under the control of a random number generator which is initialized from the key.

Keyphrase

A key, in the form of a human-friendly language phrase. In my designs, I usually hash a keyphrase into a key for an alias file, which holds the starting keys for each particular channel or target. Note that the hash can be a simple CRC, since both the keyphrase and the hash result have essentially the same security exposure. Also see: cipher system, password and user authentication.

Keyspace

The number of distinct key-selected transformations supported by a particular cipher. Normally described in terms of bits, as in the number of bits needed to count every distinct key. This is also the amount of state required to support a state value for each key. The keyspace in bits is the $\log_2$ (the base-2 logarithm) of the number of different keys, provided that all keys are equally probable.

Cryptography is based on the idea that if we have a huge number of keys, and select one at random, the opponents generally must search about half of the possible keys to find the correct one; this is a brute force attack.

Although brute force is not the only possible attack, it is the one attack which will always exist (except for ciphers with Perfect Secrecy). Therefore, the ability to resist a brute force attack is normally the design strength of a cipher. All other attacks should be made even more expensive. To make a brute force attack expensive, a cipher simply needs a keyspace large enough to resist such an attack. Of course, a brute force attack may use new computational technologies such as DNA or "molecular computation." Currently, 120 bits is large enough to prevent even unimaginably large uses of such new technology.

It is probably just as easy to build efficient ciphers which use huge keys as it is to build ciphers which use small keys, and the cost of storing huge keys is probably trivial. Thus, large keys may be useful when this leads to a better cipher design, perhaps with less key processing. Such keys, however, cannot be considered better at resisting a brute force attack than a 120-bit key, since 120 bits is already sufficient.

Keystream

The confusion sequence in a stream cipher. Also running key.

Keystroke

The transition of a key on a computer keyboard.

On a PC-style computer, a processor internal to the keyboard maintains the state of all keys (up or down). The keyboard processor also continuously scans through each possible key (perhaps every 2 msec) and reports to the PC any key which has just gone up or down. Keys are reported by position or "scan code;" the keyboard processor does not use ASCII.

Measuring keystroke timings is a common way of collecting supposedly unknowable information for a really random generator. However, even though a PC computer can measure events very closely, the keyboard scan process inherently quantizes keystrokes at a far more coarse resolution. And, when measuring software-detected events, various PC system things like hardware interrupts and OS task changes can provide substantial variable latency which is nevertheless deterministic.

Known Plaintext

The information condition of an opponent knowing both the plaintext and the related ciphertext for an encryption. We normally assume that a large amount of known plaintext is available to an opponent for use in an attack. See: known plaintext attack, ciphertext only and defined plaintext, a more-powerful case of known plaintext. Also see the "Known-Plaintext and Compression" conversation (locally, or @: http://www.ciphersbyritter.com/NEWS6/KNOWNPLN.HTM).

One model of a cipher is a key-selected mathematical function or transformation between plaintext and ciphertext. To an opponent this function is unknown, and one of the best ways to address an unknown function is to look at both the input and output. More than that, even though an opponent has ciphertext, *something* must be known about the plaintext or an opponent has no way to measure attack success.

Public key ciphers allow opponents to create known-plaintext. Thus, public key ciphers force us to *assume* they will resist known-plaintext attacks, even though that may or may not be correct. However, most so-called "public key" ciphers do not protect actual data with a public key system, but are in fact hybrid ciphers, where the public key system is used only to transfer a key for a conventional secret key cipher.

The cryptanalytic literature on secret-key ciphers is rife with attacks which depend upon known-plaintext, and secret-key ciphers are still used for almost all data ciphering. Virtually all secret-key ciphers are *best* attacked with known-plaintext to the point that describing cipher weakness almost universally means some number of cipherings and some amount of known-plaintext. For example, Linear Cryptanalysis normally requires known-plaintext, while Differential Cryptanalysis generally requires the even more restrictive defined plaintext condition.

If modern cipher designers do not talk much about known-plaintext, that may be because designers think that:

- not much can be done about it;
- since typically only 1 or 2 blocks are sufficient to define a particular key for a conventional block cipher, that tiny amount of exposure cannot be prevented;
- the current best attacks require huge amounts of known-plaintext, amounts which in practice are unlikely to be available anyway;
- since modern ciphers are designed with the *goal* of having no weakness under known-plaintext conditions, designers may assume without proof that the goal has been achieved.

On the other hand, *some aspect* of the plaintext *must* be known, or it will be impossible to know when success has been achieved. Consequently, it is hard to imagine a situation in which actual known-plaintext would not benefit cryptanalysis. Since huge amounts of known-plaintext are needed for current attacks, that much exposure may be preventable at the cipher system level. And, since attacks only get better over time it would seem only prudent to hide as much known-plaintext as possible.

It is surprisingly reasonable that an opponent might have a modest amount of known plaintext and the related ciphertext: That might be the return address on a letter, a known report or newspaper account, or even just some suspected words. Sometimes a cryptosystem will carry unauthorized messages such as birthday greetings which are then discarded in ordinary trash, due to their apparently innocuous content, thus potentially providing a small known-plaintext example. (It is harder to see how really huge amounts of known-plaintext might escape, but one possibility is described in security through obscurity.)

Unless the opponents know *something* about the plaintext, they will be unable to distinguish the correct deciphering even when it occurs. Hiding all structure in the plaintext thus has the potential to protect messages against even brute force attack; this is essentially a form of Shannon-style Ideal Secrecy.

One approach to making plaintext "unknown" would be to pre-cipher the plaintext, thus hopefully producing an unstructured ciphertext which would prevent success when attacking the second cipher. In fact, each cipher would protect the other. Successful attacks would then have to step through *both* ciphering keys instead of just one, which should be exponentially more difficult. This is one reason for using multiple encryption. Also see the known plaintext discussion "Known-Plaintext and Compression" (locally, or @: http://www.ciphersbyritter.com/NEWS6/KNOWNPLN.HTM).

Known Plaintext Attack

Any attack which takes place under known plaintext information conditions. Clearly, when the opponent already knows both the plaintext and the associated ciphertext, the only thing left to find is the key.

A known plaintext attack typically needs both the plaintext value sent to the internal cipher and the resulting ciphertext. Typically, a large amount of plaintext is needed under a single key. A cipher system which prevents any one of the necessary conditions also stops the corresponding attacks.

Known plaintext attacks can be opposed by:

- minimizing the exposure of exact message plaintext (although, realistically, only so much can be done),
- limiting the amount of data enciphered under one key,
- changing keys frequently,
- using multiple encryption and so hiding the actual plaintext or ciphertext or both for each internal cipher, and
- enciphering each message under a different random message key which users cannot control. (And that is something we have every right to expect will be done by even the most basic cipher system.)

Although known plaintext *per se* is not always needed to attack a cipher, *some aspect* of knowledge about the plaintext is absolutely *required* to know when any attack has succeeded (see ciphertext only attack). Also see defined plaintext attack.

A known plaintext attack is especially dangerous to the conventional stream cipher with an additive combiner, because the known plaintext can be "subtracted" from the ciphertext, thus completely exposing the confusion sequence. This is the sequence produced by the cryptographic random number generator, and can be used to attack that generator. Such attacks can be complicated or defeated by ciphers using a nonlinear combiner, like a keyed Latin square, or a Dynamic Substitution Combiner instead of the usual additive combiner.

Kolmogorov-Chaitin Complexity

(Algorithmic complexity.) The size of the smallest program which can produce a given sequence.

Unfortunately, different abilities to sense and use deep structure or correlations in a sequence can make major differences in the complexity value. In general, we cannot know if we have found the smallest program.

One of many possible measures of sequence complexity; also see: linear complexity and entropy.

Kolmogorov-Smirnov

In statistics, a goodness of fit test used to compare two distributions of ordinal data, where measurements may be re-arranged and placed in order. Also see chi-square.

- *n* random samples are collected and arranged in numerical order in array *X* as *x*[0]..*x*[*n*-1].
- *S*(*x*[*j*]) is the fraction of the *n* observations which are less than or equal to *x*[*j*]; in the ordered array this is just ((*j*+1)/*n*).
- *F*(*x*) is the reference cumulative distribution, the probability that a random value will be less than or equal to *x*. Here we want *F*(*x*[*j*]), the fraction of the distribution to the left of *x*[*j*] which is a value from the array.

There are actually at least three different K-S statistics, and two different distributions:

The one-sided statistics are:

$$Dn^{+} = \mathrm{MAX}(\ S(x[j]) - F(x[j])\ ) = \mathrm{MAX}(\ ((j+1)/n) - F(x[j])\ )$$

```
   Dn- = MAX( F(x[j]) - S(x[j]) )
       = MAX( F(x[j]) - (j/n) )
```

where "MAX" is computed over all *j* from 0 to *n*-1. Both of these statistics have the same distribution.

The two-sided K-S statistic is:

```
   Dn* = MAX( ABS( S(x[j]) - F(x[j]) ) )
       = MAX( Dn+, Dn- )
```

and this has a somewhat *different* distribution.

What's the Difference?

This "side" terminology is standard but unfortunate, because *every* distribution has two sides which we call "tails." And *every* distribution can be *interpreted* in one-tailed or two-tailed ways.

Here, the "side" terminology refers to different statistic computations: the "highest" difference, the "lowest" difference, and the "greatest" difference between two distributions. Thus, "side" refers *not* to the tails of a distribution, but to values being either *above* or *below* the reference. If we knew that only one direction was news, we could use the appropriate "one-sided" statistic, and still interpret the results in a "two-tailed" way.

For example, if we used the "highest" test, we could detect large positive differences from the reference distribution (if the p-value was near 1.0) and also detect a distribution which was unusually close to the reference (if the p-value was near 0.0). Obviously, the "highest" test would hide negative differences.

On the other hand, if we compute *both* the "highest" *and* "lowest" statistics, we cover all the information (and slightly more) that we could get from the "two-sided" or "greatest" statistic.

We can base a hypothesis test of any statistic results on critical values on either or both tails of the null distribution, depending on our concerns. One problem with a "two-tailed" interpretation is that we accumulate critical region on both ends of the distribution, even though the ends are not equally important.

What Does the P-Value Mean?

Normally, the p-value we get from a statistic comparing distributions is the probability of that value occurring when both distributions are the same. Finding p-values near zero or one is odd. Repeatedly finding p-values too close to zero shows that the distributions are unreasonably similar. Repeatedly finding p-values too close to one shows that the distributions are different. And we do not need critical-value trip-points to highlight this.

The Knuth Versions

Knuth II multiplies Dn^+, Dn^- and Dn* by SQRT(*n*) and calls them Kn^+, Kn^- and Kn*, so we might say that there are at least *six* different K-S statistics.

The one-sided K-S distribution is easier to compute precisely, especially for small *n* and across a wide range (such as "quartile" 25, 50, and 75 percent values), and may be preferred on that basis. There is a modern evaluation for the two-sided K-S distribution which should be better than the old versions, but usually we do not need to accept its limitations. Often the experimenter can choose to use tests which are more easily evaluated, and for K-S, that would be the "one-sided" tests.

See the Kolmogorov-Smirnov section of "Normal, Chi-Square and Kolmogorov-Smirnov Statistics Functions in JavaScript" page (locally, or @: http://www.ciphersbyritter.com/JAVASCRP/NORMCHIK.HTM#KolSmir).

Latency

A delay; the time needed to perform an operation. Often a hardware issue, but sometimes also important in organizing system-level software. There is often a systems-level tradeoff between latency and throughput (data rate).

Consider three cold slices of pizza: We might put all three on a plate and heat them in a microwave in 3 minutes. That would be 3 minutes from thought to mouth, a 3-minute eating latency. But if we put just one slice on a plate, we can heat that slice in 1 minute, for an eating latency of just 1 minute. So we can be eating fully three times as soon, just by making the appropriate choice, but then we can only eat one a minute. System design is often about making such choices.

Or consider a mixing cipher, which typically needs log n mixing sub-layers to mix n elements (i.e., n log n operations). The latency is the delay from the time we start computing until we get the result. So if we double the number of elements, we also double the necessary computation, *plus* another sub-layer. Thus, in software, when we double the block size, the latency increases somewhat, and the data rate decreases somewhat (but see huge block cipher advantages).

In hardware, things are much different: Even though the larger computation is still needed, that can be provided in separate on-chip hardware for each sub-layer. Typically, each sub-layer may take a single clock cycle to perform the computation. So if we double the block size, we need another sub-layer, and do gain one more clock cycle of latency before a particular block pops out. But the data rate is still a full block per cycle, and stays that, no matter how wide the block may be. In hardware, when we double the block size, we double the data rate, giving large blocks a serious advantage.

In the past, hardware operation delay has largely been dominated by the time taken for gate switching transistors to turn on and off. Currently, operation delay is more often dominated by the time it takes to transport the electrical signals to and from gates on long, thin conductors.

The effect of latency on throughput can often be reduced by pipelining or partitioning the main operation into many small sub-operations, and running each of those *in parallel,* or at the same time. As each operation finishes, that result is latched and saved temporarily, pending the availability of the next sub-operation hardware. Thus, throughput is limited only by the longest sub-operation instead of the overall computation.

Latin Square

A Latin square of order *n* is an *n* by *n* array containing symbols from some alphabet of size *n*, arranged such that each symbol appears exactly once in each row and exactly once in each column.

```
2  0  1  3
1  3  0  2
0  2  3  1
3  1  2  0
```

Since each row contains the same symbols, every possible row can be created by re-arranging or permuting the *n* symbols into *n*! possible rows. At order 4 there are 24 possible rows. A naive way to build a Latin square would be to choose each row of the square from among the possible rows, and that way we can build $(n!)^n$ different squares. At order 4, we can build 331,776 squares, but only 576 of those (about 0.17 percent) have the Latin square form, and things get exponentially worse at higher orders.

Cyclic Squares

The following square is *cyclic*, in the sense that each row below the top is a rotated version of the row above it:

```
    0   1   2   3
    1   2   3   0
    2   3   0   1
    3   0   1   2
```

This is a common way to produce Latin squares, but is generally undesirable for cryptography, since the resulting squares are few and predictable.

Other Constructions

It is at least as easy to make a more general Latin square as it is to construct an algebraic group: The operation table of any finite group is a Latin square, as is the addition table of a finite field. Conversely, while some Latin squares do represent associative operations and can form a group, most Latin squares do not. At order 4 there are 576 Latin squares, but only 16 are associative (about 2.8 percent). So non-associative (and thus non-group) squares dominate heavily, and may be somewhat more desirable for cryptography anyway.

Standard Form

A Latin square is said to be *reduced* or *normalized* or *in standard form* when the symbols are in lexicographic order across the top row and down the leftmost column. Any Latin square of any order reduces through a single rows and columns re-arrangement into exactly one standard square.

A Latin square is reduced to standard form in two steps: First, the columns are re-arranged so the top row is in order. Since that places the first element of the leftmost column in standard position, only the rows *below the top row* need be re-arranged to put the leftmost column in order. (Alternately, the rows can be re-arranged first and then the columns *to the right of the leftmost column*; the result is the same standard square.)

The 576 unique Latin squares of order 4 include exactly 4 squares in standard form:

```
    0 1 2 3      0 1 2 3      0 1 2 3      0 1 2 3
    1 2 3 0      1 3 0 2      1 0 3 2      1 0 3 2
    2 3 0 1      2 0 3 1      2 3 1 0      2 3 0 1
    3 0 1 2      3 2 1 0      3 2 0 1      3 2 1 0
```

Expanding Standard Latin Squares

A Latin square of order *n* can be shuffled or *expanded* into, and so can represent, *n*!(*n* - 1)! different squares. This is accomplished by permuting the *n* - 1 rows below the top row in (*n* - 1)! ways, and then permuting *n* columns in *n*! ways. Clearly, this reverses the process that will reduce any of those squares into the original standard square. However, in practice, apparently all the columns can be permuted first and the rows (less one) permuted next, or both at the same time, or even all the rows with the columns less one, each case apparently producing exactly the same set of permuted or expanded squares. (We can also permute the *n* symbols in *n*! ways, but this will produce no new squares.)

At order 4, by permuting 4 rows and 3 columns, each standard square expands to 4! * 3! = 24 * 6 = 144 permuted squares. Instead permuting 4 columns and 3 rows of the same standard square produces exactly the same set of permuted squares. Exactly one of each permuted set of 144 is in standard form, and that is the original standard square. Each of the squares expanded from the 4 standard squares is unique, and the accumulation of 4 * 144 = 576 permuted squares includes every possible order 4 Latin square exactly once. Thus we see the value of standard form, where 4 reduced squares represent 576.

Also see Latin square combiner and orthogonal Latin squares.

And see:

- "Latin Squares: A Literature Survey," locally, or @: http://www.ciphersbyritter.com/RES/LATSQ.HTM
- a Latin square conversation, locally, or @: http://www.ciphersbyritter.com/NEWS5/LATSQCMB.HTM
- my article: "Practical Latin Square Combiners," locally or @: http://www.ciphersbyritter.com/ARTS/PRACTLAT.HTM

for the fast checkerboard construction.

Latin Square Combiner

A cryptographic combining mechanism in which one input selects a column and the other input selects a row in an existing Latin square; the value of the selected element is the combiner result.

A Latin square combiner produces a balanced and nonlinear, yet reversible, combining of two values. The advantage is the ability to have a huge number of different, yet essentially equivalent, combining functions, thus preventing opponents from knowing which combining function is in use. To exploit this advantage, we must both create and use keyed Ls's. One efficient way to do that is what I call the checkerboard construction, which I also describe in my article: "Practical Latin Square Combiners" (locally or @: http://www.ciphersbyritter.com/ARTS/PRACTLAT.HTM). Also see Latin square and orthogonal Latin squares for other articles.

A Latin square combiner can be seen as the generalization of the exclusive-OR mixing concept from exactly two values (a bit of either 0 or 1) to any number of different values (e.g., bytes). A Latin square combiner is inherently balanced, because for any particular value of one input, the other input can produce any possible output value. A Latin square can be treated as an array of substitution tables, each of which is invertible, and so can be reversed for use in a suitable extractor. As usual with cryptographic combiners (including XOR), if we know the output and a specific one of the inputs, we can extract the value of the other input.

For example, a tiny Latin square combiner might combine two 2-bit values each having the range zero to three (0..3). That Latin square would contain four different symbols (here 0, 1, 2, and 3), and thus be a square of order 4:

```
   2  0  1  3
   1  3  0  2
   0  2  3  1
   3  1  2  0
```

With this square we can combine the values 0 and 2 by selecting the top row (row 0) and the third column (column 2) and returning the value 1.

When extracting, we will know a specific one (but only one) of the two input values, and the result value. Suppose we know that row 0 was selected during combining, and that the output was 1: We can check for the value 1 in each column at row 0 and find column 2, but this involves searching through all columns. We can avoid this overhead by creating the row-inverse of the original Latin square (the inverse of each row), in the well-known way we would create the inverse of any invertible substitution. For example, in row 0 of the original square, selection 0 is the value 2, so, in the row-inverse square, selection 2 should be the value 0, and so on:

```
   1  2  0  3
   2  0  3  1
   0  3  1  2
   3  1  2  0
```

Then, knowing we are in row 0, the value 1 is used to select the second column, returning the unknown original value of 2.

A practical Latin square combiner might combine two bytes, and thus be a square of order 256, with 65,536 byte entries. In such a square, each 256-element column and each 256-element row would contain each of the values from 0 through 255 exactly once.

Law

1. The body of rules intended to limit the conduct of man.
2. In Science, the rules of thumb consistent with the current scientific models. It is extremely rare that these are actual, proven limits on the conduct of reality.

Layer

In the context of block cipher design, a layer is particular transformation or set of operations applied across the block. In general, a layer is applied once, and different layers have different transformations. As opposed to rounds, where a single transformation is repeated in each round.

Layers can be confusion layers (which simply change the block value), diffusion layers (which propagate changes across the block in at least one direction) or both. In some cases it is useful to do multiple operations as a single layer to avoid the need for internal temporary storage blocks.

LC

1. In cryptography, Linear Cryptanalysis.
2. In electronics, a circuit consisting of inductance (L), and capacitance (C). A resonant circuit.

Lemma

A theorem used in the proof of another theorem. Also see: corollary.

Letter Frequencies

The number of times a letter occurs in a body of text or set of messages. Often of interest in classical cryptanalysis. Letter frequencies vary widely depending on the kind of writing used, so there is no one right answer. One good source is:

> Kullback,S. 1938. *Statistical Methods in Cryptanalysis*. (Reprinted by Aegean Park Press.)

but the following table is original for the Crypto Glossary.

At the top of the table are letters in a general rank ordering, most-common at the left and least-common at the right. Some of the variation possible in different sets of messages or text is shown by the different ranks a given letter may have. From this we might conclude that N, R, O, A, I should be treated as a group, while S may (or may not) be unique enough to identify specifically from the usage rank in a message.

```
   E T N R O A I S D L H C P F U M Y G W V B X Q K J Z
1  1 | | | | | | | | | | | | | | | | | | | | | | | | |
2 .. 2 | | | | | | | | | | | | | | | | | | | | | | | |
3 .... 3 3 | | | | | | | | | | | | | | | | | | | | | |
4 .... 4 4 4 4 | | | | | | | | | | | | | | | | | | | |
5 .....5 5 5 5 5 | | | | | | | | | | | | | | | | | | |
6 ...... 6 6 6 6 | | | | | | | | | | | | | | | | | | |
7 .......7 7 7 7 | | | | | | | | | | | | | | | | | | |
8 .............. 8 | | | | | | | | | | | | | | | | | |
9 ................ 9 | 9 | | | | | | | | | | | | | | |
10 ................. 10 0 | | | | | | | | | | | | | | |
11 ................. 11 1 1 | | | | | | | | | | | | | |
12 ................... 12 2 2 | | | | | | | | | | | | |
13 ..................... 13 3 3 3 | | | | | | | | | | |
14 ..................... 14 4 4 4 4 | | | | | | | | | |
15 ..................... 15 5 5 5 5 | | | | | | | | | |
16 ..........................16. 16 | | | | | | | | | |
17 ............................... 17 7 | | | | | | | |
18 ............................... 18 8 8 8 | | | | | |
19 ............................... 19 9 9 9 | | | | | |
20 ............................... 20 0 0 0 0 | | | | |
21 ..................................... 21 1 | | | | |
22 ......................................... 22 2 2 | |
23 ......................................... 23 3 3 3 |
24 ......................................... 24 4 4 4 |
25 ........................................... 25 5 5 5
26 ........................................... 26 ... 6
```

LFSR

Linear Feedback Shift Register.

Lie

In argumentation:
1. A falsehood, knowingly told. Possibly spin.
2. A falsehood, unknowingly told, when the teller had the responsibility to investigate the topic and did not. As distinct from an error based on sufficient research. (See belief.)
3. A falsehood implied, or a false impression allowed to exist by silence.

In scientific argumentation honesty is demanded. Lies cannot further the cause of scientific insight and conclusion. However, lies can waste the time of everyone involved, not just for the time of the discussion, but potentially years of effort by many people, based on false assumptions.

Personally, I take an accusation of lying very seriously. Absent a public apology, my response is to end my interaction with that person. That is what I do, and that is what I think everyone should do.

Some people think that failing to defend against even the most heinous assertion is a sign of weakness or even an admission of guilt. But when a person is accused of lying, the accused immediately knows whether the accusation is correct or not. And if not, the accuser has just shown *themselves* as a liar or a fool. There is no need to separate these possibilities. In either case there is no reason to further dignify whatever points they wish to present.

A participant in a discussion has no responsibility to respond, no matter what claim an opponent may make. Instead, is the responsibility of the claimant to present logical arguments or proof, as opposed to a mere possibility or belief. Simply making a claim then demanding that it be accepted if it cannot be proven false is the *ad ignorantium* fallacy.

LIFO

Last In, First Out. A stack. A form of queue which accumulates values as they occur and holds them until they are consumed in reverse order. As opposed to FIFO.

Linear

Literally, "like a line" or "resembling a line."
1. The description of an electronic circuit or amplifier in which a plot of input voltage x versus output voltage $y = f(x)$, over a limited range, is approximately a straight line, $y = ax$. In such a circuit, a known percentage change in the input produces a similar percentage change in the output.
2. The description of an electronic circuit or data transmission system in which independent signals do not

interact, or interact only weakly. In a linear system, the output result of one signal does not change when other signals are added, nor are spurious signals created.
3. A very simple mathematical function. For example, the geometric equation *y = f(x) = ax + b*.
4. A linear equation.

Suppose we have a cipher to analyze which has some unknown internal function: If that function is random, with no known pattern between input and output or between values, we may have to somehow traverse every possible input value before we can understand the full function.

But if there *is* a simple pattern in the function values, then we may only need a few values to predict the entire function. And a linear function is just about the simplest possible pattern, which makes it almost the weakest possible function.

There are at least two different exploitable characteristics of cryptographic linearity:

- Mathematical manipulation: Unknown values on one side of a linear function can be seen as simply modified values on the other side, thus avoiding the function entirely. Or unknowns might be solved by simultaneous equations and other techniques of linear algebra. These techniques more or less require exact mathematical linearity.
- Predictability: If we are willing to accept some error, a supposedly complex function might be modeled by a much simpler function which is "often" correct. Alternately, an unknown function might be exposed by a relatively small amount of data, even if the function is only "approximately" linear. (Also see rule of thumb.) Neither approach requires full mathematical linearity.

Technical Definitions of Linearity

One math definition of linearity is: function *f* : G -> G is *linear* in field G if:

```
   f(a*x + b*y) = a*f(x) + b*f(y)
```

for any a, b, x, y in G. To be linear, function *f* is thus usually limited to the restrictive form:

```
   f(x) = ax
```

that is, "multiplication" only, with no "additive" term. Functions which *do* an additive term; that is, in the form:

```
   f(x) =  ax + b
```

are thus technically distinguished and called affine. Affine functions are virtually as weak as linear functions, and it is very common to casually call them "linear."

Another definition of linearity is:

```
   1) f(0) = 0
   2) f(ax) = a * f(x)
   3) f(a + b) = f(a) + f(b)
```

where (1) apparently distinguishes linear from affine.

It is also possible to talk about linearity with respect to an "additive group." A function *f*: G -> H is linear in groups G and H (which have addition as the group operation) if:

```
   f(x + y) = f(x) + f(y)
```

for any x,y in G.

Uniqueness and Implications of Linearity

There are multiple ways a relationship can be linear: One way is to consider *a, x,* and *b* as integers. But the exact same bit-for-bit identical values also can be considered polynomial elements of GF(2^n). Integer addition and multiplication *is* linear in the integers, but when seen as mod 2 operations, the exact same computation producing the exact same results is *not* linear. In this sense, linearity is contextual.

Moreover, in cryptography, the issue may not be as much one of strict mathematical linearity as it is the "distance" between a function and some linear approximation (see rule of thumb and Boolean function nonlinearity). Even if a function is not technically linear, it may well be "close enough" to linear to be very weak in practice. So even a mathematical proof that a function could not be considered linear under any possible field would not really address the problem of linear weakness. A function can be very weak even if technically nonlinear.

True linear functions are used in ciphers because they are easy and fast to compute, but they are also exceedingly weak. Of course XOR is linear and trivial, yet is used all the time in arguably strong ciphers; linearity only implies weakness when an attack can exploit that linearity. Clearly, a conventional block cipher design using linear components must have nonlinear components to provide strength, but linearity, when part of a larger system, does not necessarily imply weakness. In particular, see Dynamic Transposition, which ciphers by permutation. In general, there is a linear algebra of permutations, but that seems to be not particularly useful when a different permutation is used on every block, and when the particular permutation used cannot be identified externally.

Linear Complexity

The length of the shortest linear feedback shift register (LFSR) which can produce a given sequence. The algorithm for finding this length is called Berlekamp-Massey.

One of many possible complexity measures; also see Kolmogorov-Chaitin complexity and entropy.

Also see my article: "Linear Complexity: A Literature Survey," locally, or @: http://www.ciphersbyritter.com/RES/LINCOMPL.HTM.

Linear Cryptanalysis

(LC). A form of attack in which linear equations, based on plaintext bits and ciphertext bits, show a slight bias for a particular key bit. These approximations are identified by human analysis of the cipher internal structure, and especially S-boxes. Also see Differential Cryptanalysis.

When the LC approximation equations include both plaintext and ciphertext bits, they obviously require at least known plaintext for evaluation, with sufficient data to exploit the usually tiny bias. LC typically also requires knowing the contents of the internal S-boxes to establish LC approximations.

Accordingly, most LC attacks are prevented by the simple use of keyed S-boxes.

LC attacks also can be addressed at the cipher system level, by:

- Limiting the amount of data encrypted under the same key, and,
- Using multiple encryption.

While these may seem like overkill for the simple purpose of addressing Linear Cryptanalysis, if they are to be part of the cipher or cipher system anyway, LC is pretty much out of the picture without added cost or analysis.

Moreover, these are clear, understandable and believable ways to address apparently unlimited anxieties about LC attacks.

Also see: "Linear Cryptanalysis: A Literature Survey," locally, or @: http://www.ciphersbyritter.com/RES/LINANA.HTM.

Linear Equation

An equation of one or more terms, each term being a constant factor of a single variable to the first power only:

$$y = a_1x_1 + a_2x_2 + . . . + a_nx_n$$

A first degree equation, such as: ax + b.

Linear Factor

In a mathematical expression, a factor which contains an unknown to the first power only: ax + b.

Linear Feedback Shift Register

LFSR. Basically a shift register with feedback, thus causing a sequence to be produced. An especially efficient design, often used in random number generator applications. Also used for scramblers.

In an *n*-element shift register (SR), if the last element is connected to the first element, a set of *n* values can circulate around the SR in *n* steps. But if the values in two of the elements are combined by exclusive-OR and that result connected to the first element, it is possible to get an almost-perfect maximal length sequence of 2^n-1 steps. (The all-zeros state will produce another all-zeros state, and so the system will "lock up" in a degenerate cycle.) Because there are only 2^n different states of *n* binary values, every state value but one must occur exactly once, which is a statistically-satisfying result. Moreover, the values so produced are a perfect permutation of the counting numbers (1..2^n-1).

```
A Linear Feedback Shift Register

    +-----+    +-----+    +-----+    +-----+    +-----+  "a0"
+-<-| a5  |<---| a4  |<-*-| a3  |<---| a2  |<---| a1  |<--+
|   +-----+    +-----+  | +-----+    +-----+    +-----+   |
|                       v                                 |
+------------------->  (+)  ------------------------------+

  1          0          1          0          0          1
```

In the figure we have a LFSR of degree 5, consisting of 5 storage elements a[5]..a[1] and the feedback computation a[0]=a[5]+a[3]. The stored values may be bits and the operation (+) addition mod 2. A clock edge will simultaneously shift all elements left, and load element a[1] with the feedback result as it was before the clock changed the register. Each SR element is just a time-delayed replica of the element before it, and here the element subscript conveniently corresponds to the delay. We can describe this logically:

```
a[1][t+1] = a[5][t] + a[3][t];
a[2][t+1] = a[1][t];
a[3][t+1] = a[2][t];
a[4][t+1] = a[3][t];
a[5][t+1] = a[4][t];
```

Normally the time distinction is ignored, and we can write more generally, for some feedback polynomial C and state polynomial A of degree *n*:

```
        n
a[0] = SUM  c[i]*a[i]
```

```
          i=1
```

The feedback polynomial shown here is 101001, a degree-5 poly running from c[5]..c[0] which is also irreducible. Since we have degree 5 which is a Mersenne prime, C is also primitive. So C produces a maximal length sequence of exactly 31 steps, provided only that A is not initialized as zero. Whenever C is irreducible, the reversed polynomial (here 100101) is also irreducible, and will also produce a maximal length sequence.

LFSR's are often used to generate the confusion sequence for stream ciphers, but this is very dangerous: LFSR's are inherently linear and thus weak. Knowledge of the feedback polynomial and only *n* element values (from known plaintext) is sufficient to run the sequence backward or forward. And knowledge of only 2*n* elements is sufficient to develop an unknown feedback polynomial (see: Berlekamp-Massey). This means that LFSR's should not be used as stream ciphers without in some way isolating the sequence from analysis. Also see jitterizer and additive RNG.

Linear Logic Function

A Boolean switching or logic function which can be realized using only XOR and AND types of functions, which correspond to addition mod 2 and multiplication mod 2, respectively.

Log$_2$

The base-2 logarithm function. Generally used to measure information and keyspace in bits. Easily computed using the logarithm operations available on most scientific calculators:

$$\log_2(x) = \log_{10}(x) / \log_{10}(2)$$

$$\log_2(x) = \ln(x) / \ln(2)$$

Logic

1. electronic devices which realize symbolic logic, such as Boolean logic, the TRUE and FALSE values used in digital computation. Also see logic function.
2. A branch of philosophy related to distinguishing between correct and incorrect reasoning. The science of correct reasoning. The basis of mathematics and arithmetic.

Since math is based on logic, it is not supposed to be possible for math to support fuzzy or incorrect reasoning. However, math does exactly that in practical cryptography. (See some examples at proof and old wives' tale.) The problem seems to be a false assumption that math *is* cryptography, so whatever math proves must apply in practice. But math only *is* cryptography in theoretical systems for theoretical data; in *real* systems, the necessary assumptions can almost never be guaranteed, which means the conclusions are no longer proven. It is logically invalid (and extremely dangerous) to imagine that unproven conclusions can provide confidence in a real system.

The science of logic is intended to force reasoning into patterns which always produce a correct conclusion from initial assumptions. Of course, even an invalid argument can sometimes produce a correct conclusion, which can deceive us into thinking the argument is valid. A *valid* argument must *always* produce a correct conclusion. See: subjective, objective, contextual, absolute, inductive reasoning, deductive reasoning, and especially fallacy. Also see: argumentation, premise and conclusion; scientific model, hypothesis and null hypothesis.

Logic Fallacy

See fallacy.

Logic Function

Fundamental digial logic operations. The fundamental two-input (dyadic) one-output Boolean functions are

AND and OR. The fundamental one-input (monadic) one-output operation is NOT (or inversion). These can be used in various ways to build exclusive-OR (XOR), which is also widely used as a fundamental function. Here we show the truth tables for the fundamental functions:

INPUT	NOT
0	1
1	0

INPUT	AND	OR	XOR
0 0	0	0	0
0 1	0	1	1
1 0	0	1	1
1 1	1	1	0

These Boolean values can be stored as a bit, and can be associated with 0 or 1, FALSE or TRUE, NO or YES, etc. Also see: gate and DeMorgan's Laws.

Logic Level

The voltage range used by electronic digital logic devices to represent binary or Boolean bit values. Every digital logic device is built from analog amplifiers, and can only guarantee valid digital results when every input is in a valid voltage range.

In TTL-compatible devices, a logic zero input must be 0.8 volts or lower, and a logic one input must be 2.0 volts or higher. That leaves the range between 0.8 and 2.0 volts as *invalid.* When a logic device has an invalid voltage on some input, it is not guaranteed to perform the expected digital function. In particular, attempts to latch invalid voltage levels can lead to metastability problems.

Note that different logic families have different valid signal ranges:

	74	L	H	S	LS	AS	ALS	F	HC	HCT	AC	ACT
VOH	2.4	2.4	2.4	2.7	2.7	2.5	2.5	2.5	3.8	4.3	4.4	4.4
VIH	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	3.2	2.0	3.2	2.0
VIL	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.9	0.8	1.4	0.8
VOL	0.4	0.3	0.4	0.5	0.4	0.5	0.4	0.5	0.3	0.3	0.1	0.1
Clk	35	3	50	100	50	125	35	150	24	25	155	160

```
Output High, Input High, Input Low, Output Low; Clk = max. FF toggle MHz
NOTE: different texts and vendors list different values
```

These are component specifications that characterize the situation of a logic output pin connected to logic input pins. Assuming that the supply voltage, ambient temperature, loading and time delays are all within specified limits, the output voltage is guaranteed to be either *higher* than V_{OH} (for a one), or *lower* than V_{OL} (for a zero). These output values are *beyond* the levels needed for inputs to sense a particular logic level (either V_{IH} or V_{IL}). As a result, a system can have 300 or 400 mV of noise (from the power supply, ground loops, inductive and capacitive pickup) yet still sense the correct logic value. In this way, large digital systems can be built to perform reliably. (Also see: system design.)

lsb

Least-significant bit. In a binary representation, the bit of lowest significance. Often the rightmost bit. As opposed to lsB. Also see msb and msB.

lsB

Least-significant byte. In a binary representation, the byte of lowest significance. Often the rightmost byte. As opposed to lsb. Also see msb and msB.

MAC

Mesage Authentication Code. A keyed hash function for message authentication.

Machine Language

Also "machine code." A computer program in the form of the numeric values or "operation codes" (opcodes) which the computer can directly execute as instructions, commands, or "orders." Thus, the very public code associated with the instructions available in a particular computer. Also the programming of a computer at the bit or hexadecimal level, below even assembly language. Also see source code and object code.

Magnetic Field

The fundamental physical force resulting from moving electric charges. Also see: electromagnetic field.

Man-in-the-Middle Attack

(MITM attack.) A form of attack in which an opponent can intercept ciphertext, manipulate it, then send it to the far end. The point is either to expose the traffic, or perhaps to change it in a way that may not be detected.

Public Key MITM

By far the most serious man-in-the-middle problems are public key issues. In this sort of attack, the opponent arranges for the user to encipher with the key to the opponent, instead of the key to the far end. All long, random keys look remarkably alike, but using a key from the opponent allows the opponent to decipher the message, which completely exposes the plaintext. The opponent then re-enciphers that plaintext under the key to the far end and sends along the resulting ciphertext so neither end will know anything is wrong.

The public-key MITM attack targets the idea that many people will send their public keys on the network. The bad part of this is a lack of public-key certification. Unless public keys are properly authenticated by the user, the MITM can send a key just as easily, and pretend to be the other end. Then, if one *uses* that key, one has secure communication with the opponent, instead of the far end. So a message to the desired party goes through the opponent, where the message is deciphered, read, and re-enciphered with the correct key for the far end. In this way, the opponent quickly reads the exact conversation with minimal effort and without breaking the cipher *per se,* or cryptanalysis of any sort, yet neither end sees anything suspicious.

All of this depends on the opponent being able to intercept and change the message in transit. The original cryptographic model related to radio transmission and *assumed* that an opponent could *listen* to the ciphertext traffic, and perhaps even *interfere* with it, but *not* that messages could be intercepted and completely hidden. Unfortunately, message interception and substitution is a far more realistic possibility in a store-and-forward computer network than the radio-wave model would imply. Routing is *not* secure on the Internet, and it is at least conceivable that messages between two people are being routed through connections on the other side of the world. This property might well be exploited to make such messages flow through a particular computer for special processing. Again, neither end would see anything suspicious.

Perhaps the worst part of this is that a successful MITM attack does not involve *any* attack on the actual ciphering. Even a mathematical proof of the security of a particular cipher would be irrelevant in a system which allows MITM attacks.

CBC First Block MITM

A related (but very limited) version of an MITM attack can occur in the first block of CBC block cipher operating mode. In CBC, the IV is exclusive-ORed with the plaintext of the first block during deciphering. So if the opponents can somehow *change* the IV in transit, they can also change the resulting first block plaintext. And if the opponents know what plaintext was sent (perhaps a logo, or a date, or a name, or a command, or even a fixed dollar figure), they can change it to anything they want (in the first block).

Since the problem exploited in public-key MITM attacks is a lack of authentication, one might jump to the conclusion that *all* MITM attacks are authentication problems. But the authentication needed for public-key MITM attacks is not the authentication of an IV, nor even the authentication of plaintext, but instead the authentication of the public key itself. Key authentication is a fundamentally different issue than the CBC IV and first block problem.

Unlike public-key MITM attacks, the problem with the CBC first block is **not** a lack of authentication, but rather a lack of confidentiality for the IV. It is the lack of confidentiality which allows the IV value to be usefully manipulated (see CBC). That makes CBC first-block MITM a cipher-level problem, something appropriately solved below the cipher system level. It is often said that an IV can simply be transmitted in the open, but it is exactly that exposure which enables the first-block CBC MITM problem.

Preventing MITM Attacks

The way to avoid CBC first-block MITM problems is to encipher the IV instead of exposing it. Alternately, a higher-level MAC could be used to detect any changes in the plaintext.

The way to avoid public-key MITM attacks is to certify the keys, but this is inconvenient and time-consuming. So, unless the cipher system actually *requires* keys to be certified, this is rarely done. The worst part is that a successful MITM attack consumes few resources, need not "break" the cipher proper, and may provide just the kind of white-collar desktop intelligence a bureaucracy would love.

It is interesting to note that, regardless of how inconvenient it may be to share keys for a secret-key cipher, this is an inherent authentication which prevents the horribly complete exposure of public-key MITM attacks.

Mapping

Given sets *X* and *Y,* and operation *f*

```
f: X -> Y  ,
```

the *mapping* or *function* or *transformation* or *rule f* takes any value in the domain *X* into some value in the range, which is contained in *Y*. For each element *x* in *X,* a mapping associates a single element *y* in *Y*. Element *f(x)* in *Y* is the *image* of element *x* in *X.*

- If *f(X)* covers all elements in *Y, f* is a mapping of *X* **onto** *Y,* and is **surjective**.
- If *f(X)* only partially covers *Y, f* is a mapping of *X* **into** *Y.*

If no two values of *x* in *X* produce the same result *f(x), f* is **one-to-one** or **injective**.

- If *f* is both injective and surjective, it is **one-to-one and onto** or **bijective**.
- If *f* is bijective, there exists an **inverse** f^{-1} such that:

  $$f^{-1}(f(x)) = x.$$

- If *f* is identical with f^{-1}, *f* is an **involution**.
- A permutation of *X* is a **bijection** from *X* to *X.*

Markov Chain

A type of stochastic process. See Markov process.

Markov Process

(Markov chain.) In statistics, a stochastic (random) process (function) in which all possible outcomes are defined by the current state, independent of all previous states. A type of stochastic process.

A mathematical model consisting of some number of states, with transitions between states occurring at random but with some associated probability, and a symbol associated with each state which is output when that state is entered. One specific type is the ergodic process. One example is a random walk. Also see the non-random finite state machine, stationary process and hidden Markov model.

Mathematical Cryptography

Cryptography based on mathematical operations, such as taking extremely large values to extremely large powers, modulo the product of two primes. Normally heavily involved with number theory. As opposed to mechanistic cryptography (also see heuristic).

Mathematics is based on the correct logic of argumentation which is studied in philosophy. As such, it should be *impossible* (or at least exceedingly *embarrassing*) for math to support invalid conclusions. But I claim that happens all the time in cryptography. The problem seems to be a failure to recognize the distinction between theory and practice. In particular, many common proof assumptions are simply *impossible* to guarantee in practice, which means the proof results cannot be relied upon.

Despite claims to the contrary, **cryptography is *not* mathematics!** Instead, math is a general modeling tool. Cryptography is an *applications area* which applied math can model. But, in *any* field, the utility of results *always* depends upon the extent to which some model corresponds to reality. There is nothing new about this: The distinction between theory and practice is pervasive in science. Learning the meaning of modeling, how to apply models, that models have limits, and what to do outside a model, are fundamental parts of a scientific education. Clearly, cryptographic models must first *correctly* model reality before they can be said to *apply* to reality. The need for model validation is often *unmentioned* or *forgotten* or just considered *irrelevant* in the rush to glorify a new crypto math proof.

Cryptography is simply *different* from most fields that use math models. In practice, the worth of a real cryptosystem often depends upon point of view: The various ciphering actors--sender, receiver, opponent and designer--each have a different view of a cryptosystem, limited by what they *can* know. But we almost never see mathematicians craft strength proofs for the information-limited user contexts. Nor do we see arguments that such proofs cannot be achieved, which, of course, would directly address systemic limitations in the current concept of cryptography.

In mathematics it is common to say things like: "Let us assume that we have situation x; now let's prove what that would mean for other things." In this way, absolute logical requirements are easily handwaved into mathematical existence. But actually *achieving* those requirements in practice is another story. Normally, math requirements are not *prescriptive*; that is, they do not describe *how* a property is to be provably obtained, only that it somehow *be* obtained. Math normally provides no assurance that a handwaved property even *can* be achieved in practice. But if the needed property *cannot* be achieved, then, obviously, all attempts to achieve it are, and have been, a useless waste of time. That can be somewhat irritating to those who seek the practical use of math results in real systems.

If, for example, math assumes the existence of an unpredictable random number generator, cryptographic Perfect Secrecy can be proven. But in practice, we can guarantee no such thing. Oh, we can build generators that *seem* pretty unpredictable (see really random), but finding an *absolute guarantee* that the actual machines *are* unpredictable *at the time they are used* seems beyond our reach.

Without an *absolute guarantee* that *each and every* assumption *has been identified* and is *simultaneously achieved in practice*, a supposed "proof" is not even a complete logical argument. Such a "proof" is *formally incomplete* in practice, and so technically concludes **nothing at all**. In cryptography, theoretical proofs thus tend to create unfounded belief, something both science and mathematics should be working hard to avoid and debunk.

Almost no theoretical math security proofs apply in practice, yet most math-oriented crypto texts seem to say they do (see, for example, the one time pad proofs). In my view, that means mathematical cryptography has not yet been forced to address the distinction between theory and practice. If mathematical cryptography is to apply in reality, it must be as an *applied* discipline, especially those parts which are now largely mathematical illusion.

Alas, mathematical cryptography seems not very concerned about the situation, since things could be done differently but are not. One approach might be to use *only* properties which provably *can* be achieved in practice. That would, of course, greatly restrict mathematical "progress," but only if we take "progress" to include useless results. Until the math guys really do get concerned about reality, they necessarily leave it up to the individual practitioner to identify those theoretical results that do not apply in practice. In this way, ordinary workers in the field are being required to reason *better* than the math guys themselves. Yet sloppy reasoning is how some of the most commonly-held views in cryptography can be simply false (see old wives' tales).

Difficulties also exist in taking mathematical experience and applying that to cryptography:

1. Mathematical symbology has evolved for concise expression. It is thus not "isomorphic" to the complexity of the implementation, and so is not a good vehicle for the design-time trade-off of computation versus strength.
2. Most mathematical operations are useful or "beautiful" relationships specifically intended to support understanding in either direction, as opposed to relationships which might be particularly difficult to reverse or infer. So when using the traditional operations for cryptography, we must first defeat the very properties which made these operations so valuable in their normal use.
3. Mathematics has evolved to produce, describe and expose *structure*, as in useful or "beautiful" large-scale relationships and groupings. But, in a sense, relationships and groupings are the exact opposite of the fine-grained completely random mappings that cryptography would like to see. Such mappings are awkward to express mathematically, and contain little of the structure which mathematics is intended to describe.
4. There may be an ingrained tendency in math practitioners, based on long practice, to *construct* math-like relationships, and such relationships are not desirable in this application. So when using math to construct cryptography, we may first have to defeat our own training and tendencies to group, understand and simplify.

On the other hand, mathematics is *irreplaceable* in providing the tools to pick out and describe *structure* in apparently strong cipher designs. (See, for example, Boolean function nonlinearity and my comments on experimental S-box nonlinearity measurement.) Mathematics can identify specific strength problems, and evaluate potential fixes. But there appears to be no real hope of evaluating strength with respect to *every possible* attack, even using mathematics.

Although mathematical cryptography has held out the promise of providing *provable* security, in over 50 years of work, **no** practical cipher has been generally accepted as having *proven* strength. See, for example: one time pad and proof.

Math Notation

{0,1}: a set of two values.

$\{0,1\}^n$: the set of all bit strings of length *n*.

$\{0,1\}^*$: the set of all bit strings of arbitrary length.

(a,b): possibly the open interval of real number values from *a* to *b*, **not** including *a* or *b*; or an ordered pair of two values as in complex numbers; or in number theory the gcd of *a* and *b*.
[a,b]: the closed interval of real number values from *a* to *b*, including both *a* and *b*.
C: the field of complex numbers.
F_q: a finite field of order q.
F_q*: the multiplicative group of the finite field of order q.
(F_q,+): the additive group of the finite field of order q.
(G,*): a group under multiplication.
GF(x): a Galois field.
GF(2): the Galois field of two elements: 0 and 1.
GF(2^n): the Galois field of 2^n elements.
GF(2)[x]: the ring of polynomials in x whose elements are in the set GF(2).
GF(2)[x]/p(x): the field of polynomials in x whose elements are in the set GF(2), modulo irreducible polynomial p(x).
N: the natural numbers; the set {0, 1, 2, ...}
R: the field of real numbers which represent continuous quanties.
(R,+): the group of real numbers under addition.
R[x]: the ring of polynomials in x over ring R.
R/I: the residue class of the ring R modulo the ideal I.
Z: the ring of integers; the set {..., -2, -1, 0, 1, 2, ...}.
(Z,+): the group formed by integers with addition.
(Z,+,*): the ring formed by integers with addition and multiplication.
Z/n: the ring of integers modulo n.
Z_n: the ring of integers modulo n.
Z_n^*: the multiplicative group of integers modulo n.
Z/p: the field of integers modulo prime p.
Z/pZ: the field of integers modulo prime p.
(Z/pZ)[x]: the ring of polynomials in x with coefficients in the field ***Z/pZ***.

Maximal Length

An RNG or FSM with a cycle structure consisting of just a single cycle, or perhaps a degenerate cycle plus one long cycle. In cryptographic use (especially as stream cipher running key generators) having a guaranteed minimal length is an important RNG property: Keyed RNG's may *seem* unpredictable, but when a short sequence repeats, that sequence has just become predictable and insecure.

Originally a linear feedback shift register (LFSR) sequence of 2^n-1 steps produced by an *n*-bit-wide shift register. This means that every binary value the register can hold, except zero, will occur on some step, and then not occur again until all other values have been produced. A maximal-length LFSR can be considered a binary counter in which the count values have been shuffled or enciphered. The sequence from a normal binary counter is perfectly balanced and the sequence from a maximal-length LFSR is *almost* perfectly balanced. Also see M-sequence.

MB

Megabyte. 2^{20} or 1,048,576 bytes.

Mb

Megabit. 2^{20} or 1,048,576 bits.

MDS Codes

Maximum Distance Separable codes. In coding theory, codes with the greatest possible error-correction capability. These codes expand the original data with a wider representation so that even multiple bit changes are still "closer" to the correct value than some other.

MDS codes are supposed to be useful in the wide trail strategy for conventional block cipher design. Apparently MDS codes can be used to make some minimum number of S-boxes active in such a design.

However, MDS codes are not applicable to all designs, nor are such codes needed for optimal ciphering. The obvious counterexample is my mixing cipher designs, which use small Balanced Block Mixing operations (actually, orthogonal Latin squares). The oLs's are arranged into scalable FFT-like structures to mix every input byte into every output byte, and to diffuse even a small input change across each and every output.

Mean

In descriptive statistics, a measure of the "central tendency" (the extent to which data tend to cluster around a specific value). Commonly the arithmetic average, the sum of n values divided by n.

Similar computations include the geometric mean (the nth root of the product of n values, used for average rate of return) and the harmonic mean (n divided by the sum of the inverses of each value, used to compute mean sample size), among others.

Mechanism

The logical concept of a machine, which may be realized either as a physical machine, or as a sequence of logical commands executed by a physical machine.

A mechanism can be seen as a process or an implementation for performing that process (such as electronic hardware, computer software, hybrids, or the like).

Mechanistic Cryptography

Cryptography based on mechanisms, or machines. As opposed to mathematical cryptography. (Also see heuristic).

Although perhaps looked down upon by those of the mathematical cryptography persuasion, mechanistic cryptography certainly does use mathematics to design and predict performance. But rather than being restricted to arithmetic operations, mechanistic cryptography tends to use a wide variety of mechanically-simple components which may not have concise mathematical descriptions. Rather than simply implementing a system of math expressions, complexity is constructed from the various efficient components available to digital computation.

Median

In descriptive statistics, in a list of numbers, the smallest value which is greater or equal to at least half of the numbers. The "middle" value in an ordered list.

Mere Semantics

A phrase generally implying that using a different word to describe the same facts does not change the essence of the facts. (Also called just semantics.)

However, words are how we discuss facts, and semantics is the meaning of those words, making semantics somewhat more important than "mere." Finding that a discussion is not using the expected meaning of a term, or that multiple different meanings are being used simultaneously, shows that the discussion itself is in trouble. (See argumentation, logic and fallacy.)

For example:

- If someone claims that a cipher is proven secure, and then we look at the many meanings for "proof" and find that none of them apply, we can see the claim is false. That is not just using a different word to describe reality, it is showing that the claims are factually false.

- If a theoretical proof leads someone to believe that the one time pad is "unbreakable," the uncomfortable fact that some such ciphers actually have been broken in practice can create considerable cognitive dissonance. Fortunately, Science does not require belief. Normally, when a theoretical scientific model is shown to not apply in practice, we publicize that or correct the model. That is not using a different word to describe the same situation, it is finding the correct words to describe reality.

Mersenne Prime

A prime p for which $2^p - 1$ is also prime. For example, 5 is a Mersenne prime because $2^5 - 1 = 31$, and 31 is prime. For mod 2 polynomials of Mersenne prime degree, *every* irreducible is also primitive.

```
Mersenne Primes:
    2         107          9689         216091
    3         127          9941         756839
    5         521         11213         859433
    7         607         19937        1257787
   13        1279         21701        1398269
   17        2203         23209        2976221
   19        2281         44497        3021377
   31        3217         86243        6972593
   61        4253        110503       13466917
   89        4423        132049
```

Message

Information represented as a sequence of symbols. Often as readable human language, and now often also as values represented as binary for digital transmission. Also see plaintext.

A message can be seen as sequence of values. Consequently, it may seem that the only cryptographic manipulations possible would be either to change values (substitution) or change position (transposition). However, it is also possible to include meaningless values in or between messages (as in nulls). A variation is to intermix several different messages as in a braid or grille. Another possibility is to collect groups of symbols and change those values as a unit, which is technically a code.

Note that message *length* is rarely hidden by ciphers. One way of hiding length is by continuous transmission with nulls between messages. Naturally, it is then necessary to identify the start and end of each message, which may involve various synchronization techniques. Another way of hiding length is to use nulls to expand the message by a random amount.

Message Archives

Storage of messages for later use. This becomes a particular issue when the messages contain sensitive information. Clearly, if we save messages in their encrypted form, we also need keys to expose their contents, even though updated keys may be in use for that same channel (see key storage).

Fortunately, if we have all the keys ever used for some channel, a start date for each is sufficient to define the period of activity for a particular key. So if we know the message date, we can select the key which was active on that date. With email, we might parse the header for the message date, and so automatically select the correct old key.

One possibility for local user archives is to decrypt all messages when received, and accumulate the plaintext files. A modified version of that is to re-encrypt each message under the user's own keyphrase, although that could be a problem when it comes time to change that keyphrase, or if the user leaves. Yet another alternative might be to simply archive each message as received, even encrypted, and then the select old keys needed to access old encrypted messages.

All these user-centric approaches have in common the problem that the user's archives are assumed to be intact, that the computer has not crashed or been deliberately erased. That is an assumption a corporation may not be prepared to accept.

In contrast, corporate security policies may want to archive *all* messages, from and to all users. When the corporation is the source of all keys (see key creation), it would have all the alias files and all the old keys for each user. To select the correct key, we need to identify the user, the email address or name (the "alias") for the far end, and the message date. Again, email message headers can be parsed for this information, especially if users follow reasonable alias protocols. (Note that alias protocols may the least part of dealing with sensitive information and encryption.) If each message is kept in a different file, that file can be given the appropriate date for that message which then provides another source of the date for key selection.

Since any decent cipher system should report the use of the wrong key, all new encrypted messages could be automatically checked for correct decryption. Action could then be taken quickly if the appropriate keys were not being used.

Message Authentication

Assurance that a message is from the purported sender. Sometimes part of message integrity.

To a large extent, message authentication depends upon the use of particular keys to which opponents should not have access. Simply receiving an intelligible message thus indicates authentication. But automatic authentication may require added message redundancy in the form of a hash value that can be checked upon receipt, similar to message integrity.

Another approach to message authentication is to use an authenticating block cipher; this is often a block cipher which has a large block, with some "extra data" inserted in an "authentication field" as part of the plaintext before enciphering each block. The "extra data" can be some transformation of the key, the plaintext, and/or a sequence number. This essentially creates a homophonic block cipher: If we know the key, many different ciphertexts will produce the same plaintext field, but only one of those will have the correct authentication field.

The usual approach to authentication in a public key cipher is to encipher with the private key. The resulting ciphertext can then be deciphered by the public key, which anyone can know. Since even the wrong key will produce a "deciphered" result, it is also necessary to identify the resulting plaintext as a valid message; in general this will also require redundancy in the form of a hash value in the plaintext. The process provides no secrecy, but only a person with access to the private key could have enciphered the message. Also see: key authentication and user authentication.

Message Authentication Code

(MAC). A keyed error detecting code or hash. Integrity and authentication are both verified when the correct MAC key produces the same result as the transmitted authentication field. See: message authentication and message integrity.

Although widely touted and used, a MAC is hardly the only possible form of authentication. A MAC normally functions across a whole message, and thus requires that the entire message exist before authentication can operate. One alternate form of authentication is a per-block authentication field (see homophonic substitution and block code). This allows each block to be authenticated, and possibly could even replace standard Internet Protocol error-detection, thus reducing system overhead. Presumably, other forms of authentication are also possible.

Message Digest

A small value which represents an entire message for purposes of integrity or possibly authentication; a hash.

Message Integrity

Assurance that a message has not been modified during transmission, sometimes including message authentication. One approach computes a CRC hash across the plaintext data, and appends the CRC remainder (or *result*) to the plaintext data: this adds a computed redundancy to an arbitrary message. The CRC result is then enciphered along with the data. When the message is deciphered, if a second CRC operation produces the same result, the message can be assumed unchanged.

Note that a CRC is a fast, linear hash. Messages with particular CRC result values can be constructed rather easily. However, if the CRC is hidden behind strong ciphering, an opponent is unlikely to be able to change the CRC value systematically or effectively. In particular, this means that the CRC value will need more protection than a simple exclusive-OR in an additive stream cipher or the exclusive-OR approach to handling short last blocks in a block cipher.

A similar approach to message integrity uses a nonlinear cryptographic hash function or MAC. These also add a computed redundancy to the message, but generally require more computation than a CRC. While cryptographic hashes generally purport to have significant security properties, those are rarely if ever proven to the same extent as the lesser properties of a simple CRC. It is thought to be exceedingly difficult to construct messages with a particular cryptographic hash result, so the hash result perhaps need not be hidden by encryption. Of course doing that is just tempting fate.

Another approach to message integrity is to use an authenticating block cipher; this is often a block cipher which has a large block, with some "extra data" inserted in an "authentication field" as part of the plaintext before enciphering each block. The "extra data" can be some transformation of the key, the plaintext, and/or a sequence number. This essentially creates a homophonic block cipher: If we know the key, many different ciphertexts will produce the same plaintext field, but only one of those will have the correct authentication field.

Message Key

A key transported with the message and used for deciphering the message. (The idea of a "session key" is very similar, but presumably lasts across multiple messages.)

Normally, the message key is a large really random value or nonce, which becomes the key for ciphering the data in a single message (see cipher system). Normally, the message key itself is enciphered under the User Key or other key for that link (see alias file and key management). The receiving end first deciphers the message key, then uses that value as the key for deciphering the message data. Alternately, the random value itself may be sent unenciphered, but is then enciphered or hashed (under a keyed cryptographic hash) to produce a value used as the data ciphering key.

Message keys have very substantial advantages:

- A message key assures that the actual data is ciphered under an arbitrary selection from a huge number of possible keys; it therefore prevents weakness due to user key selection.
- A message key is used exactly once, so even a successful brute force attack on a message key exposes just one message.
- A message key is constructed internally, and this construction cannot be controlled by a user; this prevents all attacks based on repeated ciphering under a single key or a known sequence of keys.
- To the extent that the message key value is unpredictable, it is more easily protected than ordinary text (see Ideal Secrecy).
- Since message key values are never exposed to users on on either end, the opponent never has the known plaintext for the message key values.

It is important that message key construction be made clear and straightforward in design and implementation.

Like most nonces, a message key is "extra" data, the value of which is not important. That value thus could be subverted to become a hidden side-channel for disclosing secure information.

In a sense, a message key is the higher-level concept of an IV, which is necessarily distinct for each particular design. Some form of message key is the usual way to implement a hybrid or public key cipher.

Metastability

The condition where the voltage output of a digital device is an *invalid* logic level between logic 0 and logic 1 for an unexpectedly long time. This happens to be the voltage which, when placed at an input, and amplified through two particular inverting stages, produces exactly the same voltage (or waveform) on the second output. We know there must be such a point, because an input which is "slightly 1" gives a "hard 1" out, and "slightly 0" gives a "hard 0" out; somewhere between these levels, some voltage must reproduce itself. This is a consequence of digital logic being built from internal transistor amplifiers which are inherently analog, and not digital.

Metastability is typically caused by violation of the set-up and hold time requirements of a flip-flop, which can cause an intermediate voltage level to be latched. Note that intermediate voltage levels *always* occur when a digital signal changes state; in a well-designed system they are ignored by clock signals which provide time for the transient condition to pass. Metastability occurs when the "amplified" level in a flip-flop happens to be exactly the same as the input level at the time the clock connects these points. The condition can endure indefinitely, until internal noise causes a collapse one way or the other.

Metastability *cannot be prevented* in designs which use a digital flip-flop or latch to capture raw analog data or unsynchronized digital data such as digitized noise. Metastability can be *reduced* by minimizing the time the signal spends in the invalid region, for example by using faster logic and/or Schmitt trigger devices. Metastability can be *greatly* reduced by using more than one stage of clocked latch. Metastability is *eliminated* in logic design by assuring valid logic levels and timing, such that setup and hold times are never violated.

In the simple case, where a single input line is sampled, metastability may only cause an occasional unexpected delay and an uncontrolled non-random bias. But if multiple lines are sampled and metastability occurs on even one of those, a completely different value can be produced. In some systems, latching a wrong value could lead to entering unexpected or prohibited states with undefined results.

Method of Proof and Refutations

A process for developing mathematical proof:

- "**Rule 1:** If you have a conjecture, set out to prove it and refute it. Inspect the proof carefully to prepare a list of non-trivial lemmas (proof-analysis); find counterexamples both to the conjecture (global counterexamples) and to the suspect lemmas (local counterexamples).

- "**Rule 2:** If you have a global counterexample and discard your conjecture, add to your proof-analysis a suitable lemma that will be refuted by the counterexample, and replace the discarded conjecture by an improved one that incorporates that lemma as a condition. Do not allow a refutation to be dismissed as a monster. Try to make all 'hidden lemmas' explicit.

- "**Rule 3:** If you have a local counterexample, check to see whether it is not also a global counterexample. If it is, you can easily apply Rule 2.

- "**Rule 4:** If you have a counterexample which is local but not global, try to improve your proof-analysis by replacing the refuted lemma by an unfalsified one.

- "**Rule 5:** If you have counterexamples of any type, try to find by deductive guessing, a deeper theorem to which they are counterexamples no longer."

". . . a valid *[formal]* proof is one in which *no matter how one interprets the descriptive terms*, one never produces a counterexample -- i.e. its validity does not depend on the meaning of the descriptive terms . . . ." [pp.100,124]

"For any *[informal]* proposition there is always some sufficiently narrow interpretation of its terms, such that it turns out true, and some sufficiently wide interpretation, that it turns out false." [p.99]

". . . informal, quasi-empirical, mathematics does not grow through a monotonous increase of the number of indubitably established theorems, but through the incessant improvement of guesses by speculation and criticism, by the logic of proofs and refutations." [p.5]

"Refutations, inconsistencies, criticism in general are very important, but only if they lead to improvement. *A mere refutation is no victory*. If mere criticism, even though correct, had authority, Berkeley would have stopped the development of mathematics and Dirac could not have found an editor for his papers." [p.112]

-- Lakatos, I. 1976. *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge University Press.

Military Grade

(Also "Military Strength.") A casual description of cipher strength; a term intended to convey the idea that data security is being taken seriously. Unfortunately, this is at best a belief, and not proven fact.

Any such claim is flawed in multiple ways:

- It seems unlikely that anyone who knows anything about real military ciphers is going to talk about them in a marketing bulletin. So probably what we get is someone's delusion about what military ciphers must be like.
- *Even if* some cipher is *similar* to a military cipher, mere similarity is not enough to predict strength, thus quality and worth.
- Nobody at all, anywhere, knows the strength of an unbroken cipher, whether military or civilian, because it is the breaking which exposes and defines the weakness. And even that result is just the strength known to a particular community with particular techniques; the strength can be much less to someone else.
- *There can be no informed opinion on the strength of an unbroken cipher.*

Misdirection

The deliberate attempt to focus attention away from some important reality.

MITM

Man In The Middle.

Mixing

The act of transforming multiple input values into one or more output values, such that changing any input value will change the output value. There is no implication that the result must be balanced, but effective mixing may need to be, in some sense, complete. Also see Mixing Cipher, combiner, Latin square combiner, and Balanced Block Mixing.

Mixing Cipher

A block cipher based on Balanced Block Mixing of small elements in FFT-like or FWT-like mixing patterns.

Below, we have a toy 32-bit-block Mixing Cipher. Plaintext at the top is transformed into ciphertext at the bottom. Each "S" is an 8-bit substitution table, and each table (and now each mixing operation also) is individually keyed.

Horizontal lines connect elements which are to be mixed together: Each *---* represents a single Balanced Block Mixing or BBM. Each BBM takes two elements, mixes them, and returns two mixed values. The mixed results then replace the original values in the selected positions just like the butterfly operations used in some FFT's.

```
   A 32-Bit Mixing Cipher

    |   |   |   |    <- Input Block (Plaintext)
    S   S   S   S    <- Fencing
    |   |   |   |
    *---*   *---*    <- 2 BBM Mixings
    |   |   |   |
    *-------*   |    <- 1 BBM Mixing
    |   *-------*    <- 1 BBM Mixing
    |   |   |   |
    S   S   S   S    <- Fencing
    |   |   |   |
    *-------*   |
    |   *-------*
    |   |   |   |
    *---*   *---*
    |   |   |   |
    S   S   S   S    <- Fencing
    |   |   |   |    <- Output Block (Ciphertext)
```

By mixing each element with another, and then each pair with another pair and so on, every element is eventually mixed with every other element. Each BBM mixing is dyadic, so each "sub-level" is a mixing of twice as many elements as the sublevel before it. A block of *n* elements is thus fully mixed in $\log_2 n$ sublevels, and each result element is equally influenced equally by each and every input element.

The pattern of these mixings is exactly like some implementations of the FFT, and thus the term "FFT-style." See Balanced Block Mixing and Mixing Cipher Design Strategy. Also see the info and articles in the "Mixing Ciphers" section of the main page, locally, or @: http://www.ciphersbyritter.com/index.html#MixTech.

Mixing Cipher Design Strategy

A Mixing Cipher basically consists of a fixed number of separate layers (not rounds) of Shannon-style product ciphering (see Algebra of Secrecy Systems), consisting of layered confusion and diffusion. Typically, the confusion layers consist of substitution tables or S-boxes; I call such layers fencing. The diffusion layers are scalable FFT-style networks using Balanced Block Mixing operations; I call those layers mixing. Both the fencing and mixing layers can be keyed, and the particular table or BBM used at any position can be selected from a keyed array of tables. Some advantages include:

- We avoid simply *giving* the opponent the explicit table contents so often present in cipher designs.
- The use of layers instead of rounds avoids re-using the same operations, so that what is revealed about one layer does not apply to subsequent layers.
- Instead of doing things over and over and hoping for the best, we do things right once and then move on.

Goals

The usual goals for a conventional block cipher design are *strength* and *speed*. But to those goals, I add *scalability*, *huge blocks*, *massive keyed internal state* and *clarity*:

- **Scalability** is the *only* way we can exhaustively test a real cipher. Without scalability, all we have are mathematical lemmas, each with assumptions that somehow never can be completely satisfied in practice. Only scalability can allow us to exhaustively test and certify the exact code (software) actually used in practice.
- **Huge Blocks** offer usage advantages which are simply impractical in relatively small blocks. These

include (see Huge Block Cipher Advantages):
    - Dynamic Keying
    - Per-Block Authentication
    - Homophonic Operation
- **Large Keyed Internal State** is information an opponent must somehow reconstitute from external observation. This is a slightly different and more believable form of strength than the usual claim that a complex computation based on fixed constants and tables with the ordinary (small) keyspace somehow cannot be undone.
- **Clarity** leads to a better understanding of cipher operation, and, thus, easier analysis and a better awareness of strength, which, after all, is the primary design goal.

We can achieve these goals with Shannon product ciphering, by interleaving mainly confusion and mainly diffusion operations in independent layers.

Diffusion

Perhaps the main issue in the design of block ciphers with huge blocks has always been the ability to efficiently mix information across the whole block. First recall substitution-permutation ciphering, where we build a conventional block cipher using only fencing layers (substitution tables) and wiring. By wiring substitution table outputs to two different following tables, changing the input to the first table may change both of the secondary tables, and that is diffusion. The problem is that both tables may *not* change. If some input change causes no change on the wires to some table, that table *and possibly its subsequent tables* will not be involved, which will give the opponent a simpler ciphering transformation to attack. We want to eliminate that possibility. Accordingly, what I call ideal mixing will cause *any* input change to be conducted to *every* following table, thus forcing *all* tables to actively participate in the result.

We now know that ideal mixing can be accomplished with FFT-like networks using relatively small BBM operations. However, in the U.S., ciphers which do not use my Balanced Block Mixing technology must use mixing operations which are inherently unbalanced. Since I view balance as the single most important concept in cryptography, avoiding that when we can get it would seem to be a very serious decision.

Building a scalable balanced block mixing process is fairly easy using ideal mixing BBM technology. Basically, each mixing operation must have a pair of orthogonal Latin squares, and those can be linear, nonlinear, or key-created, or combinations thereof. My bias is to use key-created oLs's in tables. (It is easy to construct keyed nonlinear orthogonal pairs of Latin squares of arbitrary 4n order as I describe in my articles:
- "Orthogonal Latin Squares, Nonlinear Balanced Block Mixers, and Mixing Ciphers" locally, or @: http://www.ciphersbyritter.com/ARTS/NONLBBM.HTM)
- and "Practical Latin Square Combiners" locally, or @: http://www.ciphersbyritter.com/ARTS/PRACTLAT.HTM

Confusion

I prefer to use "many" keyed substitution tables, because these hold a large amount of unknown internal state which an opponent must somehow reconstruct. These S-boxes are simply shuffled (twice) by a keying sequence produced by some cryptographic RNG. Keying by shuffling is very old technology, very common in software stream ciphers (although simply using that technology does not somehow comvert a block into a stream). Tables need to be at least "8-bit" or "byte-wide" with at least 256 byte entries.

One of the more subtle problems with scalable Mixing Ciphers is that some limited quantity (maybe 16, maybe 64, maybe more) of keyed tables (and keyed oLs mixing operations as well) may have to support a block of essentially unlimited size. Thus, tables must be re-used; tables can be selected from an array of such tables based on some keyed function of position. It is important that this be keyed and sufficiently complex so that knowing something about a table in one position does not immediately allow assigning that knowlege to other table positions.

One way to handle table selection by cipher position is to have a maximum block size, and then have a table of that size for each layer, indicating which S-box or BBM to use at each position. Those tables can be keyed (shuffled) just like other tables. Again, we avoid exposing any significant constants in the design by constructing what we need with RNG sequences based on a key.

Layers

We do something right, then move on. We do not have to do things over and over until they finally work. In the past I have used two linear mixing layers, which implies three fencing layers, and I still might use that structure with keyed nonlinear mixing. However, having become somewhat more conservative, I might now use three linear mixing layers (instead of two) along with four fencing layers. With a layered design, where it is easy to add or remove layers, there is always a desire to reduce computation and increase speed, and it is easy to go too far.

Mod 2

The field denoted GF(2) and formed from the set of integers {0,1} with operations + and * producing the remainder after dividing by modulus 2. Thus:

```
     0 + 0 = 0
     0 + 1 = 1
     1 + 0 = 1
     1 + 1 = 0

     1 + 1 + 1 = 1

     0 * 0 = 0
     0 * 1 = 0
     1 * 0 = 0
     1 * 1 = 1
```

Subtraction mod 2 is the same as addition mod 2. The operations + and * can also be considered the logic functions XOR and AND respectively.

Mod 2 Polynomial

The ring of polynomials, denoted GF(2)[x], in which the coefficients are taken mod 2. The four arithmetic operations addition, subtraction, multiplication and division are supported, but, like integers, division is not closed. As usual in mod 2, subtraction is the same as addition. Each column of coefficients is added separately, without a "carry" to an adjacent column:

```
Addition and Subtraction:

       1 0 1 1
     + 0 1 0 1
     + 1 1 0 0
      ---------
       0 0 1 0


Multiplication:

          1 0 1 1
        * 1 1 0 0
        ----------
```

```
                  0
                0
        1 0 1 1
      1 0 1 1
     ---------------
      1 1 1 0 1 0 0
```

Polynomial multiplication is *not* the same as repeated polynomial addition. But there is a fast approach to squaring mod 2 polynomials:

```
                a   b   c   d
                a   b   c   d
               ------------
               ad  bd  cd  dd
            ac  bc  cc  dc
         ab  bb  cb  db
      aa  ba  ca  da
     ----------------------
        a   0   b   0   c   0   d
```

To square a mod 2 polynomial, all we have to do is "insert" a zero between every column. Note that aa = a for a = 0 or a = 1, and ab = ba, so either 0 + 0 = 0 or 1 + 1 = 0.

```
Division:
                    1 0 1 1
            ----------------
    1 1 0 0 ) 1 1 1 0 1 0 0
              1 1 0 0
             ---------
                  1 0 1 0
                  1 1 0 0
                 ---------
                    1 1 0 0
                    1 1 0 0
                   ---------
                          0
```

The decision about whether the divisor "goes into" the dividend is based exclusively on the most-significant (leftmost) digit. This makes polynomial division far easier than integer division.

Mod 2 polynomials behave much like integers in that one polynomial may or may not divide another without remainder. This means that we can expect to find analogies to integer "primes," which we call *irreducible* polynomials.

Since division is not closed, mod 2 polynomials do not constitute a field. However, a finite field of polynomials can be created by choosing an irreducible modulus polynomial, thus producing a Galois field GF(2^n).

Mode

1. In descriptive statistics, in a list of values, the value or values which occur most often. The most common values.
2. A block cipher operating mode.

Modulo

Casually, the remainder after an integer division by a modulus; see congruence. When the modulus is prime, this may generate a useful field.

Monadic

Relating to *monad*, which is Greek for single or one. In particular, a function with a single input or argument, also called unary. Also see: arity and dyadic.

Monoalphabetic Substitution

Substitution using a single alphabet. Also called simple substitution. As opposed to Polyalphabetic Substitution. Also see: a cipher taxonomy.

Monographic

Greek for "single letter." A cipher which translates one plaintext symbol at a time into ciphertext. As opposed to polygraphic; also see homophonic and polyphonic.

Monoid

In abstract algebra, a nonempty set *M*, and a closed dyadic operation with associativity, and an identity element. Whatever the operation is, we may choose to call it "multiplication" and denote it with * as usual. Closure means that if elements (not necessarily numbers) *a, b* are in *M*, then *ab* (that is, *a*b*) is also in *M*.

In a monoid consisting of set M and closed operation * :

1. **The operation * is associative:** (ab)c = a(bc)
2. **There is a single identity element which works with all elements:** for e and any a in M, ea = ae = a

A set with a closed operation which is just associative is a semigroup. A set with a closed operation which is associative, with an identity element and inverses is a group.

Monomial

An expression with a single term; an expression composed only of multiplied factors. As opposed to polynomial.

msb

Most-significant bit. In a binary representation, the bit of highest significance. Often the leftmost bit. As opposed to msB. Also see lsb and lsB.

msB

Most-significant byte. In a binary representation, the byte of highest significance. Often the leftmost byte. As opposed to msb. Also see lsb and lsB.

M-Sequence

A maximal length shift register sequence.

Multipermutation

"A new cryptographic primitive for perfect generation of diffusion and confusion. For an arbitrary set E we call a permutation $B:E^2 \to E^2$, $B(a,b)=(B_1(a,b),B_2(a,b))$ a *multipermutation* if, for every a,b in E, the mappings $B_i(a,*),B_i(*,b)$, for i=1,2 are permutations on E."

Schnorr, C. and S. Vaudenay. 1994. Parallel FFT-Hashing. *Fast Software Encryption.* 149-156.

This definition would seem to include orthogonal Latin squares, a more-desirable balanced form well-known in mathematics for hundreds of years. Their paper, on non-reversible hashing, may have been presented at the "Cambridge Security Workshop," Cambridge, December 9-11, 1993. My work on Balanced Block Mixing was published to the net on March 12, 1994, which was before the earlier paper was available in published proceedings. Even accepting the earlier paper, however, my work was the first to demonstrate FFT-like reversible ciphering using butterfly functions that turn out to be orthogonal Latin squares.

Multiple Anagramming

An attack on classical transposition ciphers. Given two messages of the same length, a classical transposition cipher would re-arrange the messages in a similar way. The attack is to anagram or re-arrange the characters in two same-length ciphertexts so that both make sense.

The attack depends upon the idea that messages of the same length will be permuted in the same way, and probably will not apply to modern transposition ciphers.

Multiple Ciphering

Multiple encryption, product ciphering, or cascade ciphering. First enciphering plaintext into ciphertext, then enciphering that ciphertext, probably using another cipher and key.

Multiple Encryption

Product ciphering, multiple ciphering or cascade ciphering. Enciphering or encrypting a message more than once. Normally this involves using an independent key for each ciphering, and may also use different ciphers (see Shannon's Algebra of Secrecy Systems). Since most conventional block ciphers are in fact product ciphers, they testify to the worth of multiple encryption, even with weak internal ciphering functions. Also see risk analysis and Pure Cipher.

The Usual Complaints

The point of multiple encryption is to reduce the damage if our main cipher is being broken without our knowledge. We thus compare the single-cipher case to the multiple-cipher case. But some people just do not like the idea of multiple encryption. Complaints against multiple encryption include:

- **It is not proven to make ciphering stronger.** We need to compare apples to apples: There is no proof that the alternative single cipher will have *any strength at all!* If we need a proof of strength in practice, we cannot use encryption, because no such cipher and proof exists (see, for example, proof and one-time pad). And if we can speculate that one of the ciphers is weak, the single-cipher case *completely fails*, whereas the two-cipher case is as strong as the second cipher would be alone.
- **There is no reason to *think* it would make ciphering stronger.** There are *very* good reasons to think that multiple encryption makes a stronger cipher! First is the real-world example of Triple DES: Despite extensive analysis, Triple DES is still unbroken, and Triple DES is the multiple encryption composed *only* of broken DES. There could hardly be a more dramatic example of strength improvement than Triple DES! Next, many if not most ciphers are *best* attacked with some form of known plaintext attack. But using multiple ciphers in sequence hides the known plaintext from external analysis, which prevents those attacks. Similar hiding is not possible with only a single ciphering.
- **It is not proven necessary.** If we are using a cipher, then presumably there is no successful attack on it *that we know of*. But there is *also* no proof that a successful attack does *not* exist, and that essentially **is** *proof of potential cipher weakness*, which is the risk of complete cipher failure. Moreover, that failure would occur in secret, and since we would not know about the failure, we would continue to use the broken cipher forever. We can hope to minimize the possibility of failure with the redundancy of multiple cipherings.
- **It has not been analyzed.** First, multiple encryption *was* analyzed in 1949 by Shannon in his famous work, Communication Theory of Secrecy Systems (see Algebra of Secrecy Systems). Next, the well-respected cipher Triple DES is an example of multiple encryption, and has had extensive analysis. Then, the rounds in conventional block ciphers are essentially a form of multiple encryption, and also have been heavily analyzed. And there *is* a substantial and growing body of work on multiple encryption; some modern articles are noted below.
- **There is no proof that using another cipher would not make things *weaker*.** Weak sequences of ciphers can be deliberately constructed, but their constructions must be *coordinated* to demonstrate weakness. Getting two different ciphers, with independent keys, to coordinate in weakness, seems very, very unlikely. And if any particular cipher would reduce the strength of another cipher, it would be a

fine academic attack. We see no serious proposals for this sort of attack on serious ciphers.
- **Each separate cipher stack can be seen as a resulting cipher that should be analyzed.** While multiple encryption has various advantages, it is first a response to a real, existing problem. That problem is the risk of the single point of failure represented by any single cipher. If cryptanalysis could prove that no failure could occur in our "main" cipher, the problem would be solved. The problem continues to exist specifically because cryptanalysis can *not* provide such proof. Demanding that analysis produce results for multiple ciphers beyond what it can produce for a single cipher is just deceptive nonsense.
- **Since a new attack could apply to all ciphers, it provides no real redundancy.** There is a real example: DES *was* "broken" by Differential and Linear Cryptanalysis and other techniques. Nevertheless, the multiple encryption with three DES cipherings called Triple DES was *not* broken. Triple DES remains secure because Differential and Linear Cryptanalysis cannot usefully be applied to it.

Is Weakness Possible?

When we have one cipher, could adding a second cipher weaken the result? Well, it is *possible*, but it also seems *extremely* unlikely. For example, weakening could happen *if* the second cipher was the same as the first, *and* in decipher mode (or an involution), *and* using the same key. But, except for the keying, *exactly* that situation is *deliberately constructed* in "EDE (encipher-decipher-encipher) Triple DES," about which there are no anxieties at all.

Remember that it is *always* possible for *any* cipher to be weak in practice, no matter how strong it is in general, or whether it has another cipher after it or not: All the opponent has to do is pick the right key. So, when we think about potential problems in *any* form of encryption, we also need to think about the likelihood of those causes actually happening in practice. Constructing a case of weakness is not particularly helpful if that does not apply generally. In any practical analysis, it is not very useful to find a counterexample which does not represent the whole; that is the classic logic fallacy known as accident.

Despite the "EDE Triple DES" example, the most obvious possibility of weakness would be for the same cipher to appear twice, in adjacent slots. Obviously, we can prevent that! Could a *different* cipher expose what a first cipher has just enciphered? Perhaps, but if so, they are not really *different* ciphers after all, and that would be something we could check experimentally. Can a single cipher have several fundamentally different constructions? That would seem to be difficult: Normally, even a small change in the ciphering process has far-reaching effects. But, again, we could check for that.

The idea that a completely *unrelated* cipher could decipher (and thus expose) what the first cipher had protected may seem reasonable to those with little or no experience in the practice of ciphering or who have never actually tried to break ciphertext. But if that approach was reasonable, it would be a major issue in the analysis of any new cipher, and we do not see that.

Ciphers transform plaintext into key-selected ciphertext, and we can describe the number of possible cipherings mathematically. Even restricting ourselves to small but serious block ciphers like DES or AES, the number of possible transformations is BIG, BIG, BIG (see AES)! Out of the plethora of possible ciphers, and every key for each of those ciphers, we expect only one cipher and one key to expose our information.

If many different ciphers and keys could expose an enciphered message, finding the limits of such vulnerability would be a major part of every cipher analysis, and that is not done because it is just not an issue. If just adding a second cipher would, in general, make the first cipher weaker, that would be a useful academic attack, and we see no serious proposals for such attacks on serious ciphers.

A Deceptive Article

Much of the confusion about the potential risks of multiple encryption seems due to one confusing article with a particularly unfortunate title:

> Maurer, U. and J. Massey. 1993. Cascade Ciphers: The Importance of Being First. *Journal of Cryptology*. 6(1):55-61.

Apparently the main goal of the article is to present a contradiction for:

> "**Folk Theorem.** *A cascade of ciphers is at least as difficult to break as any of its component ciphers*." [p.3]

In the end, the main result from the article seems to be:

> "It is proved, for very general notions of breaking a cipher and of problem difficulty, that a cascade is at least as difficult to break as the first component cipher." [Abstract]

But while that may *sound* insightful, it just means that if the first cipher is *weak*, the cascade may be strong anyway, which is no different than the "Folk Theorem." And if the first cipher is *strong*, the result tells us nothing at all. (What really would be significant would be: "at least as *weak* as," but that is not what the article gives us.)

The implication seems to be that we should simply discard a useful rule of thumb that is *almost* always right, for a result *claimed* to be right which gives us nothing at all. But we would not do that even in theoretical mathematics! (See Method of Proof and Refutations.) Instead, we would seek the special-case conditions that make our statement false, and then integrate those into assumptions that support valid proof.

The example "ciphers" used in the "proof" are constructed so that each of two possible keys do produce different ciphertexts for two plaintexts, but not for two others. Then we assume that only two original plaintexts actually occur. In this case, for one cipher being "first," that key has no effect, and the resulting ciphertext is also not affected by the key in the second cipher. Yet if the ciphers are used in reverse order, both keys are effective.

However, knowing which cipher is "first" is only "important" when the first cipher results support attacks on the second cipher *and not vise versa*. But if we are allowed to create arbitrary weakness in examples, we probably can construct some that are mutually weak, in which case the question of which is "first" is clearly a non-issue, despite both the "result" and article title.

Both of the example ciphers are seriously flawed in that, for fully *half* of their plaintexts, changing the key does not change the ciphertext. Thus, for half their plaintexts, the example ciphers are essentially *unkeyed*. Since the example ciphers *start out* weak, I do not accept that either ordering has *reduced* the strength of the other cipher, and that is the main fear in using multiple encryption. Moreover, all we need to get a strong cascade is one *more* cipher, if it is strong. And that, of course, is the point of the "Folk Theorem."

The article has been used by some to sow "FUD" (fear, uncertainty and doubt) about multiple encryption. But multiple encryption in the form of product ciphering (in rounds and with *related* keys!) is a central part of *most* current block cipher designs. So before believing rumors of multiple encryption weakness, we might first ask why the block ciphers which use this technology seem to be trusted so well.

Other References

The first advantage of multiple encryption is to address the risk of the single point failure created by the use of a single cipher. Unfortunately, "risk of overall failure" seems to be a significantly different issue than the "keyspace size" or "needed known-plaintext" measures used in most related analysis. Unfortunately, it is in the nature of cryptography that the risk of cipher failure cannot be known by the cryptanalyst, designer, or user. The inability to know the risk is also the inability to quantify that risk, which leaves the analyst without an appropriate measure.

What we most seek from multiple encryption is *redundancy* to reduce risk of overall failure, a concept almost completely missing from academic analysis. The main issue is not whether multiple ciphers produce a stronger result when they all work. Instead the issue is overall security when one cipher is weak. For the single-cipher case a broken cipher means complete failure and loss of secrecy even if we are not informed. For the multi-cipher case to be a better choice, all the remaining ciphers need do is have *any strength at all*. One would expect, of course, that the remaining ciphers would be as strong as they ever were. And even if we assume that *all* the ciphers are weak, it is possible that their composition could *still* be stronger than the abject failure of a completely exposed single cipher.

Some examples of the literature:

> "Here, in addition to formalizing the problem of chosenciphertext security for multiple encryption, we give simple, efficient, and generic constructions of multiple encryption schemes secure against chosen ciphertext attacks (based on any component schemes secure against such attacks) in the standard model."
> --Dodis, Y. and J. Katz. 2005. Chosen-Ciphertext Security of Multiple Encryption. *Second Theory of Cryptography Conference Proceedings.* 188-209.
>
> "We prove cascade of encryption schemes provide tolerance for indistinguishability under chosen ciphertext attacks, including a 'weak adaptive' variant."
> "Most cryptographic functions do not have an unconditional proof of security. The classical method to establish security is by cryptanalysis i.e. accumulated evidence of failure of experts to find weaknesses in the function. However, cryptanalysis is an expensive, time-consuming and fallible process. In particular, since a seemingly-minor change in a cryptographic function may allow an attack which was previously impossible, cryptanalysis allows only validation of specific functions and development of engineering principles and attack methodologies and tools, but does not provide a solid theory for designing cryptographic functions. Indeed, it is impossible to predict the rate or impact of future cryptanalysis efforts; a mechanism which was attacked unsuccessfully for years may abruptly be broken by a new attack. Hence, it is desirable to design systems to be tolerant of cryptanalysis and vulnerabilities (including known trapdoors)."
> "Maurer and Massey claimed that the proof in [EG85] 'holds only under the uninterestingly restrictive assumption that the enemy cannot exploit information about the plaintext statistics', but we disagree. We extend the proof of [EG85] and show that, as expected intuitively and in [EG85], keyed cascading provides tolerance to many confidentiality specifications, not only of block ciphers but also of other schemes such as public key and shared key cryptosystems. Our proof uses a strong notion of security under indistinguishability test--under plaintext only and non-adaptive chosen ciphertext attack (CCA1), as well as weak version of adaptive chosen ciphertext attack (wCCA2). On the other hand, we note that cascading does not provide tolerance for adaptive chosen ciphertext attack (CCA2), or if the length of the output is not a fixed function of the length of the input."
> --Herzberg, A. 2004. On Tolerant Cryptographic Constructions. Presented in Cryptographer's Track, *RSA Conference 2005*.
>
> "In a practical system, a message is often encrypted more than once by different encryptions, here called multiple encryption, to enhance its security." "Intuitively, a multiple encryption should

remain 'secure', whenever there is one component cipher unbreakable in it. In NESSIE's latest Portfolio of recommended cryptographic primitives (Feb. 2003), it is suggested to use multiple encryption with component ciphers based on different assumptions to acquire long term security. However, in this paper we show this needs careful discussion." "We give the first formal model regarding public key multiple encryption."
--Zhang, R., G. Honaoka, J. Shikata and H. Imai. 2004. On the Security of Multiple Encryption or CCA-security+CCA-security=CCA-security? *2004 International Workshop on Practice and Theory in Public Key Cryptography.*

"We obtain the first proof that composition actually increases the security in some meaningful way."
--Aiello, W., M. Bellare, G. Di Crescenzo, R. Venkatesan. 1998. Security Amplification by Composition: The case of Doubly-Iterated, Ideal Ciphers. *Advances in Cryptology--Crypto 98.* 390-407. Springer-Verlag.

"We conjecture that operation modes should be designed around an underlying cryptosystem without any attempt to use intermediate data as feedback, or to mix the feedback into an interemediate round."
--Biham, E. 1994. Cryptanalysis of Multiple Modes of Operation. *Journal of Cryptology.* 11(1):45-58.

"Double encryption has been suggested to strengthen the Federal Data Encryption Standard (DES). A recent proposal suggests that using two 56-bit keys but enciphering 3 times (encrypt with a first key, decrypt with a second key, then encrypt with the first key again) increases security over simple double encryption. This paper shows that although either technique significantly improves security over single encryption, the new technique does not significantly increase security over simple double encryption.
--Merkle, R. and M. Hellman. 1981. On the Security of Multiple Encryption. *Communications of the ACM.* 24(7):465-467.

Advantages of Multiple Encryption

Multiple encryption can increase keyspace (as seen in Triple DES). But modern ciphers generally have enough keyspace, so adding more is not usually the looked-for advantage in using multiple encryption.

Multiple encryption reduces the consequences in the case that our favorite cipher is already broken and is continuously exposing our data without our knowledge. (See the comments on the John Walker spy ring in: security through obscurity.) When a cipher is broken (something we will not know), the use of other ciphers may represent the only security in the system. Since we cannot scientifically prove that any particular cipher is strong, the question is *not* whether *subsequent* ciphers are strong, but instead, what would make us believe that any particular cipher is so strong as to need no added protection.

Multiple encryption also protects each of the component ciphers from known plaintext attack. Since known plaintext completely exposes the ciphering transformation, it enables a wide range of attacks, and is likely to make almost any attack easier. Preventing known plaintext attacks has at least the potential to even make weak ciphers strong in practice.

With multiple encryption, the later ciphers work on the randomized "plaintext" produced as ciphertext by the earlier cipher. It can be extremely difficult to attack a cipher which only protects apparently random "plaintext," because it is necessary to at least find some structure in the plaintext to know that one has solved the cipher. See: Ideal Secrecy and unicity distance.

Cipher Standardization

Most of the protocols used in modern communications are *standardized*. As a consequence, most people do not question the need for standardization in ciphers. But in this way ciphers are once again very different than the things we know so well: The inherent purpose of ciphers is to *prevent* interconnection to almost everyone (unless they have the right key).

Obviously, it is necessary to describe a cipher clearly and completely if it is to be properly implemented by different people. But standardizing on a single cipher seems more likely to help the opponent than the user (see NSA). With a single cipher, an opponent can concentrate resources on one target, and that target also has the most value since it protects most data. The result is vastly increased user risk.

The alternative to having a standard cipher is to have a standard cipher *interface*, and then select a desired cipher by textual name from a continually-increasing list of ciphers. In whatever way we now transfer keys, we could also transfer the name of the desired cipher, or even the actual cipher itself.

Risks of Multiple Encryption

Multiple encryption can be dangerous if a single cipher is used with the same key each time. Some ciphers are involutions which both encipher and decipher with the same process; these ciphers will **de**cipher a message if it is **en**ciphered a second time under the same key. This is typical of classic additive synchronous stream ciphers, as it avoids the need to have separate encipher and decipher operations. But it also can occur with block ciphers operated in stream-cipher-like modes such as OFB, for exactly the same reason.

It is true that multiple encryption cannot be proven to improve security over having just a single cipher. That seems hardly surprising, however, since no single cipher can be proven to improve security over having *no cipher at all*. Indeed, using a broken cipher is far *worse* than no cipher, because then users will be mislead into not taking even ordinary precautions. And in real cryptography, users will not know when their cipher has been broken.

For more on this topic, see: superencryption; and the large sci.crypt discussions:

- "Fixing Strength Problems in Cipher Use" (644K) locally, or @: http://www.ciphersbyritter.com/NEWS4/LIMCRYPT.HTM
- "Risks of Relying on Cryptography Discussion" (840K) locally, or @: http://www.ciphersbyritter.com/NEWS5/RISKRELY.HTM
- "Cascade Ciphering" (72K) locally, or @: http://www.ciphersbyritter.com/NEWS6/CASCADE.HTM
- and my guest column in *IEEE Computer:* "Cryptography: Is Staying with the Herd Really Best?" (Copyright 1999, IEEE) locally, or @: http://www.ciphersbyritter.com/ARTS/R8INTW1.PDF

N

In math notation, the natural numbers.

National Cryptologic Museum

The museum of crypto artifacts located on the grounds near NSA headquarters, at Ft. Meade, Maryland, USA. On the web at http://www.nsa.gov/museum/index.cfm

The many exhibits include:

- a working demonstration Enigma machine,
- a multi-rack U.S. "Bombe" Enigma cracker,
- a little piece of the actual Japanese "Purple" machine,

- the "Purple" U.S. equivalent,
- SIGSALY, a WWII Presidential-level secure voice installation,
- various other WWII cipher machines, both German and U.S.,
- a KW-26 tube-based secure TTY machine,
- a KY-3 Vietnam-era solid-state Command-level secure voice box.

Located about halfway between Washington, D.C. and Baltimore, MD, just off Rt. 295 (the Baltimore-Washington Parkway), the Museum is about a half-hour out of Washington: For example, take I-95 N., to Rt. 32 E. Then, *just* past (by under 1/10th of a mile) the cloverleaf intersection with Rt. 295, take the next exit (Canine Rd.), and follow the signs. Or take the Baltimore-Washington Parkway N., and exit just past the I-95 cloverleaf. The Museum is clearly marked on the Maryland page of the Mapsco or Mapquest *Road Atlas* 2005.

CAUTION: Due to new construction, the wooded terrain, and the near proximity of Rt. 295, the sign indicating the Museum exit on Rt. 32 is *much* too close to the turn-off. So, after the Rt. 295 cloverleaf, end up in the right lane prepared for a turn-off coming under a tenth of a mile later.

Natural Number

The set: {0, 1, 2, 3, ...} and denoted **N**. The nonnegative integers. The counting numbers and zero. The natural numbers are closed under addition and multiplication, but not under division.

Negative Resistance

In electronics, the confusing idea that the resistance quality might take on a negative value, and thus generate power instead of dissipating it. As far as is known, that does not occur in reality. However some feedback circuits can be analyzed by considering energy from another part of the circuit as being created in a local negative resistance.

What *does* occur in reality is negative *incremental* or *differential* or ***dynamic*** resistance, where an *increase* in the voltage across an active device produces a *decrease* in current the device allows. That is the reverse of the normal resistance effect, and so is a "negative-like" region in the nevertheless overall positive effective resistance of the device. Some things which may have some amount of negative dynamic resistance include:

- Some avalanching PN junctions above 6 volts or so, including so-called "Zener" diodes;
- Any gas-discharge tube (e.g., neon bulbs, fluorescent lamps, etc.), which is why a "ballast" resistor or inductor is needed for stable use;
- A unijunction transistor;
- Some NPN transistors under reverse bias with base open (collector negative, emitter positive) in breakdown;
- A tunnel diode;
- A "Lambda Diode" (a P-JFET and N-JFET with their source terminals joined, and each gate connected to the drain of the opposite FET);
- The input impedance of a bipolar emitter-follower with a capacitive load (the input admittance is a capacitance shunted by a negative resistance which increases with frequency and so tends toward very high frequency oscillation; in the transistor era it was common to include a 10k base resistor in emitter-followers to avoid instability);
- A switching power supply, as seen from the power line side;
- Some amplifier circuits with feedback;
- Any "two-terminal" oscillator circuit;
- Circuits specifically designed to vary resistance inversely to some signal (e.g., see National Semiconductor AN-263, fig 5).

NIST

The National Institute of Standards and Technology, a U.S. Government department.

Noise

1: Anything which interferes with a desired signal.
2: The random analog electrical signal composed of a myriad of impulses from a quantum source, such as thermal noise in resistance, or current flow through a semiconductor PN junction. In cryptography, mainly used in the creation of really random or physically random generators or TRNG's.

The analog electrical noise produced in semiconductor devices is typically classified as having three sources:

- thermal noise (Johnson noise), the random motion of electrons in a resistance due to heat;
- shot noise, the independent launch timing and passage of individual electrons through a PN junction, due to applied voltage and the resulting current flow (e.g., avalanche multiplication, Zener breakdown and normal forward current flow); and
- >excess noise, typically 1/f noise.

In cryptography, noise is often deliberately produced as a source of really random values. Such noise is normally the result of a collected multitude of independent tiny pulses, which is a white noise. We say that white noise contains all frequencies equally, which actually stretches both the meaning of frequency as a correlation over time, and the noise signal as a stationary source. In practice, the presence of low frequency components implies a time correlation which we would prefer to avoid, but which may be inherent in noise.

I have attacked noise correlation in two ways:

- By rolling off the amplifier response below 1kHz with an RC filter. To the extent that noise energy is equally distributed across frequency, this removes only 1/20th of the noise signal.
- By subtracting each previous sample from the current one. This provides a major improvement as measured by autocorrelation testing.

See my articles:

- "Random Noise Sources" locally, or @: http://www.ciphersbyritter.com/NOISE/NOISRC.HTM
- "Experimental Characterization of Recorded Noise" locally, or @: http://www.ciphersbyritter.com/NOISE/NOISCHAR.HTM
- "Random Electrical Noise: A Literature Survey" locally, or @: http://www.ciphersbyritter.com/RES/NOISE.HTM
- "Measuring Junction Noise" locally, or @: http://www.ciphersbyritter.com/RADELECT/MEASNOIS/MEASNOIS.HTM
- and "Junction Noise Measurements I" locally, or @: http://www.ciphersbyritter.com/RADELECT/MEASNOIS/NOISMEA1.HTM

Nomenclator

Originally, a list of transformations from *names* to symbols or numbers for diplomatic communications. Later, typically a list of transformations from names, polygraphic syllables, and monographic letters, to numbers. Usually the monographic transformations had multiple or homophonic alternatives for frequently-used letters. Generally smaller than a codebook, due to the use of the syllables instead of a comprehensive list of phrases. A sort of early manual cipher with some characteristics of a code, that operated like a codebook.

Nominal

In statistics, measurements which are in categories or "bins." Also see: ordinal, and interval.

Nonce

Literally N_{once} ("N" subscript "once") -- a value *N* to be used only once. Also see message key and really random.

In general, a nonce is at least potentially dangerous, in that it may represent a hidden channel. In most nonce use, any random data value is as good as another, and, indeed, that is usually the point. However, by selecting particular values, nonce data could be subverted and used to convey information about the key or plaintext.

Since any value should be as good as any other, the user and equipment would never know about the subversion. Of course, the same risk occurs in message keys, and that does not mean we do not use message keys or other nonces.

Nonlinearity

The extent to which a function is not linear. See Boolean function nonlinearity.

Nonrepudiation

Accountability. A possible goal of cryptography. The idea that, a message, once sent, cannot later be denied.

Normal Distribution

The gaussian distribution. The usual "bell shaped" distribution discovered in 1733 by Abraham de Moivre, and further developed by Carl Friedrich Gauss 1777-1855. Called "normal" because it is similar to many real-world distributions. Note that real-world distributions can be *similar* to normal, and yet still differ from it in serious systematic ways. Also see the normal computation page.

"The" normal distribution is in fact a family of distributions, as parameterized by mean and standard deviation values. By computing the sample mean and standard deviation, we can "normalize" the whole family into a single curve. A value from any normal-like distribution can be normalized by subtracting the mean then dividing by the standard deviation; the result can be used to look up probabilities in standard normal tables. All of which of course assumes that the underlying distribution is in fact normal, which may or may not be the case.

NOT

A Boolean logic function which is the "complement" or the mod 2 addition of 1. The logic function called an inverter.

Novelty

Not obvious to one skilled in the art. See invention and patent.

NSA

The National Security Agency, the U.S. government bureaucracy charged with "making and breaking codes." Organized under the U.S. Defense Department, NSA designs cipher systems for the State Department and Army/Navy/Air Force, and breaks ciphers as needed for the CIA, FBI and other U.S. agencies. (Also see National Cryptologic Museum.)

The NSA is the frequent topic of cryptographic speculation in the sense that they represent the opponent, bureaucratized. They have huge resources, massive experience, internal research and motivated teams of attackers. But since NSA is a secret organization, most of us cannot know what NSA can do, and there is little fact beyond mere speculation. But it is curious that various convenient conditions do exist, seemingly by coincidence, which would aid real cryptanalysis.

Standardization

One situation convenient for NSA is that some particular cipher designs have been *standardized*. (This has occurred through NIST, supposedly with the help of NSA.) Although cipher standardization can be a legal requirement only for government use, in practice the standards are adopted by society at large. Cipher standardization is interesting because an organization which attacks ciphers presumably is aided by having few ciphers to attack, since that allows attack efforts to be concentrated on few targets.

When information is at risk, there is nothing odd about having an approved cipher. Normally, managers look at the options and make a decision. But NSA has *secret* ciphers for use by government departments and the military. They also *change* those ciphers far more frequently than the standardized designs.

Would a government agency risk tarnishing its reputation by knowingly approving a flawed cipher? Well, it is NIST, not NSA, that approves standard public ciphers. And if NSA neither designed nor approved those ciphers, exactly how could a flaw be considered a risk to them? Indeed, finding a flaw in a public design could expose the backwardness of academic development compared to the abilities of an organization which normally cannot discuss what it can do. That is not only not a risk, it could be the desired outcome.

Belief in Strength

Another situation which is convenient for NSA is that users are frequently encouraged to believe that their cipher has been proven strong by government acceptance. That is a reason to do nothing more, since what has been done is already good enough. Can we seriously imagine that NSA has a duty to tell us if they know that our standard cipher is weak? (That would expose their capabilities.)

Clearly, when only one cipher is used, and that cipher fails, all secrecy is lost. Thus, any single cipher is at risk of being a single point of failure. But, since risk analysis is a well known tool in other fields, it does seem odd that cryptography users are continually using a single cipher with no redundancy at all. The multiple encryption alternative simply is not used. The current situation is incredibly risky for users, yet oddly convenient for NSA.

The OTP

The OTP or one time pad is commonly held up as the one example of an unbreakable cipher. Yet NSA has clearly described breaking the VENONA cipher, which used an OTP, during the Cold War. It is argued that VENONA was "poorly used," but if a user has no way to guarantee a cipher being "well used," there is no reason for a user to consider an OTP strong at all.

It does seem convenient for NSA that a potentially breakable cipher continues to be described by crypto authorities as absolutely "unbreakable."

Null

Something without meaning or worth. A symbol or character with no information content, due to either value or position. Also see code.

Sometimes null characters are used to assure serial-line synchronization between data blocks or packets (see the ASCII character "NUL"). Sometimes null characters are used to provide a synchronized real-time delay when a transmitter has no data to send; this is sometimes called an "idle sequence." Similarly, block padding characters are sometimes considered "nulls."

A more aggressive use of nulls in ciphering is to interleave nulls with plaintext or ciphertext data, in some way that the nulls later can be removed. When nulls are distinguished by position, they can have random or even cleverly-selected values, and thus improve plaintext or ciphertext statistics, when desired. And if the nulls can be removed only if one has a correct key, nulls can constitute another layer of ciphering beyond an existing cipher. Of course, adding nulls does expand ciphertext. (Also see multiple encryption, transposition and braid.)

Null Distribution

In statistics, in general, when we randomly sample innocuous data and apply a statistic computation, we get some statistic value. When we do that repeatedly, the statistic values accumulate in some particular distribution which we can learn to recognize. So, for any particular statistic, a null distribution is what we find when sampling innocuous data. This is the "nothing unusual found" situation.

The p-value computation for a particular statistic typically tells us the probability of getting any particular statistic value or less (or more) in the null distribution. Then, if we repeatedly find *very unusual* statistic values, we can conclude either that our sampling has been very lucky, or that the statistic is not reproducing the null distribution. That would mean that we were not sampling innocuous data, and so could reject the null hypothesis. This is the "something unusual found" situation.

Null Hypothesis

In statistics, the *status quo*, or what we accept as true absent evidence to the contrary. Apparently introduced by Fisher in 1935. What we conclude from a statistical experiment when results seem due to chance alone. The particular statement or model or hypothesis H_0 which is accepted unless a test statistic finds "something unusual." In contrast to the alternative hypothesis or research hypothesis H_1.

Normally, the null hypothesis is just the statistical conclusion drawn when the pattern being tested for is not found. Normally, the null hypothesis cannot be proven or established by experiment, but can only be disproven, and statistics can only do that with some probability of error which is called the significance.

Statistical Experiments

A statistical experiment typically uses random sampling or random values to probe a universe under test. Those samples are then processed by a test statistic and accumulated into a distribution. Good statistical tests are intended to produce extreme statistic values upon finding the tested-for patterns.

Sometimes it is thought that extreme statistic values are an indication that the tested-for pattern is present. Alas, reality is not that simple. Random sampling generally can produce *any* possible statistic value even when no pattern is present. There are no statistic values which only occur when the tested-for pattern is detected. However, some statistic values are extreme and only occur *rarely* when there is no underlying pattern. To distinguish patterns from non-patterns, it is necessary to know how often a particular statistic result value would occur with unpatterned data.

The collection of statistic values we find or expect from data having no pattern is the null distribution. Typically this distribution will have the shape of a hill or bell, showing that intermediate statistic values are frequent while extreme statistic values are rare. To know just *how* rare the extreme values are, other statistic computations "flatten" the distribution by converting statistic *values* into *probabilities* or p-values. In this way each statistic result value can be associated with the probability of that value occurring when no pattern is present.

The probability that an extreme statistic value will occur when no pattern is present is also the probability of a Type I error which is usually a "false positive." Type I errors are a consequence of the randomness required by sampling, or a consequence of random values, and cannot be eliminated. Normally, random values are expected to produce sequences without pattern. Again, reality is not like that. Instead, over huge numbers of sequences, random values must produce *every possible* sequence, including every possible "pattern." Usually the statistical test is looking for a particular class of pattern which may not correspond to what we expect.

[When every sequence is possible, the probability of finding a "pattern" depends strongly on what we interpret as a pattern. It might be possible to get some quantification of pattern-ness with a measure like Kolmogorov-Chaitin complexity (the length of the shortest program to produce the sequence). But K-C complexity testing may have its own bias, and in any case there is no algorithm to find the shortest program.]

Any particular statistic value can be used to separate "probably found" from "probably not found." Typically, scientists will use a "significance" of 95 percent or 99 percent, but that is *not* the probability that the hoped-for "something unusual" signal has been found. Instead, the complement of the significance (usually 5 percent or 1 percent) generally is the probability that a statistic value so high or greater occurring from null data having no patterns. With a 95 percent significance, null data will produce results that falsely reject the null hypothesis in 5 trials out of 100. By increasing the significance, the probably that null data will produce an extreme result value is decreased, but never zero. When multiple trials show that the statistic measurements do not follow the null distribution, "something unusual" has been found, and the null hypothesis is rejected.

Randomness testing is a special case because there we hope to find the null distribution. Success in randomness testing generally means being forced to accept the null hypothesis, which is opposite to most statistical experiment discussions. Statistical test programs in the classic mold seem to say that some statistic extremes mean "a pattern is probably found," which would be "bad" for a random generator. Again, that is not how reality works. In most cases, a random number generator (RNG) is expected to produce the null distribution. Extreme statistic values are not only expected, they are absolutely required. An RNG which produces only "good" statistical results is bad.

The Hypothesis

If we check random data which has no detectable pattern, we might expect the null hypothesis to be always rejected, but that is not what happens. In tests at a 95 percent significance level, the null hypothesis should be accepted in 95 trials out of 100. However, in 5 trials out of 100, "something unusual" will be "found" and the null hypothesis rejected or discarded. 5 times out of 100, the logically contrary alternative hypothesis or research hypothesis H_1 will be accepted or supported, *even though nothing unusual is present.* This is the normal consequence of random values or random sampling and can be especially disturbing when the false positives occur early.

Normally, the alternative hypothesis or research hypothesis H_1 includes the particular signal we test for in the randomly-sampled data, but also includes any result *other* than that specified by the null hypothesis. It is, therefore, more like "something unusual found" than evidence of the particular result we seek. When the tested-for pattern seems to have been found, the null hypothesis is rejected, although the result could be due to a flawed experiment, or even mere chance (see significance). This range of things that may cause rejection is a motive for also running control trials which do not have the looked-for signal. Also see: randomness testing and scientific method.

A common approach is to formulate the null hypothesis to expect no effect, as in: "this drug has no effect." Then, finding something unexpected causes the null hypothesis to be rejected, with the intended meaning being that the drug "has some effect." However, many statistical tests (such as goodness-of-fit tests) can only indicate whether a distribution matches what we expect, or not. When the expectation is the known null distribution, then what we expect is nothing, which makes the "unusual" stand out. But in that case, even a poorly-conducted or fundamentally flawed experiment could produce a "something unusual found" result. Simply finding something unusual in a statistical distribution does not imply the presence of a particular quality. Instead of being able to confirm a model in quantitative detail, this formulation may react to testing error as a detectable signal.

Even in the best possible situation, random sampling will produce a range or distribution of test statistic values. Often, even the worst possible statistic value can be produced by an unlucky sampling of the best possible data. It is thus important to compare the *distribution* of the statistic values, instead of relying on a particular result. It is also important to know the null distribution so we can make the comparison. If we find a *different* distribution of statistic values, that will be evidence supporting the alternative or research hypothesis H_1.

When testing data which has no underlying pattern, if we collect enough statistic values, we should see them occur in the null distribution for that particular statistic. So if we call the upper 5 percent of the distribution "failure" (this is a common scientific significance level) we not only *expect* but in fact *require* such "failure" to occur about 1 time in 20. If it does not, we will in fact have detected something unusual in a larger sense, something which might even indicate problems in the experimental design.

If we have only a small number of samples, and do not run repeated trials, a relatively few chance events can produce an improbable statistic value even in the absence of a real pattern That might cause us to reject a valid null hypothesis, and so commit a Type I error.

When we see "success" in a very common distribution, we can expect that success will be very common. A system does not have to be all that complex to produce results which just *seem* to have no pattern, and when no pattern is detected, we seem to have the null distribution. Finding the null distribution is not evidence of a lack of pattern, but merely the failure to *find* a pattern. And since that pattern may exist only in part, even the best of tests may give only weak indications which may be masked by sampling, thus leading to a Type II error. To avoid that we can run many trials, of which only a few should mask any particular indication. Of course, a weak indication may be difficult to distinguish from sampling variations anyway, unless larger trials are used. But there would seem to be no limit to the size of trials one *might* use.

OAEP

Optimal Asymmetric Encryption Padding.

> Bellare and Rogaway, 1994. Optimal Asymmetric Encryption. *Advances in Cryptology -- Eurocrypt '94.* 92-111.

An encoding for RSA.

Object Code

Typically, machine language instructions represented in a form which can be "linked" with other routines. Also see source code.

Objective

1. In the study of logic, reality observed without interpretation. As opposed to subjective or interpreted reality.
2. A goal.

Occam's Razor

(Also "Ockham".) Given a some known facts, and a desire to find some relationship between those facts, inductive reasoning and other guesses can deveop any number of different models which predict the exact same facts. However, absent further testing and new facts, the simplest model is preferred because it should be easier to use.

Note that the simplest model is not necessarily *right*: Many simple models are eventually replaced by more complex models. Nor does Science expect practitioners to defer to a particular model just because it has been published: The issue is the quality of the argument in the publication, and not the simple fact of publication itself.

A recommendation for scientists might be: "When you have multiple theories which predict exactly the same known facts, assume the simpler theory until it clearly does not apply." That tells us to *test* the simple model first, and to choose a more complex model if the simple one is insufficient. Also see: scientific method.

Octal

Base 8: The numerical representation in which each digit has an alphabet of eight symbols, generally 0 through 7.

Somewhat easier to learn than hexadecimal, since no new numeric symbols are needed, but octal can only represent three bits at a time. This generally means that the leading digit will not take all values, and that means that the representation of the top part of two concatenated values will differ from its representation alone, which can be confusing. Also see: binary and decimal.

Octave

A frequency ratio of 2:1. From an 8-step musical scale. Also see decade.

OFB

The Output Feedback operating mode.

Ohm's Law

The relationship between voltage (E), current (I), and resistance (R) in an electronic circuit.

```
   E = IR, or I = E/R, or R = E/I.
```

Old Wives' Tale

A belief accepted on the basis of *story* or *authority* instead of evidence and deductive reasoning. This is a particular problem in cryptography, because it is difficult to test gossip when there is little absolute reality to test against.

Stories are almost universal in human society. Most stories are about someone doing something, and how that is going, or how it turned out. To a large extent, stories and gossip are how we learn to interact with the world around us. Because story-listening is so common, humans may be genetically oriented toward stories. Perhaps, before the invention of writing, valuable past experiences lived on in stories, and those who listened tended to live longer and/or better, and reproduce more.

But even if evolution has put gossip in our genes, it does not seem to have worried very much about the distinction between fantasy and reality. I suppose that would be a lot to ask of mere evolution. But, even in modern technology, many plausible-sounding stories are just not right, yet people accept them anyway.

Normally, Science handles this by testing the gossip-model against reality. But cryptography has precious little *reality* to test against. Accordingly, we see the accumulation of "old wives' tales" which are at best rules of thumb and at worst flat-out wrong. Yet these stories apparently are so ingrained in the myth of the field that mere rationality is insufficient to stop their progression (see cognitive dissonance).

Examples include:

- **Modern ciphers are *proven* secure.**
  (Alas, no. Virtually no proof of cipher strength survives the transition to practice in any useful way. Cryptography cannot *measure* cipher strength unless the cipher is broken, in which case we will not use it. Because we cannot measure cipher strength, we cannot trust the strength of *any* cipher.)

- **Most cryptographers believe that modern ciphers are "strong enough."**
  (If so, they overstep. Cryptography stands virtually alone in the modern world as a product whose goals cannot be quality assured. Cipher strength occurs only in the context of opponents who work in secret. And even the opponents only know what *they* can do; they do *not* know if others can do better. There *can be* no expertise on cipher strength.)

- **Surely AES is secure.**
  (There is no such knowledge. Just because academics cannot break AES does *not* mean that nobody else can. Ciphers are broken in secret, and real opponents are not going to announce when they succeed. AES could be broken now, and we would not know. In that case, we would continue to use AES while our secret information was exposed and exploited. Presumably that would continue until somebody finally *published* a successful attack. Better approaches include multiple encryption and frequent cipher changes or keyed selection as in Shannon's Algebra of Secrecy Systems.)

- **But it *would* take a lot of effort to break AES.**
(We cannot even count on *that*. Academic vetting and cryptanalysis do not develop a guaranteed minimum strength value. Nothing we know would prevent a new insight from leading to an efficient break. And then that attack might be implemented in a computer program which almost anybody could run.)

- **We need only one good cipher.**
(Unfortunately, there is no way to know whether we have a "good" one or not. There is, and probably can be, no proof of cipher strength in practice. That means any particular cipher may fail, and if we only have that one, we risk everything in a single point of failure. To use only one cipher is to risk everything on something we know from the outset cannot be trusted. An alternative is to have and use multiple ciphers and multiple encryption. Also see Shannon's Algebra of Secrecy Systems and NSA.)

- **If a cipher has been around for a while without any known attack, we can trust it.**
(This is the basis for much of current cryptographic dogma but is simply invalid logic: Our opponents may *have* an effective attack about which they have not told us. Not telling us is understandable, since if they did we would switch to some other cipher, which could ruin their success. So just because we do not *know* of an attack does not mean that no effective attack exists. And if an effective attack *can* exist without us knowing, surely it is wrong to have confidence in any claim that an effective attack does *not* exist. Indeed, such belief is exactly what our opponents would plant and encourage if they *did* have an effective attack.)

- **We can trust accomplished academics to vet our ciphers.**
(First of all, most academic vetting is volunteer effort, and we have no real idea how much time has been spent nor how comprehensive the investigation was. But the real problem is that there simply is, *and can be*, no academic expertise with respect to what our opponents can do, and that is what we need.)

- **We can trust cryptographic proofs because, ultimately, cryptography *is* mathematics.**
(Cryptography and mathematics are different fields. While math can be content to "protect" theoretical data, cryptography has to deal with real data, real machines and real opponents. At its best, mathematics can give us a convenient model of reality. At its worst, math can deceive with results from models which just do not apply to real use. Currently there seems to be a widespread lack of desire to improve models which do not predict correct results in practice. See: proof and one time pad.)

- **A real one-time pad is mathematically proven unbreakable.**
(Although many OTP's are secure in practice, the proof only applies to theoretical OTP's. In practice, an OTP can be broken whenever the pad becomes predictable by whatever means. There are at least two issues:
    1. There is no test that will certify a given sequence as unpredictable. Even though a sequence might *be* unpredictable, we cannot *prove* that, and absolute proof about the sequence is required to apply the OTP proof.
    2. One or both users simply cannot know that the system has been used properly, and that the pad has been produced properly. A proof does not help someone who cannot control all of the assumptions made in that proof.

  In practice, an OTP may be secure or not, just like other ciphers.)

- **Known plaintext exposure does not matter anymore.**
(Exposing plaintext which can be matched to ciphertext can be fatal to some ciphers. And although modern ciphers are *claimed* to not have that weakness, that is still only a claim, not proven fact. Indeed, some of the best known attacks, like Linear Cryptanalysis, generally *require* known plaintext, while others, like Differential Cryptanalysis, generally *require* the even more restrictive defined plaintext conditions. Preventing those conditions also prevents those attacks.)

- **Cryptanalysis is how we know the strength of a cipher.**
  (Conventional cryptanalysis can only tell us of the strength of a cipher when that cipher has been broken, and then we will not use it. Cryptanalysis tells us nothing of the strength of ciphers which are not broken, and those are the only ones we use. We simply cannot extrapolate academic lack-of-success to our opponents.)

- **Cryptographers agree that current ciphers are less of a security risk than the systems or people around them.**
  (Cryptographers probably "agree" about many wrong things. Obviously there is no way to know the probability of an opponent breaking our cipher, because our opponents do not tell us when they succeed, so we cannot develop probabilities. And since that probability cannot be known, it also cannot be compared to *other* unknown probabilities such as the risks of hardware or people failure.)

- **If learned academic cryptanalysts cannot break a cipher, neither can our opponents.**
  (We cannot expect to know our opponents. Thus, we are necessarily forced to consider that, sometimes, our opponents may outsmart even our most learned academics.)

- **The best way to protect information is to "put it all in one basket," as in one much-reviewed cipher.**
  (The problem is that we *cannot* "watch that basket." That is *impossible* due to the nature of the situation. We *cannot* expect to know when the cipher fails, because that happens in private, and the opponents will not tell us, for if they did, we would change the cipher. So if we only use one cipher, and that cipher fails, *all* of our information is exposed, and our new information will *continue* to be exposed until we change the cipher. A better approach is to use multiple encryption, and to change ciphers frequently so that only a small amount of information is protected by any particular cipher.)

- **By using an appropriate threat model, we do not need to know exactly how strong our ciphers are.**
  (Unfortunately, a threat model is not very useful for ciphers, because we typically want to protect all of the information we have, from anyone, under any possible attack, forever. Loosening the constraints is not very helpful either, because cryptography cannot correctly model cipher strength.)

- **When a cipher has a known weakness already, nobody should care if we add a little more.**
  (This is one of the unstated arguments in common simplification of the BB&S design. But *I* do care! Most modern ciphers do have an unfixable "hole" in the sense that an opponent may choose the correct key by accident. But that does *not* mean that it is OK to include *other* holes which *are* easily fixed, even if they are less likely.)

- **Public key ciphers are almost proven secure.**
  (Public-key ciphers are vastly more risky than most people realize, because they support man-in-the-middle (MITM) attacks. MITM attacks are worrisome because they do not require breaking any cipher. MITM attacks can be effective *even if the ciphers have been proven secure*. That places much of the burden for security on a complex, originally expensive, and continually costly certification infrastructure or PKI, which is often simply ignored. In contrast, secret key ciphers do not support MITM attacks.)

- **Secret key ciphers need too many keys for general use.**
  (Secret keys correspond well to the metal keys we all use and love, so if we can imagine handling the situation with metal keys, we can do the same with a secret key cipher. For example, if we want a particular group to use the same door, we give them all copies of the same metal key. If we want to communicate with a company, we hardly need a separate cipher key for every person there. And local secure storage does not even need key transport.)

- **We know a really-random generator works when it passes specified statistical randomness tests.**
  (Even weak, deterministic, statistical RNG's pass those same tests. See randomness testing.)

- **We know how much unpredictable information we get from a really-random generator by just calculating the entropy.**
  (The Shannon entropy computation measures the "information rate" or efficiency of a coding structure, not "surprise" or unpredictability. The computation produces the exact same result whether the measured values are predictable or unpredictable. But if we first certify that the information really is coming from a quantum source, then Shannon entropy can tell us how much information we have in the typical uneven distribution.)

- **If we ever need to, we could use the effects of chaotic airflow in disk drives as an unpredictable noise source.**
  (A PC system is surprisingly more complex than one might expect. Simply sampling a complex system can produce values which pass statistical tests. But just passing such tests cannot certify a really random source. Some problems include:
  1. It has not been demonstrated that the expected small variations can be detected in a the noisy but deterministic PC environment of memory refresh, hardware interrupts and multitasking context changes.
  2. It has not been demonstrated that variations in disk rotation occur which can be detected on an ordinary PC.
  3. It has not been demonstrated that any such rotational variations are due to airflow, let alone that the airflow is chaotic.

  And since chaos theory started as the unexpected ability to find pattern in dripping water and other things thought essentially random, even if disk variations *were* chaotic they could be largely predictable anyway.)

- **It was in a paper in the proceedings from a crypto meeting, so our interpretation must be true.**
  (Oh, if only things were that easy.)

- **Surely we can use the "laws of physics" to get provably random values.**
  (Science does *not* "prove" physical "laws," but instead develops models which "predict" experimental results. Current models are occasionally replaced by newer models which perform better. Clearly, if a model can be replaced it could not have been proven correct originally. In the end, Science does not and *can* not provide absolute certainty in harvesting unpredictable randomness from any physical source. There are physical sources which we *assume* to be random, much like we *assume* some ciphers are strong, and we may be wrong in either case.)

- **The CBC IV can be sent in the clear.**
  (Well, if we simply *assume* that any changes that occur during message transport will be detected by a message-level MAC, sending the IV in the open seems fine. But if the authentication is forgotten or broken, an exposed IV can lead to a deceptive MITM attack which would not otherwise be available. For best confidence, the CBC IV should be sent encrypted.)

- **Ciphertext cannot be compressed.**
  (If the ciphertext has been encoded for transmission in text form, as is common for email delivery, it almost certainly *can* be compressed into binary form.)

oLs

orthogonal Latin squares.

One-Sided Test

1. In statistics, a one-tailed test.
2. In statistics, a statistic computation sensitive to variation in one direction only (e.g., "above" or "below"). See Kolmogorov-Smirnov. As opposed to two-sided test.

Ideally, "one-sided tests" are statistic computations sensitive to variations on only one side of the reference. The two "sides" are not the two ends of a statistic distribution, but instead are the two directions that sampled values may differ from the reference (i.e., *above* and *below*).

When comparing distributions, *low* p-values almost always mean that the sampled distribution is unusually close to the reference.

On the other hand, the meaning of *high* p-values depends on the test. Some "one-sided" tests may concentrate on sampled values *above* the reference distribution, whereas different "one sided" tests may be concerned with sampled values *below* the reference. If we want to expose deviations *both* above *and* below the reference, we can use *two* appropriate "one-sided" tests, *or* a two-sided test intended to expose both differences.

One-Tailed Test

In statistics, a hypothesis evaluation with a rejection region on only one end of the null distribution. The "test" part of this is the critical value which marks the start of the rejection region, thus becoming the measure of the hypothesis. Sometimes called one-sided. In contrast to two-tailed test. Also see null hypothesis.

Fundamentally a way to *interpret* a statistical result. *Any* statistic can be evaluated at both tails of its distribution, because *no* distribution has just one tail. The question is not whether a test distribution *has* two "tails," but instead what the two tails *mean*.

When comparing distributions, finding repeated p-values near 0.0 generally mean that the distributions seem too similar, which could indicate some sort of problem with the experiment.

On the other hand, the meaning of repeated p-values near 1.0 depends on the test. Some one-sided tests may concentrate on sampled values *above* the reference distribution, whereas different "one sided" tests may be concerned with sampled values *below* the reference. If we want to expose deviations *both* above *and* below the reference, we can use *two* appropriate "one-sided" tests, *or* a two-sided test intended to expose differences in both directions.

Are "One-Tailed" Tests Inappropriate?

Some texts argue that one-tailed tests are almost always inappropriate, because they start out *assuming* something that statistics can check, namely that the statistic exposes the only important quality. If that assumption is wrong, the results cannot be trusted.

There is also an issue that the significance level is confusingly different (about twice the size) in one-tailed tests than it is in two-tailed tests, since two-tailed tests accumulate rejection from both ends of the null distribution.

However, sometimes one-tailed tests seem clearly more appropriate than the altermative, for example:

- In statistical quality control, a certain maximum defect-level may be guaranteed by contract. When incoming lots are sampled, abnormally low defect rates (at the other end of the distribution) are just not an issue.
- Ultra-low bacteria levels in drinking water are also not an issue.
- If we are looking for an improved medical treatment, the null hypothesis might be that the experimental treatment does no better than some known cure rate. If the null hypothesis cannot be rejected, we may not care if the experimental treatment is extraordinarily worse. (Of course, if it turns out that the experimental treatment is so bad it is killing people, we may care more than we thought we would.)

One Time Pad

1. The spy cipher based on small booklets of random decimal digits (details below).
2. The term of art rather casually used for two fundamentally different types of cipher:

a. **The Theoretical One Time Pad:** a theoretical random source produces values which are combined with data to produce ciphertext. In a theoretical discussion of this concept, we can simply **assume** *perfect* unpredictable randomness in the source, and this **assumption** supports a mathematical proof that the cipher is unbreakable. But the theoretical result applies to reality **only if we can prove the assumption is valid** in reality. Unfortunately, we cannot do this, because *provably* perfect randomness apparently cannot be attained in practice (see really random). So the theoretical OTP does not really exist, except as a goal. (This is not an issue of being unable to prove perfection when "good enough" is available. It is instead an issue of not being able to know when a randomness flaw exists that might be exploited. Again, see proof, but also randomness testing and science.)

b. **The Realized One Time Pad:** a really random source produces values which are combined with data to produce ciphertext. But because we can neither *assume* nor *prove* perfect, theoretical-class randomness in any real generator, this cipher does not have the mathematical proof of the theoretical system. Perhaps there is some unnoticed correlation in the sequence, or even some complex generating function which we do not expect, but which nevertheless exists and may be exploited by our opponents. Thus, a realized one time pad is **NOT *proven*** unbreakable, although it may in fact *be* unbreakable in practice. In this sense, it is much like other realized ciphers.

Also see my comments from various OTP discussions locally, or @:
http://www.ciphersbyritter.com/NEWS2/OTPCMTS.HTM

Sequence Predictability

Despite the "one-time" name, the most important OTP requirement is *not* that the keying sequence be used only once, but that the keying sequence be unpredictable. Clearly, if the keying sequence *can* be predicted, the OTP is broken, independent of whether the sequence was re-used or not. Sequence re-use is thus just one of the many forms of predictability. Indeed, we would imagine that the extent of the inability to predict the keying sequence is the amount of strength in the OTP. And the OTP name is just another misleading cryptographic term of art.

The one time pad sometimes seems to have yet another level of strength above the usual stream cipher, the ever-increasing amount of unpredictability or entropy in the confusion sequence, leading to an unbounded unicity distance and perhaps, ultimately, Shannon Perfect Secrecy. Clearly, if the confusion sequence is in fact an arbitrary selection among all possible and equally-probable strings of that length, the system would be Perfectly Secret to the extent of hiding which message of the given length was intended (though not the length itself). But that *assumes* a quality of sequence generation which we cannot prove but can only assert. So that is a just another scientific model which does not sufficiently correspond to reality to predict the real outcome.

In a realized one time pad, the confusion sequence itself must be random for, if not, it will be somewhat predictable. And, although we have a great many statistical randomness tests, there is no test which can *certify* a sequence as either random or unpredictable. Indeed, a random selection among all possible strings of a given length must include even the *worst possible* patterns that we could hope to find (e.g., "all zeros"). So a sequence which passes our tests and which we thus *assume* to be random may **not** in fact *be* the unpredictable sequence we need, and we can never know for sure. (That could be considered an argument for using a combiner with strength, such as a Latin square, Dynamic Substitution or Dynamic Transposition.) In practice, the much touted "mathematically proven unbreakability" of the one time pad depends upon an assumption of randomness and unpredictability which we can neither test nor prove.

Huge Keys

In a realized one time pad, the confusion sequence must be transported to the far end and held at both locations in absolute secrecy like any other secret key. But where a normal secret key might range perhaps from 16 bytes to 160 bytes, there must be as much OTP sequence as there will be data (which might well be megabytes or even *gigabytes*). And whereas a normal secret key could itself be sent under a key (as in a message key or under a public key), an OTP sequence *cannot* be sent under a key, since that would make the OTP as weak as the key, in which case we might as well use a normal cipher. All this implies very significant inconveniences, costs, and risks, well beyond what one would at first expect, so even the realized one time pad is generally considered **impractical**, except in very special situations.

There are some cases in which an OTP can make sense, at least when compared to using nothing at all. One advantage of any cipher is the ability to distribute key material instead of plaintext. Whereas plaintext lost in transport could mean exposure, key material lost in transport would not affect security. That allows key material to be securely transported at an advantageous time and accumulated for later use. Of course it also *requires* that key material transport be successfully completed before use. And the existence of a key material repository allows the repository to be targeted for attack immediately, before secure message transport is even needed.

A realized one time pad requires a confusion sequence which is as long as the data. However, since this amount of keying material can be awkward to transfer and keep, we often see "pseudo" one-time pad designs which attempt to correct this deficiency. Normally, the intent is to achieve the theoretical advantages of a one-time pad without the costs, but unfortunately, the OTP theory of strength no longer applies. Actual random number generators typically produce their sequence from values held in a fixed amount of internal state. But when the generated sequence exceeds that internal state, only a subset of all possible sequences can be produced. RNG sequences are thus *not* random in the sense of being an arbitrary selection among all possible and equally-probable strings, no matter how statistically random the individual values may appear. Of course it is also possible for unsuspected and exploitable correlations to occur in the sequence from a really random generator whose values *also* seem statistically quite random. Accordingly, generator ciphers are best seen as classic stream cipher designs.

Weakness in Theory

Nor does even a theoretical one time pad imply unconditional security: Consider *A* sending the same message to *B* and *C,* using, of course, two *different* pads. Now, suppose the opponents can acquire plaintext from *B* and intercept the ciphertext to *C*. If the system is using the usual additive combiner, the opponents can *reconstruct the pad* between *A* and *C*. Now they can send *C* any message they want, and encipher it under the correct pad. And *C* will never question such a message, since *everyone knows* that a one time pad provides "absolute" security as long as the pad is kept secure. Note that both *A* and *C* have done this, and they are the only ones who had that pad.

Even the theoretical one time pad fails to hide message length, and so *does* leak some information about the message.

Weakness in Practice, Including VENONA

In real life, theory and practice often differ. The main problem in applying theoretical proof to practice is the requirement to guarantee that each and every assumption in the proof absolutely does exist in the target reality. The main requirement of the OTP is that the pad sequence be unpredictable. Unfortunately, unpredictability is not a measurable quantity. *Nobody* can know that an OTP sequence is unpredictable. Users cannot test a claim of unpredictability on the sequences they have. The OTP thus requires the user to trust the pad manufacturers to deliver unpredictability when even manufacturers cannot measure or guarantee that. Any mathematical proof which requires things that cannot be guaranteed in practice is not going to be very helpful to a real user. (Also see the longer discussion at proof.)

The inability to guarantee unpredictability in practice should be a lesson in the practical worth of mathematical cryptography. Theoretical math feels free to assume a property for use in proof, even if that property clearly cannot be guaranteed in practice. In this respect, theoretical math proofs often *deceive* more than they inform, and that is not a proud role for math.

At least two professional, fielded systems which include OTP ciphering have been broken in practice by the NSA. The most famous is VENONA, which has its own pages at http://www.nsa.gov/docs/venona/. VENONA traffic occurred between the Russian KGB or GRU and their agents in the United States from 1939 to 1946. A different OTP system break apparently was described in: "The American Solution of a German One-Time-Pad Cryptographic System," *Cryptologia* XXIV(4): 324-332. These were real, life-and-death OTP systems, and one consequence of the security failure caused by VENONA was the **death by execution** of Julius and Ethel Rosenberg. Stronger testimony can scarcely exist about the potential weakness of OTP systems. And these two systems are just the ones NSA has told us about.

Apparently VENONA was exposed by predictable patterns in the key and by key re-use. At this point, OTP defenders typically respond by saying: "Then it wasn't an OTP!" But that is the logic fallacy of circular reasoning and tells us nothing new: What we *want* is to know whether or not a cipher is secure *before* we find out that it was broken by our opponents (especially since we may never find out)! Simply *assuming* security is what cryptography *always* does, and then we may be surprised when we find there was no security after all, but we expect *much more* from a security proof! We expect a proof to provide a guarantee which has no possibility of a different outcome; we demand that there be zero possibility of surprise weakness from a system which is mathematically proven secure in practice. Surely the VENONA OTP *looked* like an OTP to the agents involved, and what can "proven secure" possibly mean if the user can reasonably wonder whether or not the "proven" system really *is* secure?

Various companies offer one time pad programs, and sometimes also the keying or "pad" material. But random values sent on the Internet (as plaintext or even as ciphertext) are of course unsuitable for OTP use, since we would hope it would be easier for an opponent to expose those values than to attack the OTP.

The Spy Cipher

Typically, the "pad" is one of a matched pair of small booklets of thin paper sheets holding random decimal digits, where each digit is to be used for encryption at most once. When done, that sheet is destroyed. The intent is that only the copy in the one remaining booklet (presumably in a safe place) could possibly decrypt the message.

In hand usage, a codebook is used to convert message plaintext to decimal numbers. Then each code digit is added without carry to the next random digit from the booklet and the result is numerical ciphertext. In the past, a public code would have been used to convert the resulting values into letters for cheaper telegraph transmission.

Based on theoretical results, the practical one time pad is widely thought to be "unbreakable," a claim which is false, or at least only *conditionally* true (see the NSA VENONA practical successes above). For other examples of failure in the current cryptographic wisdom, see AES, BB&S, DES, proof and, of course, old wives' tale.

One-To-One

Injective. A mapping f: *X* -> *Y* where no two values *x* in *X* produce the same result *f(x)* in *Y*. A one-to-one mapping is invertible for those values of *X* which produce unique results *f(x)*, but there may not be a full inverse mapping g: *Y* -> *X*.

One Way Diffusion

In the context of a block cipher, a one way diffusion layer will carry any changes in the data block in a direction from one side of the block to the other, but not in the opposite direction. This is the usual situation for fast, effective diffusion layer realizations.

One Way Hash

Presumably, a hash function which produces a result value from a message, but does not disclose the message from a result value. But hash function irreversibility is not a special cryptographic property, being instead is a natural property of (almost) any hash when the information being hashed is substantially larger than the resulting hash value. (Also see polyphonic as a related concept.)

Many academic sources say that a "one way" hash must make it difficult or impossible to create a particular result value. That would be an important property for authentication, since, when an opponent can easily create a particular hash value, an invalid message can be made to masquerade as real. But it is not clear that we can guarantee that property any more than we can guarantee cipher strength. (Also see cryptographic hash and MAC.)

In contrast, many other uses of hash functions in cryptography do not need the academic "one way" property, including:

- Converting a user key phrase into a fixed-size key value,
- Protecting a linear (e.g., LFSR) sequence behind a hash result,
- Accumulating real randomness, and, of course,
- Fast table look-up.

(Also see crc.)

Onto

Surjective. A mapping f: *X* -> *Y* where *f(x)* covers all elements in *Y*. Not necessarily invertible, since multiple elements *x* in *X* could produce the same *f(x)* in *Y*.

```
+----------+          +----------+
|          |   ONTO   |          |
|    X     |          | Y = f(X) |
|          |    f     |          |
|          |   --->   |          |
+----------+          +----------+
```

Op Amp

Operational amplifier.

Opcode

Operation code: a value which selects one operation from among a set of possible operations. This is an encoding of functions as values. These values may be interpreted by a computer to perform the selected operations in their given sequence and produce a desired result. Also see: software and hardware.

Operating Mode

With respect to block ciphers, a way to handle messages which are larger than the defined block size. Usually this means one of the four block cipher "applications" defined for use with DES:

- ECB or Electronic Codebook;
- CBC or Cipher Block Chaining;
- CFB or Ciphertext FeedBack; and
- OFB or Output FeedBack.

It can be argued that block cipher operating modes are stream "meta-ciphers" in which the basic transformation is a full block, instead of the usual bit or byte.

Operational Amplifier

(Op amp). The general model of an ideal amplifier with:

- DC-coupling (works on slowly-changing signals, in addition to audio and higher frequencies),
- differential inputs (two input connections, + and -, with the actual input being the difference between the two),
- infinite common-mode rejection (a signal "cancels" when applied to both inputs simultaneously),
- infinite gain,
- infinite power-supply noise rejection, and
- protection against output short circuits.

The schematic symbol for an op amp is a triangle pointing right, with the two inputs at the left and the output on the right. Power connections come out the top and bottom, but are often simply assumed. Power is always required, but is not particularly innovative, and can obscure the crucial feedback path from OUT to -IN.

```
              +PWR
           | \ |
           |  \
   -IN   --| - \
           |     >--  OUT
   +IN   --| + /
           |  /
           | / |
              -PWR
```

Op amps were originally used to compute mathematical functions in analog computers, where each amplifier was an "operation."

In the usual voltage-mode idealization, each input is imagined to have an infinite impedance and the output has zero impedance (here the ideal output is a voltage source, unaffected by loading). In the the rarer current-mode form, each input is imagined to have zero impedance and the output has infinite impedance (that is, the output is a current source). Some current-feedback op amps for RF use have a low-impedance voltage output and low-impedance current inputs. Of course, no real device has anything like infinite gain, although op amp gain can be extremely high at DC.

Stability

One important feature of an op amp is *stability* (as in lack of spurious oscillation; see discussion in amplifier). Op amp transistors have substantial gain at RF frequencies, and unexpected coupling between input and output can produce RF oscillations. Unfortunately, these may be beyond the frequency range that a modest oscilloscope can detect, with the main indication being that the device gets unreasonably hot.

In the early days of IC op amps, the designer was expected to produce a feedback network for each circuit that included stability compensation to prevent oscillation. Nowadays, the IC manufacturer generally buys stability by rolling off the frequency response at the usual RC rate of 6dB/octave (20dB/decade) so that no gain remains at RF frequencies to allow oscillation. As a result, many modern general purpose op amps do not have a lot of gain at high audio frequencies. A few (generally older) op amps (like the TL081) do allow the designer to change the compensation and place the start of roll-off at a higher frequency. And while that can provide considerably more gain, it also carries considerably more risk of instability.

The usual cures for instability include first isolating the power supply, since that will go everywhere:

- Bypass RF and audio from op amp (+) power pin to (-) power pin, with a tantalum capacitor *right at the pins*. (Tantalum capacitors are somewhat lossy at RF frequencies, and in this case loss is good, to prevent power pulses from bouncing around in a LC system composed of discrete capacitors and interconnect inductance.)

- More power isolation is available with 10 to 100 ohm resistors and/or RF-lossy ferrite beads to decouple the op amp from the power supply.
- It is important that the signal input connections be very short so they do not pick up signal from the output, either by electrostatic or magnetic coupling.
- It is also possible to add small resistors on both inputs, and a tiny capacitor across the input pins, to form an RC lowpass filter and thus cut the RF level in the normal input path.
- The negative feedback resistor may be mounted right on the device pins, for the shortest possible connection.
- A tiny capacitor can be placed across the feedback resistor to cut high frequency response above the needed bandwidth.

In-Circuit Gain

One of the main advantages of op amps is an ability to precisely set gain with resistors and negative feedback. In an environment where the available devices have wildly different gain values, the ability to set gain precisely over all production devices is a luxury. If the feedback is purely resistive, and thus relatively insensitive to different frequencies, an op amp can be given a wide, flat frequency response even though the open-loop response typically droops by 6dB/octave (20dB/decade). By using reactive components (typically capacitors) in the feedback loop, the frequency response can be tailored as desired. Moreover, in general, whatever gain is available beyond that specifically programmed acts to minimize distortion. For example, if we want a gain of 20 decibels (20dB or 10x) at 20kHz, we probably want an op amp to have 40dB (100x) or more gain at 20kHz, so that 20dB remains to reduce amplifier distortion.

Operational amplifiers typically roll off high frequency gain at around 20dB/decade for stability. With that roll-off slope, the numerical product of gain and frequency is approximately constant in the roll-off region. A good way to describe this might have been "gain-frequency product," but the phrase actually used is "gain-bandwidth product" or GBW. The GBW is the frequency at which gain = 1, which is way beyond the useful region, since op amps are supposed to have "infinite" gain. (In practice, GBW varies with supply voltage, load, and measurement frequency, not to mention faster than expected rolloff in different designs, so the relation is approximate). To get 40dB (100x) of open-loop gain at 20kHz, we will need a minimum GBW of *about* 100x 20kHz or 2MHz. Exactly the same computation is used in bipolar transistors, where GBW is known as the "transition frequency," or f_T.

In-Circuit Input Impedance

In most cases, the positive op amp input is not part of the feedback system, and so has the normal high impedance expected of an op amp input. However, the negative op amp input almost always *is* part of the feedback system, which changes the apparent input impedance. High amounts of negative feedback act to keep the negative input at almost the same voltage as the positive input. If the positive input is essentially ground, the negative input is forced by feedback also to be essentially ground, often described as a *virtual* ground.

In most cases, external signals will see the negative input as a low-impedance ground, and this happens because of feedback, not op amp input impedance. Circuits which require inversion and so use the (-IN) input may:

- add an op amp simply to buffer the input signal,
- increase resistor values to reduce the problem, or
- use non-traditional feedback architectures, e.g., the LT1193.

The Linear Technology LT1193 has an extra pair of inputs for a total of 4: 2 positive and 2 negative. One (-IN) pin can handle feedback and in that way set the gain. One (+IN) pin can adjust output bias. That leaves another (-IN) and another (+IN), both high-impedance differential inputs unaffected by feedback, which seems ideal for balanced line receiving. Unfortunately, as a wideband video op amp, the LT1193 has a massive current budget (43mA) and high noise (50nV/sqrt-Hz).

Single-Supply Operation

Most op amp circuits show bipolar (that is, both positive and negative) power supplies referenced to a center ground. But few if any op amps have a ground pin, so they see only a single power circuit across the device whether bipolar supplies are used or not. The problem is that op amps have to be biased just like transistors: their output needs to rest between supply and ground or it will not be possible to represent both positive and negative signals. Even op amps with rail-to-rail input and output ranges cannot reproduce a negative voltage when operating on a single positive supply. Conventional op amps with a limited input voltage range may demand that the bias level be near half the supply. Often we need a low-noise, low-hum and sometimes even high-power voltage reference, typically at about half the supply voltage.

The usual way to get an intermediate voltage in a single-supply system is to use two similar resistors in series from power to ground. Unfortunately, this means that noise on the power lines will just be divided by two and then used on the input side of which could be a high-gain circuit. And the resistors will add thermal noise, and possibly resistor excess noise. We can reduce supply hum and noise by splitting the upper resistance into two and adding a serious capacitor to ground there. Another capacitor from the lower resistor to ground will act to filter out high frequency signals from Johnson white noise. Since most noise power is in high frequencies, removing those frequencies can reduce the effective noise level.

In contrast to Johnson noise, resistor excess noise is a 1/f noise and is highest at low frequencies, and power filtering may have little effect. Non-homogenous resistors will generate 1/f noise proportional to resistance and current, and can be a serious problem in very low-level circuits, especially below 100Hz. Fortunately, most resistors connected to op amp inputs will carry negligible current, making excess noise a non-issue there. Metal film resistors generate minimal 1/f noise and should be used in low-signal situations where there is significant current and resistance, such as bias voltage dividers.

Another bias alternative for single-supply operation is to generate some fixed moderate voltage level above ground instead dividing Vcc. In contrast to the usual voltage reference, a bias typically does not have to be an exact voltage, and can vary somewhat with time and temperature. For example, if power is to be a 9V battery, we expect to see only about 4.5V at the end of life. Half of that would be about 2.25V, but we might make do with an output bias of only 1.5V, provided that is within the input range of the op amp (a good reason to use an op amp with rail-to-rail I/O). We can develop about 1.5V with low noise from 3 silicon diodes carrying about 100uA of current to ground. Or we could get about 1.6V at low noise from a single red LED at low current. Or we could use a LM336 2.5V "IC zener" or some other sort of low voltage regulator, which would cost more, provide more voltage accuracy and stability than we need, and contribute substantial noise.

It is possible for the bias output from one stage to be the bias input to the next stage, which may simplify the system design.

Many op amp circuits connect the load between their output and the bias level, which is an issue for a relatively low-current bias supply. We might make a stronger bias supply by using another op amp to buffer the bias level. High power output op amps also may want to connect the load between utput and bias, which can make the bias supply even more exciting. An audio alternative might be to use a substantial capacitor to pass the output signal while stopping the DC bias level. When substantial capacitors are involved, it becomes important to deal with turn-off, since a charged capacitor can make the voltage at an op amp signal terminal much higher than the now-off supply. When significant DC output power is needed, bipolar supplies may be the best choice.

Most op amps actually sense signal potential between an input pin and a power pin, often the negative supply. With bipolar supplies, ground-referenced input signal current flow runs from input, to negative supply, through the negative filter capacitors, and back to ground. That makes power supply filtering part of the low-level input signal path in circuits with bipolar supplies. Normally this is not an issue, but is avoided with a single supply.

Biasing seems easier when ground is a low-noise reference between supply extremes so we can bias op amps to that level, even though that means generating bipolar supplies.

Capacitive Loads

Most op amps will have trouble driving a capacitive load. This is at least partly due to short-circuit protection which limits the output current to some reasonable level. The problem is that quickly changing the voltage across a capacitance may take a more current than the current-limiting will allow. So driving a discrete capacitor, or even a long cable, can present a challenge to an op amp. With a capacitive load, substantial current will be required for every voltage change, and because of internal delays, the op amp may tend to overrun the target voltage. Then it has to reverse itself, and may again overrun in the opposite direction. This is spurious ringing or even oscillation, and this form is typically addressed by decoupling the op amp output with a small resistor before the capacitive load, and perhaps a small capacitor from the op amp output to the inverting input.

Part Differences

There are hundreds of op amp designs. Many modern op amps are made for 20kHz audio, others for operation beyond 50MHz for video, and still others for use with RF signals and fast analog-to-digital (A/D) and digital-to-analog (D/A) conversion. It seems to be very difficult to build a very high input impedance wideband op amp with low noise that works at low supply voltages and currents and so could be used for almost anything. So there is no one best op amp, but there are various issues:

- Bandwidth (high-frequency response)
- Power (low voltage and current)
- Noise (a common issue, especially in audio, but substantially more complex than the simple voltage noise listed below to indicate design goals)
- Super-high input impedance (mainly FET-input devices)
- Rail-to-Rail (RRIO) (designs with an input common-mode range exceeding both supplies, and outputs that can be almost at supply levels)
- Input offset and drift

Specs often vary with different manufacturers, even for the same part number (especially the classic 741 devices). Here is a somewhat eclectic selection of specs for generally modest op amps (a list which may have some errors):

MFR	**PART**	**DESC**	**GAIN**	**POWER**	**NOISE**
			(dB@20kHz)	**(V, mA)**	**(nV/sqrt-Hz)**
AD	AD8052	dual RRO 70Mhz	70	3 to 12, 10	16
AD	OP27	bi low noise	50	10 to 40, 3	3.5
AD	OP421	quad bipolar	35	5 to 30, 1	30
AD	OP467	quad bi prec	50	9 to 36, 13	7
AD	OP184	bi prec RRIO	40	3 to 36, 1.45	3.9
BB	OPA627	JFET prec	60	9 to 36, 7.5	4.5
Fair	KM4170	bipolar RRIO	40	2.5 to 5.5, 0.19	21
Fair	KM7101	bipolar RRIO	40	2.7 to 5.5, 0.19	21
JRC	NJM4556	dual bi 70mA	50	4 to 36, 9	9
LT	LT1193	video (80MHz)	60?	4 to 18, 43	50
LT	LT1227	video current	55?	4 to 30, 10	3.2
LT	LT1361	dual bipolar	65	5 to 30, 10	9
LT	LT1881	dual bi RRO	25	2.7 to 36, 2	14
LT	LT1884	dual bipolar	40	2.7 to 36, 3	9.5
Mot	MC34001	JFET	40	8 to 36, 2.7	25
Mot	MC34184	quad JFET	40	3 to 36, 1	38
Nat	LM3900	quad current	35	4 to 32, 10	no spec!
Nat	LM324	quad bipolar	30	3 to 32, 1.2	no spec!
Nat	LF353	dual JFET	40	10 to 35, 6.5	16
Nat	LF356	JFET	40	10 to 40, 7	12

Nat	LF357	JFET	50	10 to 40,	7	12
Nat	LM358	dual bipolar	30	3 to 32,	2	no spec!
Nat	LM359	dual current	70	5 to 22,	22	6
Nat	LM837	quad bipolar	50?	8 to 34,	15	4.5
Nat	LM7171	bipolar 100mA	80	10 to 30,	9.5	14
Nat	LMC6482	dual CMOS RRIO	30	3 to 15,	2	37
PMI	OP421	quad bipolar	30	5 to 30,	1	40
Sig	NE5532	dual bipolar	55	6 to 40,	16	5
Sig	NE5533	dual bipolar	60	30 to 36,	10	4
Sig	NE5534	bipolar	50	8 to 30,	9	4.5
ST	TSH24	quad bipolar	40	3 to 30,	12	14
ST	TS462	dual bi RRO	52	2.7 to 10,	5.6	4
ST	TS522	dual bipolar	50	5 to 36,	5	4.5
ST	TS922	dual RRIO 80mA	44	2.7 to 14,	3	9
ST	L149	buffer 25W 4A	0	20? to 40,	30	no spec!
ST	L165	buffer 20W 3.5A	0	12 to 36,	60	2000
TI	TL072	dual JFET	40	10 to 30,	5	18
TI	TL074	quad JFET	40	10 to 30,	10	18
TI	TL081	JFET	40/85	10 to 30,	2.8	18
TI	TLC272I	dual CMOS	40	4 to 16,	3.6	25

Opponent

A term used by some cryptographers to refer to the opposing cryptanalyst or opposing team. Sometimes used in preference to "the enemy."

In academic cryptography, "the opponent" is just the mythical "other side" in the cryptography war. However, to the extent that opponents actually exist, they are practical enemies with their own goals. In practice, opponents presumably wish to:

1. break the cipher currently in use,
2. lead users to believe that the cipher can *not* be broken, so it will continue to be used, and
3. perhaps also try to misguide the open progress of cryptanalysis so that the cipher will not be exposed as weak and thus changed.

Note that items (2) and (3) represent an interference with the normal course of reality which perhaps could be identified by an intelligence officer. That could lead to the identification of a particular opponent, which would itself lead to a number of possibilities:

- the introduction of appropriate false information in the exposed channel;
- the development of countermeasures to prevent that channel from being exploited, possibly at particular times; and
- planning for the day the cipher will change.

Of course, other sucessful opponents also may exist, so simply identifying one of those may not be very useful in a security sense.

Option

In the study of logic, a decision between two hypotheses.

In "The Will to Believe" (1897), William James proposes that a hypothesis be called "live" if it seems to the listener to be a real possibility, and "dead" otherwise, as measured by "willingness to act."

He then proposes that an option be:

- **Living** if both hypotheses are live, otherwise *dead*;
- **Forced** if the two hypotheses represent the complete universe of possibilities, otherwise *avoidable*;
- **Momentous** for a unique, significant and irreversible selection, otherwise *trivial.*

James then calls a living, forced and momentous option "genuine." The original goal for these particular distinctions seems to have been to connect logic with religious faith and choice. But they may be useful in cryptography for very similar reasons.

In particular, a genuine option in cryptography is to use a particular cipher to protect a message. This is despite an admitted lack of proof or even valid evidence that the cipher will provide the security we want. The problem is that security is contextual, and we know neither our opponents nor the capabilities they possess. The forced decision is thus made on the basis of claims and irrational belief, so we have no reason to complain if our supposition of effectiveness is false.

OR

A Boolean logic function which is also nonlinear under mod 2 addition.

Order

In mathematics, typically the number of elements in a structure, or the number of steps required to traverse a cyclic structure.

- In a Latin square, the number of entries in a row, or in a column, or the number of unique symbols.
- For an element in a finite field, the number of repeated multiplications which take that element to unity. That is, the smallest power of an element which produces 1. (Since the next multiplication would reproduce the original element, this is the length of that cycle.)

Ordinal

A position in an ordered sequence. Also see: cardinal.

In statistics, measurements which are ordered by value. Also see: nominal, and interval.

Orthogonal

At right angles; on an independent dimension. Two structures which each express an independent dimension.

Orthogonal Latin Squares

(oLs). Two Latin squares of order *n,* which, when superimposed, form each of the n^2 possible ordered pairs of *n* symbols exactly once. At most, *n*-1 Latin squares may be mutually orthogonal.

```
   3 1 2 0   0 3 2 1       30  13  22  01
   0 2 1 3   2 1 0 3   =   02  21  10  33
   1 3 0 2   1 2 3 0       11  32  03  20
   2 0 3 1   3 0 1 2       23  00  31  12
```

Orthogonal Latin Squares of Order 4

Exactly 576 unique Latin squares exist at order 4, so there are 576 * 576 = 331,776 pairs which might be orthogonal. By checking each pair, we find that exactly 6912 (about 2 percent) are orthogonal (which hints at why finding such pairs is difficult). By examining the squares in each orthogonal pair, we find exactly 144 unique squares, each of which reduces to exactly the same standard form:

```
   0  1  2  3
   1  0  3  2
   2  3  0  1
   3  2  1  0
```

By permuting 4 rows and 3 columns (or 4 columns and 3 rows) of this square, we get a set of 144 expanded squares that is the same as the set of unique squares found from the orthogonal pairs. Thus, we can produce every square which participates in an orthogonal pair simply by permuting this one square. Each of the 144 resulting squares has 48 orthogonal partners, so exactly 1/3 of the squares are orthogonal partners for any particular square.

Constructing Orthogonal Pairs of Order 4

One way to construct an orthogonal pair of order 4 is to first build a randomized Latin square, by starting with the orthogonal base standard square (above), and shuffling *n* rows and *n* - 1 columns. Then we build other squares repeatedly, in the same way, until some pair is orthogonal. We can either throw away unsuccessful attempts, or save them for use later.

Another option is to expand the orthogonal base standard square into the complete list of the 144 squares which have orthogonal partners. If we naively use 16 bytes per square for that, the list will be 144 * 16 = 2304 bytes long. Then we can pair each of those with each other square, checking for orthogonality 144 * 144 = 20,736 times, to make a list of the indexes of successful orthogonal partners for each square. That list is only 144 * 48 = 6912 bytes long.

Once we have the lists in place, to make an orthogonal pair at order 4 we need only choose one index from 0 through 143 which directly selects one square, and another from 0 through 47 which selects an index to a guaranteed orthogonal partner. This direct construction does not need or use shuffling or orthogonal testing and always succeeds. Each result is one particular selection from among 6912, which represents about 12.75 bits of keying.

As a Balanced Block Mixer, an order-4 oLs mixes 2 bits with 2 bits and produces a 4-bit result. We thus need two of these mixings per byte for each sublayer of the FFT-style mixing. Each order-4 table has 16 entries of 4 bits each, for a total of 8 bytes per table.

Constructing Orthogonal Pairs of Order 16

Orthogonal pairs of small tables can be used to build orthogonal pairs of larger tables using the fast checkerboard construction. In particular, 17 pairs of order-4 tables can build a single order-16 oLs for Balanced Block Mixing 4 bits with 4 bits and producing an 8-bit result. Such an order-16 table would need 256 bytes of storage and represent about 216.75 bits of keying. Then we shuffle the table in 16! * 15! ways with another 84.5 bits, for a total of about 300 bits (as compared to the 1684 bits of keying in the usual 256-entry table of exactly the same size). As a BBM, an order-16 oLs pair mixes two 4-bit nybbles into a pair of mixed nybbles. For use in a Mixing Cipher, ideally we would have an independent mixing for each of *n* bytes across the block, for each of log2(*n*) FFT-like sublayers.

Constructing Orthogonal Pairs of Order 256

We can build an order-256 pair with only 257 order-16 tables. The resulting table would require 128k bytes for storage, and imply about 77,100 bits of keying, before being shuffled. Shuffling would add another 3,360 bits, for a total of about 80,460 bits (as compared to about 954,000 bits in a 64k-entry table). As a BBM, an order-256 pair mixes two bytes into a pair of mixed bytes, and so we need only one mixing for every two bytes across the block, for each layer, but with one less layer than for nybble mixing.

For the fast checkerboard construction, see:

- my article "Orthogonal Latin Squares, Nonlinear Balanced Block Mixers, and Mixing Ciphers," locally, or @: http://www.ciphersbyritter.com/ARTS/NONLBBM.HTM

Oscillator

A generator of a repetitive signal; a fundamental building block of electronics which marks the passage of time. Sometimes in contrast to an amplifier.

Often used to provide a digital "edge" or clock to time or regulate digital logic operations. Approaches range from relatively unstable RC (resistance-capacitance) "relaxation" designs, through fairly stable LC (inductor-capacitor) sine wave designs, to highly stable crystal oscillators, some of which convert an internal sine wave to square waves for digital use.

Oscillation occurs when:

- an amplified signal finds its way back to the amp input; AND
- the gain through the amplifier and feedback exceeds 1.0; AND
- the total phase shift around the feedback loop is 360 degrees.

OTP

One Time Pad.

Output Feedback

OFB. Output Feedback is an operating mode for a block cipher.

OFB is closely related to CFB, and is intended to provide some of the characteristics of a stream cipher from a block cipher. OFB is a way of using a block cipher to form a random number generator. The resulting pseudorandom confusion sequence can be combined with data as in the usual stream cipher.

OFB assumes a shift register of the block cipher block size. An IV or initial value first fills the register, and then is ciphered. Part of the result, often just a single byte, is used to cipher data, and *also* is shifted into the register. The resulting new register value is ciphered, producing another confusion value for use in stream ciphering.

One disadvantage of this, of course, is the need for a full block-wide ciphering operation, typically for each data byte ciphered. The advantage is the ability to cipher individual characters, instead of requiring accumulation into a block before processing.

Overall Diffusion

That property of an ideal block cipher in which a change of even a single message or plaintext bit will change every ciphertext bit with probability 0.5. In practice, a good block cipher will approach this ideal. This means that about half of the output bits should change for any possible change to the input block.

Overall diffusion means that the ciphertext will appear to change at random even between related message blocks, thus hiding message relationships which might be used to attack the cipher.

Overall diffusion can be measured statistically in a realized cipher and used to differentiate between better and worse designs. Overall diffusion does not, by itself, define a good cipher, but it is *required* in a good conventional block cipher.

Also see diffusion, avalanche, strict avalanche criterion, complete, ideal mixing and Balanced Block Mixing.

P-Value

In statistics, the probability of finding a particular statistic value or less (or greater) from the null distribution, with "nothing unusual found." "Something unusual" is indicated by the repeated occurrence of unusual statistic values, which implies that the distribution being sampled is not the null distribution. See null hypothesis.

Statistics functions are used to convert a raw statistic distribution into the a probability (or p-value) ranging from zero and one. See the "Normal, Chi-Square and Kolmogorov-Smirnov Statistics Functions in JavaScript" page, locally, or @: http://www.ciphersbyritter.com/JAVASCRP/NORMCHIK.HTM.

Padding

1. In classical cryptography, random data added to the start and end of messages so as to conceal the length of the message, and the position where coding actually starts.
2. In more conventional computing, some additional data needed to fill-out a fixed-size data structure. This meaning also exists in cryptography, where the last block of a fixed-size block cipher often must be padded to fill the block. Also see null.

Paradox

Something which seems self-contradictory. Possibly a statement which expresses a deeper truth.

Parallel

In electronics, components connected such that the same voltage is applied to all, while current divides though each according to their resistance or impedance. As opposed to series.

Parity

1. In data communications, data processing, digital design and computer design, the mod 2 sum of the data bits in a single code value. Typically, the exclusive-OR of every bit in a data byte, represented as an additional bit. This is a limited form of error-detection which will expose any odd number of data bit-errors occurring in communication or storage.
2. In mathematics, odd versus even; the least significant bit of a binary value.

Password

Typically, a key, in the form of a word. Originally a memorized word used to allow safe passage through a security checkpoint, such as a battlefield front line. See: user authentication and keyphrase, for multiple-word keys.

Patent

1. The legal right, formally granted by a government, to exclude others from **making**, **importing**, **offering for sale**, **selling**, or **using** the specific invention described in the patent deed. Note that "selling" is generally understood to cover even free distribution. Note that "making" is generally understood to include even personal construction without sale, if that results in continued operation or "use" (as opposed to transient educational research).
2. The patent grant deed itself. A patent document defines the protected invention and contains knowledge and information about operation.

General Patent Coverage

Patents are *not* about protecting the idea of an invention from publication, discussion, analysis or criticism. Indeed, a patent document is a deeply detailed description which is deliberately open to be read: patents are *published.* It is intended that patent documents provide a basis for improved understanding, further research and invention. Talking or writing about a patent is not infringement.

What patents restrict is not knowledge or speech, but instead *operation* of what is ultimately some form of physical machine. The steps in manufacturing processes have long been protected by patent; in some cases we might now see these step-descriptions as "mere algorithms." Writing mathematical equations is not covered by patents, but using particular equations to guide a real machine to do some real thing just might be.

In the U.S. there are three types of patent: *plant patents*, *design patents* and *utility patents*; here we discuss **utility patents**.

Advantages to Open Society

An issued patent is the open *publication* and *disclosure* of an invention, in return for a limited-term monopoly on its use. Patenting is thus the open alternative to secret development and trade secrecy.

If every inventor wished to simply give away the results of his or her work, patents probably would not exist. It is easy enough simply to publish inventions, and publication also prevents future patenting (unless somebody has already started the process). But inventors and employers often wish to retain an advantage, and their first response is to keep the invention secret. Unfortunately, when organizations holding trade secrecy disband or fail or succumb to disaster, what secrets they have often are lost with them. More than that, trade secrecy prevents others from building on the invention to make something even better, and can last indefinitely. Patenting thus aids society to an extent that the monopoly grant encourages open disclosure instead of secrecy. Patents mean an invention is not lost if an organization fails, the idea is available for future development, and the invention itself becomes available for use without restriction after a modest fraction of a lifetime.

Patenting also aids society to the extent that the monopoly grant supports the investment needed for commercialization, including development, production and sales, leading to eventual advantageous use by the consumer. Moreover, the U.S. PTO is funded by fees, not taxes, and issued patents which are not worth using do not stop anyone from doing what they want to do. From a purely selfish point of view, it is true that active patents can be more of a problem for a designer than just doing whatever one wants any time one wants. The increased cost to the design process is thus weighed against the advantage to society of open publication instead of secrecy. Inventors who cannot be bothered either to publish or patent do not establish the prior art which would prevent a patent from issuing. More seriously, selfish inventors do not inform society in general so the invention could be understood and built upon.

Patents versus Copyright

A patent is said to protect the *application* or operation or functioning of an idea (as opposed to the idea itself), and is distinct from copyright, which protects a particular *fixed expression* of an idea. Copyright typically protects particular words and sentences, not ideas or operation. When copyright protects the text and diagrams describing an idea, even different text and diagrams describing *the exact same idea* are not protected, let alone the idea itself. In contrast, a patent can protect many, most or even *all* implementations of an invention, but still not the idea itself. (Also see intellectual property, patent claims, prior art and patent infringement, as well as software patent and patenting software.)

Legal Background

In the United States, the basis of patent and other intellectual property law is Article 1, Section 8 of the Constitution (Powers of Congress):

> "Congress shall have power . . . To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries."

The concept behind patenting is to establish intellectual property in a way somewhat related to a mining claim or real estate. An inventor of a machine or process can file a claim on the invention, provided that it is not previously published, and also provided that someone else does not already have such a claim. Actual patents normally do not claim an overall machine, but just the newly-innovative part, and wherever that part is used, it must be licensed from the inventor. It is common for an inventor to refine earlier work patented by someone else, but if the earlier patent has not expired, the resulting patent often cannot be practiced without a license from the earlier patent holder.

Publication

Someone who comes up with a patentable invention and wishes to give up *their* rights can simply publish a full description of the invention. Simple publication should prevent an application from anyone who has not already established legal proof that they previously came up with the same invention. In the U.S., publication also apparently sets a 1-year clock running for an application to be filed by anyone who *does* have such proof. But coming up with an invention does not take away *someone else's* rights if they came up with the same thing first: they may have a year to file, and their case might take several years to prosecute and issue.

In the U.S., a patent is a non-renewable grant, previously lasting 17 years from issue date, now lasting 20 years from application date. Much like European patents, U.S. patents are now published 18 months after filing, which is typically before a patent examiner has reviewed the application. (Apparently an applicant can elect at application time to not allow early publication, but only if foreign applications will not be filed.) Both application and issue fees are required, as are periodic "maintenance" fees throughout the life of the patent.

Since patent applications cannot be filed in most countries after open publication, U.S. patent law establishes a simple "provisional application" which can establish a filing date, after which the invention can be published without loss of rights in other countries. The provisional must be followed within a year by a normal patent application. In practice, however, it is often the rigor of constructing the formal application which reveals the true novelty of an invention. And if the provisional does not include sufficient evidence for the things which turn out to be most important in the future application, those things will not be covered by the early filing date. Thus, going the provisional route can be risky.

Requirements

To obtain a patent, there are four main requirements (this is a simplified summary):

1. **Statutory Class** (35 USC 101): The invention must be either:
   - a process,
   - a machine,
   - a manufacture,
   - a composition of materials, or
   - a new use for one of the above.

2. **Utility** (35 USC 101): The invention must be of some use.

3. **Novelty** (35 USC 102): The invention must have some aspect which is different from all previous inventions and public knowledge.

   A U.S. patent is **not** available if -- **before the *invention* date** -- the invention was:
   - Publicly known or used the United States of America, or
   - Described in a printed publication (e.g., available at a public library, see **prior art**) anywhere

   (35 USC 102(a)).

   A U.S. patent is **not** available if -- **more than a year before the *application* date** -- the invention was:
   - In public use or on sale in the United States of America, or
   - Described in a printed publication anywhere

   (35 USC 102(b)).

   Thus, an inventor can obtain a U.S. patent even after publishing the details of an invention, simply by applying up to a year later. However, since publication will prevent patenting in most other countries, one approach is to establish a U.S. application date by submitting either a full or provisional U.S.

application, and publishing thereafter.

4. **Unobviousness** (35 USC 103): The invention must have **not** been obvious *to someone of ordinary skill* in the field of the invention *at the time of the invention.* Unobviousness has many arguments, such as:
    - Unexpected Results,
    - Unappreciated Advantage,
    - Solution of Long-Felt and Unsolved Need, and
    - Contrarian Invention (contrary to teachings of the prior art),

   among many others.

Ownership

When the same invention is claimed by different inventors, deciding who has "priority" to be awarded the patent can require *legally provable* dates for both "conception" and "reduction to practice":

- *Conception* can be proven by disclosure to others, preferably in documents which can be signed and dated as having been read and understood. The readers can then testify as to exactly what was known and when it was known.
- *Reduction to Practice* may be the patent application itself, or requires others either to watch the invention operate or to make it operate on behalf of the inventor. These events also should be carefully recorded in written documents with signatures and dates.

"In determining priority of invention, there shall be considered not only the respective dates of conception and reduction to practice of the invention, but also the reasonable diligence of one who was first to conceive and last to reduce to practice . . ." (35 USC 102(g)).

Also see the discussions:

- "What's the Meaning of 'Invent'?," locally, or @: http://www.ciphersbyritter.com/NEWS4/FSTINVNT.HTM
- and "Patent Notebook Consequences," locally, or @: http://www.ciphersbyritter.com/NEWS4/PATNOTBK.HTM

Coverage

The limits to a patent monopoly grant are found in the claims. Each claim defines some particular range of coverage. To find some art in infringement, it is *not* necessary that *every* claim apply to that art, only that *one* claim apply fully. A claim is said to "read on" infringing art when each requirement and limitation in the claim is satisfied. Note that the actual art may have a great deal of additional elaboration which is not relevant; infringement only requires that the claimed material be *present*, not *exclusive*.

Ideally, a patent rewards the inventor for doing research and development, and then *disclosing* an invention to the public, as opposed to keeping it secret. A patent is a way of protecting an idea for future exploitation, especially in the context of business funding. But a patent is only effective when others want to use the covered material: If the patented invention is not worthwhile and competitive, nobody will want to use it, so that particular patent should not be a bother. Most patents do not earn back their application fee, do not form a useful monopoly and so do not prevent anyone from doing anything they want to do.

If someone infringes a patent in a way which affects sales, or which implies that the inventor cannot do anything about it, the patent holder can be expected to show some interest. But when little or no money is involved, a patent can be infringed repeatedly with little or no response, and typically this will have no effect on future legal action.

This simple introduction cannot begin to describe the complexity involved in filing and prosecuting a patent application. Your author does *not* recommend going it alone, unless one is willing to put far more time into learning about it and doing it than one could possibly imagine. Find and use a patent attorney you can trust. (Also see patent changes, patent complaints, patent consequences, patenting software, and patent reading.)

Patent Changes

World intellectual property "harmonization" agreements resulted in various U.S. patent law changes in the 1990's:

- The old 17-year grant from issue date was changed to a 20-year grant from application date.
- A new "provisional application" was created to establish a filing date in advance of a formal application.
- Most applications are published about 18 months after filing, generally before first examination.

Patent Claims

A patent *claim* defines the limits of the temporary monopoly granted by the patent. The meaning of the words in the claims is known and fixed either by definition or use in the body of the patent itself. To infringe a claim, every aspect of that claim must occur in the infringing device. Although a typical patent has many claims, infringing any *one* claim is sufficient to infringe the patent. (Also see patent reading.)

When we look at patent claims, we notice a distinction between those which reference some earlier claim, and a few claims which have no such reference. Claims which do *not* refer to other claims are called "independent," and carry the broadest coverage because they are constructed to have the fewest requirements and limitations. (A patent holder typically wants short claims with the fewest restrictions possible, thus providing the greatest possible coverage.) Claims which *do* refer to other claims are called "dependent" claims, and these *include* the requirements and limitations of some other claim, further narrowed with *additional* requirements and limitations.

The reason for having *dependent* claims is that it is impossible in practice to know everything which was published or in common use anywhere in the world. Therefore, a broad claim may well someday be found to have been "anticipated," and ruled unenforceable. In this case the patent holder hopes that the increasingly fine restrictions of the dependent claims will remain unanticipated and, thus, still in effect. The Patent and Trademark Office (PTO) tries to limit patents to typically three independent claims.

Patent Claims and Science

Patent claims (and, by association, patents themselves) often seem scientifically hopeless to the uninitiated. But claims are just the ownership part of a patent. The much larger scientific part is disclosed everywhere else: in the Abstract, the Drawings, and in the Background, the Figures, the Summary, the Description of the Preferred Embodiment, and other sections in the Body. Claims are not about how to build the invention, nor are they about every detail which is present in practice.

Sometimes a novice at reading patent claims sneers that the claimed structure is insufficient to work in practice, and asserts that any such patent is worthless and should not have been granted. Such criticism, though, is completely wrong-headed. Claims are not intended to provide all implementation details, even those required for effective use. Instead, claims establish the minimum novel structure which is *covered* by the patent, and thus provide a compact description of ownership. When the described structure is found in a presumably far more complex device, the invention is present. Claims with minimum structure thus cover far more deviant designs and so establish a more effective and worthwhile monopoly than detailed claims.

Claims are about *ownership,* which is what the patent holder gets from the deal. Claims also describe the *limits* of ownership, which is the extent to which the patent could intrude upon society. But the larger part of the patent document addresses how to build the invention and the scientific worth, which is what society gets. And if, in the end, the invention does not have much value in the marketplace, the patent is not much of a limitation on design freedom.

Patent Complaints

Complaints about "software patents" inevitably criticize particular patent grants.

I Invented that Years Ago!

Anybody can *say* they invented something earlier, and they can believe it. But for any such statement to be respected under law requires proof beyond a reasonable doubt that would stand up in court. An issued patent is presumed valid until actually proven otherwise.

One issue is that a prospective patentee is required to move toward making a formal patent application. Someone who does not make the effort (and pay the cost) to apply for a patent obviously will not get it. But someone who keeps an invention secret has essentially chosen trade secrecy over patent protection. One cannot have both. The choice of trade secrecy precludes that inventor from getting a patent, and does not generate the prior art which would preclude others from patenting exactly the same thing.

Another issue is the exact definition of the invention. Someone just talking about an invention may have a general idea for an invention. But a general idea is not the level of detail required for a patent. Often, specific precise wording in the claims spells the difference between a patent grant and rejection. Mere handwaves in conversation do not equate to a patent.

The next issue is proving an exact date that an invention was known. Patents are not infrequently granted or refused on the basis of a few days difference in application or reduction to practice. Proving that a specific, precise invention existed as of an exact date without formal documentation is very difficult.

One form of legal proof for definition and date would be a prior patent, but then the issues described here already would be understood. Another form of legal proof would be a published article, thus establishing what was known when published. Yet another form of legal proof would be detailed written disclosures signed and dated by multiple knowledgeable associates who could testify under oath as to what was known and specifically when. Mere anecdotal descriptions of old conversations, even if "well known," seem unlikely to be sufficient as legal proof in the U.S., Europe, or any formal legal system whatsoever.

Novelty

One of the common complaints about controversial patents is that the invention was already known. But was it really that same invention? And by whom was it known? And exactly when did that occur?

Patent claims are not invalidated by outraged anecdotal evidence, even of personal work. Patents are legal documents, and generally require documentary evidence about who knew what and when. A formal patent application is one example of fixing what was known when by whom; a published article is another. Patent invalidation usually requires prior art in the form of another patent or an article, fully describing the same invention, and doing so a year or more before the supposedly invalid application.

Patent claims are also not invalidated by secret actions of an elite group, but instead by published prior art that, in the ideal, democratically informed the whole society. Prior art is not just a previous incarnation of the same invention. Something somebody did at school or work is normally not prior art unless it somehow informed the average worker in that field. For a patent claim to be ruled invalid, something other than the patent must have done the publication job at a time before a new inventor secured ownership rights by invention and patent application. Thus, the usual prior art is a printed description in a patent, magazine or journal article, available in public libraries. In the U.S., typically, the art must have been published a year or more before the application date.

Unobviousness

Another common complaint about controversial patents is that the patent is trivial or "obvious." For some reason it is generally assumed that the examiner just sits back and somehow decides whether or not something is obvious, based on personal experience and knowledge. In reality, unobviousness is *formally argued by the applicant*.

Apparently the courts and the PTO use a four-part obviousness test:

- What is the scope and content of the prior art?
- What is the ordinary skill level in the field?
- What are the differences between the prior art and the application subject matter as a whole?
- What is the objective evidence of unobviousness: Is the invention:
  - A solution to a long-felt need?
  - A formerly unappreciated advantage?
  - Successful where others failed?
  - Such that experts are skeptical?
  - In a direction away from the current literature (teaching away, contrarian invention, unexpected results)?
  - Winning congratulations from experts in the field?
  - A commercial success?
  - Shown to be unobvious in some other way?

Typically, unobviousness complaints occur when something one might like to use turns out to have been patented. Then one is in the "sour grapes" position of arguing that this invention is useful but obvious and so does not deserve a patent. But if some invention is both useful and obvious, surely it should exist in the prior art. So if there is no prior art, maybe the invention was not quite as obvious as we thought. Or maybe the invention was just "obvious" to a special group, which is not the same as being obvious to the average worker (and student) in that field.

Ideally, patenting is an essentially democratic institution which seeks to open expertise to all of society. Thus, patenting can be a problem for special groups who have no interest in informing others. But when specialized knowledge is not published, it can be patented by a later inventor, and the resulting monopoly then applied against those who simply could not be bothered to publish in the first place.

Patent Consequences

A patent typically protects something that works in a new way. A patent is about functioning, operating, and changing something, as opposed to copyright which is about things that have a fixed, publishable form. So copyright does not protect functioning even when that is described in words. Thus, copyrighting software may restrict others from copying a particular command sequence, but still allows others to use different sequences that accomplish the same thing. In contrast, a patent protects a particular functioning and will apply to any implementation of the same basic invention. Fortunately, patents are granted only for ideas not already in the open literature (but unpublished knowledge known only within a special group does not count, see patent complaints.)

One consequence of patenting is to achieve democratic publication to society at large, so that others can build on each invention. The alternative to patenting is to have inventions frequently kept as trade secrets or even as just a localized body of knowledge by individuals or elite groups. Without patents this happens automatically, as we can see from the software field, where patents were originally thought to not apply.

In return for publication as a patent document, the patent holder is compensated with a limited-term grant of monopoly on the invention. The patent holder is able to prevent anyone else from manufacturing, selling or using the patented invention without permission, and that includes even individual and non-profit manufacture, distribution and use. However, there does appear to be a distinction between repeated advantageous "use," and transient intellectual or scientific investigation.

Another effect of patenting is to protect investments made in research and development (R&D): Without a patent grant, inventions can be immediately copied and marketed at a lower price by a competitor who has no R&D overhead. In the extreme, a lack of patenting might kill commercial research, leaving only the government labs to do basic research and develop new types of consumer product. Most inventions can be "engineered around" to a greater or lesser extent, but doing that also requires R&D, which helps to keep the market competition fair. This is also a direct spur to creativity, and can produce various solutions to the original problem, some of which may be better than the one described by the first patent.

Patenting Cryptography

Historically, many ciphers have been covered by patent, including DES, IDEA and RSA. Unfortunately, obtaining a patent can be both arduous and expensive. Using a patent firm with experience in electronic hardware, computer software and cryptography could cost $50k or more. A more general firm might charge $20k, but at, say, $250 per hour, you might get only 80 hours of work, and much of that will be spent in teaching them what is important. Sometimes there may be cheaper options, such as having a patent expert in the family, or finding a retired patent guy still keeping his hand in.

The attorney starts out by working to understand the invention, which is quite different from understanding what can be built from the invention. The actual invention is the set of elements which distinguish from all prior art in the literature. Although modern on-line patent searching is both faster and easier than in the days of printed patents, each found patent or article still must be read and understood which inevitably takes considerable time. The inventor is the technical expert and will want to review and probably modify the application before it is submitted, which could imply substantial changes in attorney understanding and more art search and reading. All of that time will be paid for by the inventor.

It is possible to apply for a patent without an attorney or agent, but for a lay-person that is quite difficult. Someone involved has to know enough patent law to engage the bureaucratic process. Learning patent law from scratch can be surprisingly disturbing to math or science people: In law, words can have meanings and implications far removed from their ordinary dictionary definition. Understanding relationships between patent concepts is complicated by an implicit, inconsistent and sometimes irrational model based on case law. A self-education in basic law and patent law may take a full-time year, plus some experience and mistakes (perhaps the least-important invention should go first).

All U.S. patents also require an application fee and an issue fee, plus sizable periodic maintenance fees to keep the patent in force.

Rights in Conflict

After inventing something and intending to patent it, various things can go wrong:

1. *Somebody else may have already found the same invention.* If they have already established legal proof of invention and reduction to practice, they are on track to get the patent you want.
2. *Somebody may be working toward the same invention.* If you have established legal proof of invention and reduction to practice, you have the advantage, but time is of the essence.
3. *Somebody may have found out about the invention, and, combined with prior work (as in 1), be encouraged to move toward their own patent.* If you have established legal proof of invention and reduction to practice, you may have the advantage. Again, time is of the essence.
4. *The invention may be exposed, in print, in voice, by working example, or by sales offer.* Generally speaking, U.S. patent law allows a patent application to be submitted up to a year after public disclosure of the invention. Patent laws elsewhere generally do not allow that year of grace.

Since one generally does not know the situation with other parties, it may make sense to establish the earliest dates possible, and submit the formal application as soon as possible.

Establishing What One Knew, When

Under 35 USC 102(g), it can become important to have legally-provable dates for invention and reduction to practice. If nothing else, a formal patent application will create those dates, but that could be months later than possible. In some cases it may turn out to have been important to have established the earliest possible dates for who knew what when.

There are various reasonable ways to establish dates:

- The classic way to establish what one knew when is to write and publish an article in a magazine or journal. Ideally, that will produce a thorough description in a fixed form as of a provable date. But an article may harm patent rights unless published *after* the application has been submitted.
- An alternative way to establish what one knew when is to write and submit a formal patent application. That allows a thorough description in a fixed form and establishes unarguable dates. But the rigor involved in creating the application may cause those dates to be substantially later than they could be.
- Yet another way to establish what one knew when would be to write a formal disclosure document, and have someone who could understand it read it in confidence. Both inventor and witness should sign and date each page, and there should be more than one witness. The legally-provable part of this would be testimony in court by witnesses who read and signed the document on the indicated date. That means the document had better be thorough enough to provide all the evidence needed to support a future patent.

It is sometimes suggested that a discussion can provide legal evidence that an invention had been created. That approach has very serious problems, not only in the U.S., but in any formal legal system whatsoever:

- Verbal discussions rarely expose an invention in complete detail, but patents are granted or rejected on the basis of exact wording. That level of detail is rarely available from witnesses even if present in the original discussion.
- Memories of discussion are not be in "fixed form," but instead are free to change over time. For example, memories may tend to portray full content in one discussion, when details were actually filled in at a later date.
- A complete invention is not just an idea, but also a reduction-to-practice that demonstrates operation. A discussion cannot be reduction-to-practice, even though a formal application can.
- In the U.S., an inventor must move toward formal application, or give up rights to that patent. Mere discussions are not evidence of moving toward a patent.

The cross-examination of mere memories of a technical discussion would be brutal.

Patenting Software

In the 1980 decision "Diamond v. Chakrabarty", the U.S. Supreme Court held that patent law is intended to cover "anything under the sun that is made by man." Given such an enormous scope from the highest court, it is going to be very, very tough to successfully argue that software does not qualify for (ahem) "equal protection under the law."

The issue of software patents was addressed in courts and in massive discussions in the 1980's and 1990's. One of those discussions was the highly biased November, 1990 *Dr. Dobb's Journal* article by "The League for Programming Freedom." (The LPF paper "Against Software Patents" http://lpf.ai.mit.edu/Patents/against-software-patents.html appears to be the same as the DDJ article.) I published a response to the LPF article in 1991, which I called "The Politics of Software Patents" (locally, or @: http://www.ciphersbyritter.com/ARTS/POLIPAT4.HTM).

In 1990 it was relatively awkward and time-consuming for ordinary people to gain access to patents. But now is easy to browse to the PTO web site, and look up the text for any particular patent. Now we can look at the patents cited in the LPF article, and see if their criticisms can be supported.

In the case of "the technique of using exclusive-or to write a cursor onto a screen," for example, we actually find a patent on a way to create a graphic display that is larger than the screeen and which can be "zoomed." From a quick scan, it appears that only independent claim 10 and dependent claim 11 cover the graphic XOR. That is just 2 claims out of 16, but apparently enough for the LPF to judge the whole patent "absurd." But if this technique is both important and obvious, we have to ask: "Why is there no prior art?" Why was this "important" technique only annecdotal knowledge in a special group and not published for use by the rest of society (see patent complaints).

Patents have been covering software since about 1980. Contrary to the hysterical tone of the LPF article, over two decades have now passed without patents ruining the software industry. If anything, patents have been shown to be *too weak* to protect small businesses from big ones in a monopolized marketplace. Nor can we blame patents for making software expensive: Software is expensive because it is complex, and creating a lot of sofware takes a lot of work. And while licensing may be a problem for free software, just what part of free software is "free" if it uses someone else's stuff?

The general result of these discussions was a 1995 set of PTO "Final Patent Examination Guidelines" covering software presented for patent protection. Just as in the past, patents are *not* granted on pure algorithms. On the other hand, patents have *always* been granted on useful machines and processes, *all* of which can be described as "merely" implementing some algorithm.

There are no special "software patents." Software is not a new statutory class (see patent). However, patents can cover software just like patents have always covered a process or machine. (To be patented, inventions also must be useful, novel, and unobvious.)

Software does not and can not function by itself. When software supposedly operates (and, thus, passes out of the copyright arena and into the patent arena), it always does so in the context of a physical machine or hardware. Apparently it is the specific operation of the physical machine which is covered by patent.

Patent Infringement

Patent infringement occurs when someone makes, sells, or uses a patented invention without license from the patent holder.

```
UNITED STATES CODE
          TITLE 35 - PATENTS
               PART III - PATENTS AND PROTECTION OF PATENT RIGHTS
                    CHAPTER 28 - INFRINGEMENT OF PATENTS

§ 271. Infringement of patent
     (a) Except as otherwise provided in this title, whoever without
authority makes, uses or sells any patented invention, within the
United States during the term of the patent therefor, infringes the
patent.
     (b) Whoever actively induces infringement of a patent shall be
liable as an infringer.
     (c) Whoever sells a component of a patented machine, manufacture,
combination or composition, or a material or apparatus for use in
practicing a patented process, constituting a material part of the
invention, knowing the same to be especially made or especially
adapted for use in an infringement of such patent, and not a staple
article or commodity of commerce suitable for substantial
noninfringing use, shall be liable as a contributory infringer.
```

Note that the term "sells" in paragraphs (a) and (c) is generally taken to include free distribution. In general, making an invention for one's own use *does* constitute infringement.

Normally the offender will be contacted, and then there are various options:

- there may be a settlement and proper licensing, or
- the offender may be able to design around the patent, or
- the offender may stop infringing.

Should none of these things occur, the usual response is a patent infringement lawsuit in federal court. An option against a foreign manufacturer is to obtain an "exclusion order" from the International Trade Commission to prevent the import of infringing products.

In court, one option for the offender is to show that the patent was invalid for some reason, typically due to prior art which *anticipates* the patent.

Patent Reading

How we read a patent greatly depends on what we want to do or get out of it. Some reasons to read a patent include:

To Learn about the Invention

If we are just interested in learning, possibly to build something better, we can read a patent any way we want. We generally have no interest in dates or claims, just content. In general, we would start with the Abstract, then use the text and Figures and descriptions of operation to understand what the invention was beyond the prior art. Patent language is awkward largely because of PTO rules and court decisions, since a patent is only as good as can be enforced in court.

To Use as Prior Art for Another Patent

If we are looking at a patent because it may be prior art, there is no point in looking at the claims. Even a casual text description of an idea would qualify as prior art, if the date was early and the description contained *all* parts of some claim. Then that particular claim could be invalid. But we have to find prior art for every claim before the entire patent would be invalid.

To See if the Patent Covers Our Work

Here we *are* interested in the patent claims, because the claims define what the patent covers. To understand what is covered, it is necessary to take each claim apart, word by word, and understand what that claim really means. It is not possible to just read the Title or the Abstract of a patent and then leap to a conclusion about what the patent covers. Quite often, the claims in the original application have been modified several times before they are accepted and the patent issues. The resulting claims may not cover what the original Title and Abstract imply.

In general, a claim is supposed to be written so that any particular specific example can be judged to be either covered or not covered by that claim. But claims *must* be drafted so they do not include prior art or they will be invalid and rejected by the examiner. Accordingly, claims probably will not describe all the things necessary to make the invention "work."

For the patent holder, the best claims are the most general and have the fewest restrictions, because those can apply in more cases and create a stronger monopoly. The ideal claim will cover the full concept of the new invention, under all possible future implementations, which requires substantial generality. The need for generality tends to force claims to be somewhat abstract, yet they also must clearly apply to a wide range of particular instances of real infringing art.

To see if an article covers the same invention as a particular claim, we need to match every requirement in the claim with information from the article. If we can do that, and the article is dated more than a year before the patent application date, the claim is covering prior art, which is invalid. Unfortunately, defining the exact limits of a particular instance as reflected by an abstract claim can take some interpretation skill. In close situations, an absolute answer may not be possible outside a court of law, where interpretation occurs in the context of the body of case law that lay people will not know. Accordingly, in close situations, it is not always possible to know what the ultimate answer really is. But to get to court, we have to be able to form a substantial opinion on our own.

To Seek Prior Art Against the Patent

This is basically the same as repeatedly checking a claim against different articles until a match is found. But if the patent has been granted, both the Applicant and Examiner would have had to miss prior art, so there simply may not *be* any matching articles. And, of course, prior art must be checked against each claim for the entire patent to be invalidated.

Patent Valuation

A patent is the ability to limit competition, and that may carry some property value. There are various ways to think about the value of a patent:

Unique Property: Patents are the opposite of commodities, where essentially the same product is available from many competing sources. In contrast, each patent is supposed to be completely unique among all current and previous patents, and is held by only one supplier. A patent is a single item of ownership, like a particular fine diamond (or lump of coal), and once sold may not be available again. The value of such items is whatever is acceptable to the most competitive buyer and the seller.

Return on Investment: Most patents are a result of research investment. That investment typically has easily computable costs, including salaries or lost opportunity costs, research equipment, business supplies, services, rent, utilities, etc. The patent itself also has costs for preparation as well as application, issuance and maintenance fees. The value might be similar to a typical venture capital target for successful projects, a 5-times return on investment.

Competition: The ability to limit competition would seem to be most useful to those who can profit from the content of the patent itself. Typically that requires the ability to field and market a product competitively, or licensing someone who can. The value would then be related to profit or licensing fees. But in cryptography, new technology and cipher designs have trouble competing against a government-approved free cipher.

Secret Use: Especially in cryptography, an invention may be in secret production and use, thus allowing highly profitable equipment suppliers to avoid licensing fees. Individuals and small companies only rarely would have information about such use. But large companies in the field may have inside information about equipment made by others, and a patent covering that production could be useful to them. To know the value of the patent, it would be necessary to contact the large companies and ask them what they would be willing to pay. The value would then be the highest of those amounts.

Path

A trajectory. A sequence of states in FSM state-space.

Pedantic

The attitude which emphasizes meaningless detail.

Penknife

A DOS-era secret key stream cipher which I designed and implemented. Based on the nonlinear Dynamic

Substitution combiner technology which I invented, patented (see locally, or @: http://www.ciphersbyritter.com/PATS/DYNSBPAT.HTM), and own. More than just a cipher, a full cryptosystem with various cipher system features:

- 63-bit keyspace.
- Dual LFSR / LMG autokey RNG with jitterizer nonlinear isolation.
- Dynamic Substitution and Exclusive-OR combinings.
- The first combining is selected on a byte-by-byte basis from among 16 keyed Dynamic Substitution combiners.
- Ciphertext is emailable text in independent lines.
- Encoding for email is not a separate operation but is instead built into the cipher itself.
- Each ciphertext line has its own line key.
- Ciphertext can be concatenated.
- Ciphertext is automatically found amid plaintext headers, and can be deciphered in place, thus keeping email address and date.
- Strong key-management by alias file.
- Ciphers individual files, multiple files, or an entire disk.
- Supports user-programmed batch operations.

See the documentation:

- "Penknife Features" locally, or @: http://www.ciphersbyritter.com/PENFEA.HTM
- "The Penknife Cipher User's Manual" locally, or @: http://www.ciphersbyritter.com/PROD/PENDOC3.HTM
- "Penknife Quick Start" locally, or @: http://www.ciphersbyritter.com/PROD/PENQUICK.HTM
- "The Penknife Cipher Design" locally, or @: http://www.ciphersbyritter.com/PENDESN.HTM

Also see Cloak2.

Perfect Secrecy

The unbreakable strength delivered by a cipher in which all possible ciphertexts may be key-selected with equal probability given any possible plaintext. This means that no ciphertext can imply any particular plaintext any more than any other. This sort of cipher needs as much keying information as there is message information to be protected. A cipher with Perfect Secrecy has at least as many keys as messages, and may be seen as a (huge) Latin square.

> "'Perfect Secrecy' is defined by requiring of a system that after a cryptogram is intercepted by the enemy the *a posteriori* probabilities of this cryptogram representing various messages be identically the same as the *a priori* probabilities of the same messages before the interception. . . . perfect secrecy is possible but requires, if the number of messages is finite, the same number of possible keys. If the message is thought of as being constantly generated at a given 'rate' *R* . . . , key must be generated at the same or greater rate."
>
> "Perfect systems in which the number of cryptograms, the number of messages, and the number of keys are all equal are characterized by properties that (1) each *M [message]* is connected to each *E [cryptogram]* by exactly one line, (2) all keys are equally likely. Thus the matrix representation of the system is a 'Latin square.'"
>
> -- Shannon, C. E. 1949. Communication Theory of Secrecy Systems. *Bell System Technical Journal.* 28: 656-715.

There are some examples:

- (Theoretically) the one-time pad with a perfectly random pad generator.
- The Dynamic Transposition cipher approaches Perfect Secrecy in that every ciphertext is a bit-permuted balanced block. Thus, every possible plaintext block is just a particular permutation of any ciphertext

block. Since the permutation is created by a keyed RNG, we expect any particular permutation to "never" re-occur, and be easily protected from defined plaintext attack with the usual message key. We also expect that the RNG itself will be protected by the vast number of different sequences which could produce the exact same bit-pattern for any ciphertext result.

Also see: Ideal Secrecy, pure cipher, unicity distance, balance and Algebra of Secrecy Systems.

Period

1. The time between two events.
2. For periodic functions *f(x)*, the least positive number *p* that causes no change in function value for any *x*:

```
   f(x) = f(x + p)
```

Permutation

The mathematical term for a particular arrangement or re-arrangement of symbols, objects, or other elements. With *n* symbols, there are

```
   P(n) = n*(n-1)*(n-2)*...*2*1 = n!
```

or *n*- factorial possible permutations. The number of permutations of *n* things taken *k* at a time is:

```
   P(n,k) = n! / (n-k)!
```

For computation, see the permutations section of the "Base Conversion, Logs, Powers, Factorials, Permutations and Combinations in JavaScript" page (locally, or @: http://www.ciphersbyritter.com/JAVASCRP/PERMCOMB.HTM#Permutations). Also see combination and symmetric group.

A transposition is the simplest possible permutation, the exchange of two elements. A permutation which can be produced by an even number of transpositions is called *even*; otherwise, *odd*.

A block cipher can be seen as a substitution from plaintext block values to ciphertext. Both plaintext and ciphertext have the same alphabet or set of possible block values, and when the ciphertext values have the same ordering as the plaintext, ciphering is obviously ineffective. So *effective* ciphering depends upon *re-arranging* the ciphertext values from the plaintext ordering, which is a *permutation* of the plaintext values. A block cipher is keyed by constructing a *particular* permutation of ciphertext values.

Within an explicit substitution table or block of values, an arbitrary permutation (one of the set of all possible permutations) can be produced by shuffling the elements under the control of a random number generator. If, as usual, the random number generator has been initialized from a key, a particular permutation can be produced for each particular key; thus, each key selects a particular permutation.

Also, the second part of substitution-permutation block ciphers: First, substitution operations diffuse information across the width of each substitutions. Next, "permutation" operations act to re-arrange the bits of the substituted result (more clearly described as a set of transpositions); this ends a single round. In subsequent rounds, further substitutions and transpositions occur until the block is thoroughly mixed and overall diffusion hopefully achieved.

PGP

A popular public key cipher system using both RSA and IDEA ciphers. RSA is used to tranfer a random key; IDEA is used to actually protect the message.

One problem with PGP is a relatively unworkable facility for authenticating public keys. While the users can compare a cryptographic hash of a key, this requires communication through a different channel, which is more than most users are willing to do. The result is a system which generally supports man-in-the-middle attacks, and these do **not** require "breaking" either of the ciphers.

Phase

With respect to a repetitive or cyclic activity, a position or distance from the start of the cycle. Often measured as phase angle in degrees from some sine wave oscillator base.

Phase Locked Loop

In electronics, circuits that use a phase detector and feedback to "lock" a VCO to a reference frequency source. Typically, both the reference and VCO frequencies are divided by presettable digital counters prior to phase detection, thus supporting a potentially wide range of VCO frequencies, each locked to the fixed reference.

Phase Noise

A noise-like variation in the phase of a signal, typically an oscillator. In general, phase noise will be small, bipolar, and normally distributed around some mean value, and so will not accumulate over time or change average frequency.

The most obvious source of phase noise in crystal oscillators with digital output is analog noise in the analog-to-digital conversion (typically a single transistor).

Physically Random

A random value or sequence derived from a physical source, typically thermal-electrical noise. Also called really random and truly random.

Piezoelectric

The physical property of some materials to convert electrical voltage into physical force, and vice versa. If voltage is applied across a thin wafer of quartz, that wafer may bend or twist. (In practice, the amount of bending is something like the diameter of one quartz atom.) Conversely, if force is applied to the wafer, causing it to bend or twist, a voltage can be produced. This effect can be used to cause or detect physical motion (as in ultrasonic detectors), ignite the gas in lighters, or simply to induce motion in the wafer itself (as in crystal oscillators).

Pink Noise

(1/f noise.) A random-like signal in which the magnitude of the spectrum at each frequency is proportional to the inverse of the frequency, or 1/f. At twice the frequency, we have half the energy, which is -3 dB. This is a frequency-response slope of -3 dB / octave, or -10 dB / decade. As opposed to white noise, which has the same energy at all frequencies, pink noise has more low-frequency or "red" components, and so is called "pink."

A common frequency response has half the output voltage at twice the frequency. But this is actually one-quarter the power and so is a -6 dB / octave drop. For pink noise, the desired voltage drop per octave is 0.707.

Pipeline

The concept of sending data through a *pipe* or sequence of computations, all occurring simultaneously. Each stage of computation need only add to the result of the previous stage, instead of performing the entire computation from scratch. Thus, a pipeline supports a high *data rate*, because the full computation is broken down into a series of small, fast stages, each of which is performed simultaneously in hardware.

Pipelining is especially practical in the context of modern semiconductor digital logic where the cost of added hardware is often minimal. Of course, any particular data must flow through *all* the stages of the computation, incurring latency at each stage. A pipeline thus adds hardware and trades increased latency to maintain a high data rate.

PKCS

Public-Key Cryptography Standard. A series of "standards" promoted by *RSA Security.*

PKI

Public-key infrastructure.

Plagiarism

1. Presenting someone else's words as your own. Since copyright law protects words, this also could be copyright infringement.
2. Presenting work based on the ideas of someone else without referencing that source.

Plaintext

Plaintext is the original message, before encryption. Plaintext usually is readable or understandable, but need not be. As opposed to ciphertext. Also see message.

It is sometimes convenient to think of plaintext as being represented in normal alphabetic characters, but plaintext may instead be *any* symbols or values (such as binary computer data) which need to be protected.

PLL

Phase Locked Loop.

PN Sequence

Pseudonoise sequence. See PRBS, LFSR and RNG.

Poisson Distribution

In statistics, a simplified form of the binomial distribution, justified when we have:

1. a large number of trials *n,*
2. a small probability of success *p,* and
3. an expectation *np* much smaller than SQRT(*n*).

The probability of finding exactly *k* successes when we have expectation *u* is:

```
            k   -u
   P(k,u) = u  e     /  k!
```

where *e* is the base of natural logarithms:

```
   e = 2.71828...
```

and *u* is:

```
   u = n p
```

again for *n* independent trials, when each trial has success probability *p.* In the Poisson distribution, *u* is also both the mean and the variance

The ideal distribution is produced by evaluating the probability function for all possible *k,* from 0 to *n.*

If we have an experiment which we think *should* produce a Poisson distribution, and then repeatedly and systematically find very improbable test values, we may choose to reject the null hypothesis that the experimental distribution is in fact Poisson.

Also see the Poisson section of the Ciphers By Ritter / JavaScript computation pages.

Polyalphabetic Combiner

A combining mechanism in which one input selects a substitution alphabet (or table), and another input selects a value from within the selected alphabet, said value becoming the combining result. Also called a Table Selection Combiner.

Polyalphabetic Substitution

A type of substitution in which multiple distinct simple substitution arrangements are each selected at different times. This has the effect of complicating and hiding the characteristics exposed by any one fixed permutation. Also see: a cipher taxonomy.

Polygram Substitution

A type of substitution in which one or more symbols are substituted for one or more symbols. The most general possible substitution.

Polygraphic

Greek for "multiple letters." A cipher which translates multiple plaintext symbols at a time into ciphertext. As opposed to monographic; also see homophonic and polyphonic.

Polynomial

An expression with multiple terms, each a different power of the same variable. The standard form is:

$$c_n x^n + . . . + c_1 x + c_0$$

The variable is x. The c's are coefficients of x; c_0 is the constant term. The degree *n* is the value of the exponent of the highest power term. A mod 2 polynomial of degree *n* has *n*+1 bits representing the coefficients for each power: *n*, *n*-1, ..., 1, 0. As opposed to monomial; also see: trinomial.

Perhaps the most insightful part of this is that the addition of coefficients for a particular power does not "carry" into other coefficients or columns. (See operation examples in mod 2 polynomial and GF(2^n).)

Polyphonic

Greek for "multiple sounds." The concept of having a letter sequence which is pronounced in distinctly different ways, depending on context. In cryptography, a cipher which uses a single ciphertext symbol to represent multiple different plaintext symbols. Also see homophonic, polygraphic and monographic.

Population

In statistics, the size, or the number of distinct elements in the possibly hidden universe of elements which we can only know by sampling. Also see alphabet, population estimation and cardinal.

Population Estimation

In statistics, techniques used to predict the population based only on information from random samples on that population.

Estimating population is fundamentally tricky. We want to know how many items exist without traversing them. But if items are independent, how can the information from the ones we touch indicate anything about ones we do not? This apparent contradiction can be addressed with the birthday paradox. We first assume that each element has some sort of unique identity that we can find and use, or that each element can be marked with the number of capturings. We also assume that sampling is random in the sense of making each element in the population equally likely in each sample.

We take a sample, save each identity or mark each element, then "throw them back" (this is sampling with replacement). If these are physical items, we may need to do some "mixing" or allow a time delay to make each element equally available for the next sample. As we continue to take samples, eventually we re-capture elements we have had before, which start as "2-repetitions" and count up. Repetitions start to indicate population size, except that when sampling is random, anything can happen. But when we accumulate more than just a few repetitions, we expect to find a fairly stable statistical relationship between repetitions and population, as described under augmented repetitions.

The birthday strategy depends upon the randomness of the sampling making each item equally available. If for some reason some items are inherently more likely to be sampled than others, that will distort our view of the population. One example would be to unknowingly work on a sub-population which has some amount of diffusion to and from other sub-populations. Meaningful results will require characterizing the situation beyond simply cranking out numbers.

Power

1. In algebra, a value written as a superscript, indicating repeated multiplication.
2. In statistics, the probability of rejecting a false null hypothesis, and thus accepting a true alternative hypothesis. Also see Type I error and Type II error.
3. In electronic circuits, the necessary source of energy. Because all electronic results depend upon having appropriate power, distributing power and isolating the effects of powered components is one of the most important tasks of electronic design. See: bypass and decoupling.
4. In electronics, a measure of the amount of energy used. In DC electronics, simply voltage times current. In AC electronics, the instantaneous product of voltage times current, integrated over a repetitive cycle. In either case the result is in watts, denoted W.

Practice

The application of knowledge in reality. As opposed to theory.

PRBS

Pseudorandom binary sequence. See LFSR and RNG.

Precision

The extent to which a measured value is distinguished from an adjacent value. Also see: accuracy.

Predictable

Values which can be known or computed before they occur. "Making verifiable predictions" in the form of numerical scientific models is at the heart of scientific investigation. Because of the numerical and experimental nature of science, issues of accuracy and precision are universal and readily handled. In contrast, unpredictability is the essence of cryptography. Also see hypothesis and random.

In cryptography and cryptanalysis, "predictability" does not imply the ability to predict everything all the time, nor even the ability to exactly predict values. Instead, "predictability" means the ability to repeatably predict bits with anything other than the 50 percent success we expect at random. For example, if we try to predict 8-bit byte values from a random number generator, we expect to get one match about every 256 guesses, on average. If we can reliably do better than that, or even *worse* than that, some amount of predictability must be involved.

Predictability inherently involves some *context* for knowing and reasoning: If absolutely *everything* is known there is no need for "prediction," and if *nothing* is known, prediction is not possible. Prediction thus involves taking a limited amount of information and using that to correctly extrapolate other information. In cryptography, prediction often involves knowing a generator design and having some of a sequence, and using that information to predict other parts of the sequence.

Premise

In the study of logic, statements which are assumed true for the purposes of an argument. Also see: assumption, condition and conclusion.

Primitive
A value within a finite field which, when taken to increasing powers, produces all field values except zero. A primitive binary polynomial will be irreducible, but not all irreducibles are necessarily primitive.

Primitive Polynomial
An irreducible polynomial, primitive within a given field, which generates a maximal length sequence in linear feedback shift register (LFSR) applications.

All primitive polynomials are irreducible, but irreducibles are not necessarily primitive, unless the degree of the polynomial is a Mersenne prime. One way to find a primitive polynomial is to select an appropriate Mersenne prime degree and find an irreducible using Algorithm A of Ben Or:

```
1.  Generate a monic random polynomial gx of degree n over GF(q);
2.  ux := x;
3.  for k := 1 to (n DIV 2) do
4.     ux := ux^q mod gx;
5.     if GCD(gx, ux-x) <> 1 then go to 1 fi;
6.     od
```

Ben-Or, M. 1981. Probabilistic algorithms in finite fields. *Proceedings of the 22nd IEEE Foundations of Computer Science Symposium.* 394-398.

The result is a certified irreducible. GF(q) represents the Galois Field to the prime base *q*; for mod 2 polynomials, *q* is 2. These computations require mod 2 polynomial arithmetic operations for polynomials of large degree; "ux^q" is a polynomial squared, and "mod *gx*" is a polynomial division. A "monic" polynomial has a leading coefficient of 1; this is a natural consequence of mod 2 polynomials of any degree. The first step assigns the polynomial "x" to the variable *ux*; the polynomial "x" is x^1, otherwise known as "10".

To get primitives of non-Mersenne prime degree n, we certify irreducibles P of degree n. To do this, we must factor the value $2^n - 1$ (which can be a difficult problem, in general). Then, for each factor d of $2^n - 1$ we create the polynomial T(d) which is $x^d + 1$; this is a polynomial with just two bits set: bit d and bit 0. If P evenly divides T(d) for some divisor d, P cannot be primitive. So if P does not divide any T(d) for all distinct divisors d of $2^n - 1$, P is primitive.

Prime
In general, a positive integer which is evenly divisible only by itself and 1.

Small primes can be found though the ever-popular Sieve of Eratosthenes, which can also be used to develop a list of small primes used for testing individual values. A potential prime need only be divided by each prime equal to or less than the square-root of the value of interest; if any remainder is zero, the number is not prime.

Large primes can be found by probabilistic tests of values picked at random. Also see relatively prime.

Prior Art
In patent law, the term of art representing knowledge published or otherwise *available to the public* as of some date. Traditionally, this "knowledge" is contained in ink-on-paper articles or patents, both of which have a fixed content as of provable publication dates. In contrast, private "in house" journals available only within a company generally would not be prior art, nor would information which deliberately has been kept secret. Normally, we expect prior art information to be available in a public library.

Note that "prior art" does **not** just mean *something similar at an earlier time*. An invention which is kept as a trade secret is never published and does not establish "prior art." So if the exact same invention is re-developed by someone else, the earlier trade secret development would *not* (normally) be "prior art" and would *not* (normally) prevent a patent on the new development. (Naturally, special situations may rule, and in any serious context the services of a patent attorney will be required.)

More precisely, in the U.S., prior art anticipates the current application if we can prove that the same invention was:

- described in a printed publication available to the public anywhere, or patented or used by others in the U.S., before the current invention date (from 35 USC 102(a)); or
- described in a printed publication available to the public or patented anywhere, or in public use or display or on sale in the U.S., more than one year before the current application date (from 35 USC 102(b)).

All of these bars are date-based; therefore, when an invention is identified, it is important to move swiftly toward filing some sort of application that can establish what was known and when. During any delay, publications can occur which may later be seen as prior art.

To be considered prior art, a publication must contain sufficient detail to allow someone of average skill to practice the invention. Moreover, the publication must be in some fixed form which is not continually updated, so that it captures exactly and completely what was known as of a particular date. Presumably, the invention date (which is prior to publication) could be bounded by a dated and signed disclosure to several people, each of whom could understand it and testify about what the inventor knew and when he knew it. If patent application is later to be made in countries other than the U.S., it is important that the U.S. application be filed before the invention is publicly disclosed. Again, in any real situation, seek competent professional advice.

One oft-mentioned source of prior art is "published" software But "publication" of software as object code only is *not* the same as "publication" of the details of an invention in a magazine article. Normally we expect prior art to inform the public -- especially the ordinary practitioner in the field of the invention -- about how to practice the invention. But software object code does not teach, and so is at least arguably *not* prior art. (Also see my article: "The Politics of 'Software Patents,'" locally or @: http://www.ciphersbyritter.com/ARTS/POLIPAT4.HTM.)

Released program object code might be seen as *public use* of a contained invention, but it still does not teach, and the associated source code generally is kept secret, which seems like trade secrecy. The problem with this is that neither the later inventor nor the PTO could be expected to know anything about a deliberately hidden invention. And, absent printed publication, there could be some temptation to misrepresent exactly what was inside a program as of a given date. Different versions of a program are often released without fanfare. Unless we know for sure that the invention was present on a particular date, we may not really know that the invention was publicly used. And assuring that may be tougher than it sounds if any other approach exists which could produce a similar effect, such that the distinction would not be noticed by users.

Software source code could be prior art if actually released to the public. But source code which is *not* released would seem to be more like trade secrecy than patent publication, and an inventor cannot have *both* trade secrecy *and* patent protection. When inventors choose trade secrecy, they give up their rights to a future patent on that invention. But they also have not created any searchable prior art. That means they should be in no way surprised if the very same invention is patented by a later inventor and then applied against them. Surely, neither the later inventor nor the PTO can be faulted for not finding prior art which the first inventors deliberately chose not to expose.

Prior art is a particular issue in software because, if object code distribution is *not* prior art, inventions might be in production, sold, and used for years before a valid new patent issues and is unexpectedly applied against an existing product. But the problem is easily avoided simply by the first inventor publishing software inventions and not keeping them as trade secrets. The real problem is that businesses often want things *both ways*: they *do* want the advantage of a trade secret, and *do not* want to educate the public, but also *do not* want a subsequent patent to be applied against them. Unfortunately, that may be too many things to want. (A patent attorney should be consulted in any significant case.)

In a U.S. application for patent, we are interested in the state of the open or public art as it existed as of the invention date, and also one year prior to the filing date. It is that art -- and not something secret or something later -- against which the new application must be judged. Many things which seem "obvious" in retrospect were really quite innovative at the time they were done.

Privacy
Seclusion. Not being intruded upon. Information belonging to the individual, and not society at large.

PRNG
Pseudo Random Number Generator; Pseudo RNG. Generally redundant, since "RNG" *implies* the classic deterministic finite state machine generators developed for statistical use. As opposed to TRNG.

Probability
1. The study of chance.
2. In statistics, the numerical expectation of some event, a value ranging from zero (impossibility) to one (absolute certainty).

Process
In statistics, a sequence of values; a source or generator of such a sequence; a function. One example is a stochastic process.

Product Cipher
1. A conventional block cipher composed of the repeated application of a sequence of distinct, weak operations orgainized in rounds. The actual operations may include simple substitution, fixed permutation and additive combining with a key. Typically, each round is given different keying, although the keys may be related and not independent. A simple but common and accepted form of multiple encryption. In general, an iterated block cipher.
2. The general concept of multiple encryption as described in Shannon's Algebra of Secrecy Systems. Shannon called the sequential application of different ciphers the "product," and the selection from among a variety of different ciphers the "sum."

Proof
1. A guarantee of certain truth.
2. The argument or deductive *process* involved in establishing certain truth from evidence or assumptions. Also see: scientific method.
3. The decomposition of a conjecture into sub-conjectures or lemmas. (While this does not by itself establish truth, it allows each of the sub-conjectures to be attacked by counterexample, thus increasing the ability to find error.) Related to analysis and Method of Proof and Refutations.
4. Properly-evaluated statistical evidence that something is more likely true than false.
5. Repeated experience that something can be counted on to perform some expected purpose (e.g., "proven in practice"). This is more of a heuristic or rule of thumb than a certainty, since even a reliable car *will* fail to start eventually, just as every bridge eventually falls or is replaced. But even the weak sort of rule established by experience generally is not available in cryptography, where the security outcome generally cannot be known; see: scientific method and trust.

Proofs can be either formal (independent of context, and thus mechanically verifiable) or informal (dependent on definitions, thus subjecting correctness to individual skill). Mathematics relates to the real world by modeling reality, but it is impossible to capture all aspects of reality in a model. Applied mathematics is thus forced to creatively abstract reality into a simplified form intended to contain all *significant* relationships, something which also cannot be checked mechanically. Yet even valid proof results apply to the real world only to the extent that the model is appropriate, correct and sufficient for the issue in the real-world domain (see mathematical cryptography).

Security Proofs in Practice

While the mere existence of a math security proof may inspire hope, it should provide no confidence. Only the fulfilled conclusion of a proof can really give confidence, and that is only possible when all assumptions *can* be guaranteed and *are* in fact guaranteed. Moreover, security proofs provide confidence only for someone who can and does ensure that each and every assumption the proof requires is actually met in practice. That is hardly ever possible for users.

Although a proof may seem to offer users some confidence even with partially fulfilled assumptions, that is logically wrong. Unless *all* assumptions are *completely* fulfilled *and* absolutely guaranteed, a proof guarantees *nothing at all!* Unfortunately, the typical proof has an all-knowing "omniscient" form that implies a context quite unlike what users have in reality. Real users are limited in what they can guarantee, and so generally *can* not ensure the assumptions made in such proofs, which means the proof has given them nothing at all.

In mathematics, every assumption is equally important, but in practice, some assumptions are more equal than others. Whereas a user just might be able to guarantee some assumptions, others are completely outside user control. But assumptions which cannot be controlled or verified also cannot be trusted, which means the "proven" conclusion cannot be trusted either. Such a "proof" provides no basis for confidence in practice, even though cryptography often claims otherwise.

For example, knowing that a one time pad (OTP) is proven secure in a theoretical sense might seem to give a user confidence in buying or using an OTP cipher. But user confidence is appropriate only when the user can be absolutely sure that every proof assumption really is present in practice. In particular, the OTP proof makes the assumption that the pad or sequence is really random, in the sense of being unpredictable. Now, that is a typically omniscient assumption, because it demands something outside the context of what a mere user *can* know. (Notice that the issue is *not* whether the sequence really *is* unpredictable, but instead whether a user can have an absolute guarantee that it is, because that is what it takes to guarantee the proof result.) Since there is no test that will guarantee the unpredictability of a sequence, the user cannot even *check* the assumption. Even worse, the user should know that *nobody* can check that assumption, not even the designer or supplier.

Math people may reasonably hold that an OTP without every required assumption is not really a mathematical OTP after all, implying that such discussions have nothing to do with *them*. But in that case, we can have ciphers which are *intended* to be math OTP's, and which *seem* to be math OTP's, and which users *cannot distinguish* from math OTP's, and so are *trusted* to be math OTP's, but which actually *are not covered* by the math proof. In the case of the OTP, and in most other cases as well, users (and designers) cannot trust a mathematical "proof" to predict and guarantee their security. A general inability to distinguish between conforming and non-conforming implementations destroys the value of most security proofs in practical cryptography.

With respect to OTP's, math has not only *not* helped, but has actually *damaged* security, and that damage continues. When a user cannot be assured of knowing the difference between covered and uncovered implementations, a proof provides *not* confidence but instead *cover* for delusion and deception. Since being falsely sure of security is vastly worse than remaining uncertain, the false claims of proven OTP practical security actually *reduce* security, instead of guaranteeing security. And every learned paper and text and lecture and comment that can be interpreted as saying that OTP's are proven secure in practice is part of the problem, which is not a proud record for cryptography as a science of reality.

[In practice, nobody and nothing can provide an absolute guarantee of sequence unpredictability, which means there is and can be no guaranteed math OTP in practice. So it should come as no surprise that there are actual instances of *apparently* "mathematically proven secure" OTP ciphers exposing life-critical information.]

Academic Proof

In academic cryptography there are many "proofs" in the sense of the second definition above: the proof *process*, not the result. These proofs (2) make assumptions, and their results are true only *if* all their assumptions are true, but usually that is not proven, nor even provable. In a real sense, these proofs are best described as lemmas for the desired theorems of reality which do not (and probably cannot) exist. Note that a mathematical proof (2) is considered valid, and can be considered important, *even if the assumptions in it cannot be achieved in practice*.

A cryptographic proof (2) can be more difficult to understand than outsiders might think. Only rarely is there a clear statement of every assumption which the author of the proof thinks has been made. Sometimes, some of the assumptions needed in proofs are hidden (such as those implicit in the use of particular operations with which an outsider may be unfamiliar), but even hidden asumptions must be fullfilled for the proof to "work." The inability to decide whether all assumptions have been exposed and fullfilled makes proofs difficult to depend upon in practice.

Moreover, a cryptographic proof (2) may use terms of art which can be deceptive in not having the same meaning as the exact same phrase in more general cryptography. This problem of meaning carries through to the conclusion, where a language description of the results may not correspond to the long-accepted use of the exact same terms in cryptography or security.

To use a proof in actual systems, it is important to understand what it really says in the broader context of the outside world:

- Are *all* of the assuptions known? (Can we prove that?)
- Can *all* the assumptions be guaranteed? (Can we prove that in an implemented system?)
- Are the guaranteed results really what we need?

There is no "proof" that using any particular cipher (such as DES or AES) improves security over using *no cipher at all*. This is because a user cannot know if the cipher is being broken by an opponent in secret. And if the cipher *is* being broken, using that cipher is probably *worse* than having no cipher at all, because users will be mislead into not taking even ordinary precautions. (The possibility of repeatedly using a broken cipher can be addressed by multiple encryption and by having and using many different ciphers.)

Unfortunately, often someone will see an academic result and say: "A cipher using this technique is proven secure." Usually, such a statement is false. When we say "secure" we are talking about a final result, not a process which may or may not end in that result. No cipher is proven secure in practice, and only a very few (generally inefficient) ciphers are proven to depend upon some fairly well examined math problem which has not been solved.

Sometimes the situation is even worse: Sometimes, a ciphering technique will be said to be secure, *provided* the underlying cipher is secure. But since we cannot know whether the underlying cipher really *is* secure, the argument seems to be a form of circular reasoning: We cannot tell whether the enabling assumption is true until we know the outcome, and, in cryptography, we generally cannot expect to know the outcome. And if we somehow *do* know the outcome, we do not need the proof! Such a proof adds nothing at all to practical security.

Mathematical proofs may change and develop over time; see: Method of Proof and Refutations. Also see the Triple-DES is Proven to be Very Secure? discussion (locally, or @: http://www.ciphersbyritter.com/NEWS5/PROVSEC.HTM) and my *IEEE Computer* article "Cryptography: Is Staying with the Herd Really Best?" (Copyright 1999, IEEE) (locally, or @:

http://www.ciphersbyritter.com/ARTS/R8INTW1.PDF). Also see the short article by: Devlin, K. 2003. "2003: Mathematicians Face Uncertainty." *Discover.* 25(1): 36. January, 2004.

Propaganda

Arguments made not for the purpose of reasoning to a correct result, but to further a cause. Most apparent when the reasoning is not correct, as indicated by the use of logic fallacies.

Proposition

A statement. Also see conjecture.

Pseudorandom

Something which *appears* to be random, but in fact is not. Typically, a sequence of values produced by an RNG, or any other completely deterministic computational mechanism or FSM. As opposed to really random.

The usual random number generator is *pseudo*random. Given the initial state or seed, the entire subsequent sequence is completely pre-determined, but nevertheless exhibits many of the expected characteristics of a random sequence. Pseudorandomness supports generating the exact same sequence repeatedly at different times or locations. Pseudorandomness is generally produced by a mathematical process, which may provide good assurances as to the resulting statistics, assurances which a really random generator generally cannot provide. In these ways, a pseudorandom generator is generally *better* than a really random generator.

PTO

The U.S. Patent and Trademark Office, a department of the United States government. See the United States Patent and Trademark Office site (http://www.uspto.gov/), and intellectual property.

Public Key Cipher

Also called an asymmetric cipher or a *two-key* cipher. As opposed to a secret key cipher which uses a single key for both enciphering and deciphering, a public key cipher uses one key to encipher plaintext into ciphertext, and a *different* key to decipher that ciphertext. Since the enciphering key cannot decipher the ciphertext, the enciphering key can be exposed without also exposing the message.

The keys used in public key ciphers are just extremely large numeric values, sometimes expressed in hexadecimal form. Typically, pairs of related keys are generated, and either key can be used for enciphering or deciphering. The exposed key is called the "public" key, and the retained hidden key is called the "private" key. The public key is distributed widely, so anyone can use it to encipher a message which presumably can only be deciphered by someone who has the hidden private key. There are various consequences:

- The enciphering end normally does not possess the key which will decipher a message which was just public-key enciphered.
- The whole public key concept depends upon the assumption that the private key cannot be developed from knowledge of the public key. The cipher also must resist both known-plaintext *and* defined-plaintext attack (since anyone can generate any amount of plaintext and encipher it).
- Unfortunately, public key ciphering tends to be very, very slow. Thus, most so-called "public key ciphers" are hybrid cipher systems having both public key and secret key components. In a hybrid system, the public key part is used simply to deliver the message key or session key for a secret key cipher which actually protects data.
- The effective keyspace of a public key cipher like RSA is vastly smaller than it appears. A 1000-bit public key component may have a brute force key strength similar to an 80-bit secret key cipher. This vast reduction in apparent strength occurs because public key ciphers generally demand that their keys be in a particular and extremely rare form. Most of the possible values of public-key size are not

acceptable as keys.

The Man in the Middle

Although public key ciphering was at first proclaimed as a solution to the key distribution problem, it soon became apparent that someone could *pretend* to be someone else, and send out a "spoofed" public key. When people use that key, the spoofer could receive the message, decipher and read it, then re-encipher the message under the correct key and send it to the correct destination. This is known as a man-in-the-middle (MITM) attack.

A MITM attack is unusual in that it can penetrate cipher security *without* "breaking" *either* the public key cipher *or* the internal secret key cipher, and takes almost no computational effort. This is *extremely serious* because it means that the use of even "unbreakable" ciphers is **not** sufficient to guarantee privacy. All the effort spent on proving the strength of either cipher is simply wasted when a MITM attack is possible, and MITM attacks are only possible with public key ciphers (one exception to this rule may be the first block of CBC mode).

To prevent spoofing, public keys must be authenticated (or *validated* or certified) as representing who they claim to represent (also see public key infrastructure). That can be almost as difficult as the conventional key distribution problem and generally requires complex protocols. And a failure in a key certification protocol can even expose a system which uses "unbreakable" ciphers. In contrast, the simple use of an "unbreakable" secret key cipher (with hand-delivered keys) **is** sufficient to guarantee security. This is a real, vital difference between ciphering models.

Public Key Infrastructure

PKI. The facilities, standards and laws needed for public key certification.

Network routing is insecure. Consequently, public-key technology is vulnerable to man-in-the-middle (MITM) attack: If someone can intercept and substitute messages, they can stand "in the middle" of communications between others, reading their messages in the MITM public keys, then sending the messages along in the appropriate public key for the other party. This does not require much computation, nor any breaking of a cipher. Thus, it is a vastly more serious threat than academic attacks which require massive computation and vast amounts known-plaintext.

It seems impossible to absolutely protect against MITM attacks with public key technology and only two parties. In general, the way to prevent MITM operation is to certify public keys. And, in general, this means having a third party -- a certification authority (CA) -- who can attest to the correctness of the public key for each party. Of course, if the CA lies, an MITM can operate with impunity.

The PKI is the CA facility, the technical interface to that storage, the payment structure to support it, and the legal structure to trust it.

Pure Cipher

A cipher which is almost a group.

"A cipher T is *pure* if for every T_i, T_j, T_k there is a T_s such that

$$T_i \; T_j^{-1} \; T_k = T_s$$

and every key is equally likely. Otherwise the cipher is mixed."

The operation $T_i^{-1} T_j$ means, of course, enciphering the message with key *j* and then deciphering with key *i* which brings us back to the message space."

"The importance of the concept of a pure cipher (and the reason for the name) lies in the fact that in a pure cipher all keys are essentially the same. Whatever key is used for a particular message, the *a posteriori* probabilities of all messages are equal."

"As an example . . . , simple substitution with all keys equally likely is a pure cipher."

-- Shannon, C. E. 1949. Communication Theory of Secrecy Systems. *Bell System Technical Journal.* 28:656-715.

See also: EDE, Triple DES, multiple encryption, Perfect Secrecy, Ideal Secrecy, unicity distance and Algebra of Secrecy Systems.

Qbit
Quantum bit. In a quantum computer, the storage that can be in many different states simultaneously.

Quality
The extent to which some manufactured item meets specifications. Also see quality management.

Quality Management
The idea that defects occur in produced items because the manufacturing process has allowed those problems to occur. One alternative is to use testing to find defects, then analyze the defects to identify problems in the manufacturing process, then fix the manufacturing process so those particular defects do not come up again. It is almost always cheaper to design quality into the manufacturing process then to simply discard or try to re-work parts which do not meet specifications.

Quality control requires the ability to measure the parameter being controlled to see if it meets specifications. That is a problem for cryptography, because what we want is a cipher that cannot be broken. The problem is that cipher strength exists only in the context of each opponent, a context we do not and can not know. There is no general test for cryptographic strength, and no such test can exist. Cipher strength is thus "out of control" in a quality manufacturing sense. We not only do not know how much we have, we also do not know which designs are "better," or even whether improvements actually help.

There are at least two different ways to produce precision equipment:

1. Build in some adjustment room so that manufactured result can be precisely calibrated, and
2. Make the manufacturing itself sufficiently precise that calibration is not required.

The first of these seems typical of some industries before the quality revolution of the 1980's and 1990's. With better manufacturing capabilities, the need for adjustment is reduced, leading to better overall product and cheaper manufacturing operations.

This attention to quality is especially important when the produced items are components in large systems: When systems are built from many components, even a small defect rate in each of the components can result in almost no large systems working. As a result, a builder of large systems using moderate quality components may have to test all incoming product. But it is both better and cheaper to have the component manufacturer produce extremely high-quality components than for each buyer to test all incoming product.

There are various forms, such as: "Total Quality Management (TCM)," "Zero Defect Management (ZD)," "Statistical Quality Control" and so on. Such schemes normally require extensive testing at all manufacturing levels so that faults are not propagated into the final item. (Also see: system design.)

Quantum Computer

A computer based on the physics of quantum mechanics which apparently allow a computation to exist in many different states simultaneously. It is very difficult to build such a machine, and all known examples are so small as to not provide much insight about a larger design. However, a practical quantum computer might:

- Weaken or destroy public key cryptography by allowing huge composite integers to be factored.
- Require twice as many key bits (e.g., at least 160) in secret key cryptography.

Personally, I question whether physics really has any sort of mystical action which allows a computation to be in *all possible* combinations of states simultaneously. I would be much happier to learn that hidden, apparently random changes in qbit values cause each result to be visited on a rapid statistical basis which we then take as simultaneous. That would be a sort of quantum-mechanical randomized brute force traversal. One consequence would be an exponential increase in fundamental computation time with numbers of qbits (although it might take some number of qbits before this becomes apparent). Another consequence would be an uncertainty that every possible state had in fact been covered.

Quantum Cryptography

The idea of solving the problem of key distribution by methods of quantum-mechanical physics.

Different strategies exist, but typically, light photons are given particular quantum states and transported on fiber optic cable. It is asserted that quantum physics makes it possible to detect any attempt to interfere with or monitor those photons, which thus implies secure key-transport.

Currently limited to: 1) cables directly connected between each communicating pair of users; 2) a modest distance (e.g., 122 km); and 3) relatively low data rates (e.g., 25kb/sec). Could be used to send actual data instead of keys, but low data rates argue against that, outside exceptional circumstances. Constant monitoring for external intrusion is a necessary part of security, and may be more difficult than normally assumed. Probably it will be necessary to introduce periodic controlled faults simply to assure that the monitoring system is functioning properly.

Quartz

A generally translucent, crystal form of silicon dioxide (SiO_2 or silica). The amorphous or non-crystalline form is common as beach sand and window glass and is arguably the most abundant mineral on Earth.

R

In math notation, the real numbers.

(R,+)

In math notation, the group of real numbers under addition.

R[x]

In math notation, the ring of polynomials in x over ring R.

R/I

In math notation, the residue class of the ring R modulo the ideal I.

Random

The absence of known pattern. Potentially unpredictable. Ideally, a process which selects unpredictably, each time independent of all previous times, from among multiple possible results; or a result from such a process. Best randomness requires an arbitrary stateless selection from among equiprobable outcomes, thus eventually producing a uniform distribution of values. Also see pseudorandom, really random and balance.

Some nonrandom patterns are common:

- **Counter:** There are many kinds of counter, but they all go through a sequence of values or cycle before repeating, and when a repeat happens, successive values repeat as well. But a random sequence will not have a cycle, and some values will repeat long before all values have been used (see birthday paradox).
- **Timer:** An example of a nearly ideal timer is a crystal oscillator which provides precise timing pulses with little variation. But evenly-spaced pulses are not random! The alternative of independent pulse timings (at some average rate) necessarily produces "clumps" and "gaps" which are the essence of timing randomness, and the basis for shot noise.

The usual statistical or computer-based random number generator (RNG) often is a form of counter, with one long main cycle. Having a single cycle will assure a completely flat distribution of result values, since each value will occur before any are repeated. But a counter RNG is random only in the sense that values may occur in an unexpected order or permutation. In contrast, true randomness is "clumpy," not evenly-distributed and flat.

Complex statistical RNG's have many more counting steps than values, and so avoid being mere permutation generators. But any computational RNG will cycle through fixed sequences if we can take enough values. True randomness does repeat individual values, but not a long sequence of values, and is not predictable.

A sequence of bits might be called "ideally random" if each new bit could not be consistently predicted with other than 50 percent success, by any possible technique, knowing all previous bits. Also see hypothesis.

Randomness is an attribute of the *process* which generates or selects "random" numbers rather than the numbers themselves. But the numbers do carry the ghost of their creation: If values really are randomly generated with the same probability, we expect to find *almost* the same number of occurrences of each value or each sequence of the same length. Over many values and many sequences we expect to see results form in distributions which accord with our understanding of random processes. So if we do not find these expectations in the resulting numbers, we may have reason to suspect that the generating process is not random. Unfortunately, any such suspicion is necessarily statistical in nature, and cannot produce absolute proof in either direction: Randomness can produce *any* relationship between values, including apparent correlations (or their lack) which do not in fact represent the systematic production of the generator.

Also see

- the discussions of randomness testing in Statistics and Null Hypothesis,
- my article "Chi-Square Bias in Runs-Up/Down RNG Tests," locally, or @: http://www.ciphersbyritter.com/ARTS/RUNSUP.HTM
- "Randomness Tests: A Literature Survey," locally, or @: http://www.ciphersbyritter.com/RES/RANDTEST.HTM
- "Randomness Links" on the links page, locally, or @: http://www.ciphersbyritter.com/NETLINKS.HTM#RandomnessLinks

From one point of view, there are no "less random" or "more random" sequences, since any sequence can be produced by a random process. And any sequence (at least any *particular* sequence) *also* can be produced by a deterministic computational random number generator. (We note that such generators are specifically designed to and do pass statistical randomness tests.) So the difference is not in the sequences, *per se,* but instead in the generators: For one thing, an RNG sequence is deterministic and therefore may somehow be predicted. But, in practice, extensive analysis could show deviations from randomness in *either* the deterministic RNG designs *or* the nondeterministic really random generation equipment, and this could make even a nondeterministic generator somewhat predictable.

There are "more complex" and "less complex" sequences according to various measures. For example:

- Linear complexity grades sequences on the size of the minimum shift-register state needed to produce the sequence.
- Kolmogorov-Chaitin complexity grades sequences on the size of the description of the algorithm needed to produce the sequence.

These measures produce values related to the amount of pattern in a sequence, or the extent to which a sequence can be predicted by some algorithmic model. Such values describe the uncertainty of a sequence, and are in this way related to entropy.

We should note that the subset of sequences which have a high linear complexity leaves a substantial subset which does not. So if we avoid sequences with low linear complexity, any sequence we do accept must be *more* probable than it would be in the unfiltered set of all possible sequences. In this case, the expected higher uncertainty of the sequence itself is at least partly offset by the certainty that such a sequence will be used. Similar logic applies to S-box measurement and selection.

Oddly -- and much like strength in ciphers -- the "unpredictable" part of randomness is contextual and subjective, rather than the absolute and objective qualities we like in Science. While the sequence from a complex RNG can *appear* random, if we know the secret of the generator construction, and its state, we can predict the sequence exactly. But often we are in the position of seeing the sequence alone, *without* knowing the source, the construction, or the internal state. So while *we* might see a sequence as "random," that same sequence might be absolutely predictable (and thus *not* random) to someone who knows "the secret."

Randomness Testing

The use of statistical tests to identify sometimes subtle patterns in supposedly random sequences of values.

An "ideal" random number generator (RNG) repeatedly selects among the possible values. Ideally, each value has the same probability of being selected, and each selection is made independent of all other selections. In this way, any possible sequence can be created. Although some apparent patterns will occur by chance, an ideal RNG produces no predictable patterns. But the "ideal" generator is just a goal: Real RNG's are imperfect.

Overview

The typical randomness test takes a sizable sequence and caculates a corresponding statistic value. Over many trials with ideal random sequences, the statistic values form themselves into a distribution having some particular shape as formed by the statistic computation. Typically, this distribution will have most values in the middle, sloping off at each end. Generally we can choose a fraction at each end of the distribution as values which are particularly rare. If we assert that rare events should not happen, we will not expect to find those statistic values in practice. So if we do get those statistic values from a test, we might call that "failure."

That approach is adopted from statistical hypothesis testing, where it is appropriate. When an experiment is arranged so that the desired signal will produce results which are specifically *not* random, finding a very unusual statistic value may tell us something. But that is not an appropriate approach for testing randomness where what we want is nothing unusual at all.

If we insist that a sequence with rare characteristics is bad, we should be happy with an unbalanced set of sequences which contains no "bad" ones. Such a set would be specifically and perhaps dangerously non-random, and would differ for each test. Clearly, that is not what we want.

In randomness testing, every sequence is allowed, no sequence is better than any other, and each and every statistic value produced by those sequences is valid, even if rare. On the other hand, the statistics from the sequences we test should form themselves into an expected distribution. So if we conduct many tests, and accumulate many statistic values, and then measure how close we are to the expectation, then we have an ability to decide that nothing unusual has been found.

Evaluating the closeness between two distributions will be statistical in itself. We should use a statistical test designed to detect whether or not two distributions are the same. Sometimes the result may not be a clear *yes* or *no* answer. In practice, however, substantial and marked differences are common and the decision obvious. But if we are not sure, we can just run more tests until we are. That is one advantage the hypothesis testing people generally do not have.

Finding a significant difference between the test and expected distributions is a serious event. It means what we hoped to be random is not. If we are testing for design, it means looking into the design again. If we are testing before or during operation, it means we need to cease operation and have the system re-certified. Perhaps we will have other equipment on standby to use instead. But if we are willing to use a known-bad generator, why bother testing at all?

Passing Randomness

When testing randomness, we might think to try every known statistical test and hope that none of them "fail." Unfortunately, that had better not happen.

Most ordinary statistical tests label very improbable results as "failure." But *all* sequences are possible from an "ideal" RNG, and no sequence is better than any other. When judged by an ordinary statistical test, an "ideal" random generator must land on the improbable one percent or ninety-nine percent "tails" of the distribution about one or two percent of the time; and if not, something is wrong. We thus *require* some "failure" with our "success."

Always producing sequences which some statistical test labels "pass" is explicitly wrong: A generator which always "passes" a statistical test is actually "bad." Sequences which the statistical test labels as "fail" are *required* to occur, in appropriate proportion. Accordingly, the concept of "pass" versus "fail" is probably inappropriate for testing random number generators. But since hypothesis testing so dominates statistical tests and statistical texts, many people have trouble thinking of statistics in other terms, so here is another way to look at it:

The Reversed Hypothesis

When using statistical tests in randomness testing, most tests take randomness as the null hypothesis. The research hypothesis is that the particular pattern which that test detects does exist. Since we hope to find randomness, we are hoping for the exact opposite result from that normally used in medical testing. It is normally impossible to prove a null hypothesis or establish it by experiment.

As in any hypothesis testing, there are two failure modes:

1. **False Negative.** We "fail" a good generator, which can be a Type I error.
2. **False Positive.** We "pass" a bad generator, which can be a Type II error.

In cryptography, these two failure modes have vastly different consequences:

1. If we fail a good generator, we may lose some design time.
2. If we accept and field a *bad* generator, we may lose security itself.

Ordinary statistical tests will "fail" or "reject" only a small proportion of random sequences (this is the significance level). So the two failure modes are detected differently:

1. A false negative looks like any other "fail," and is an unusual report. It is a "red flag" that something may be wrong and, at the very least, more testing is required. That is an appropriate indication.
2. In contrast, a false *positive* looks just like any other "pass," and so is easily accepted without question, despite being by far the failure with the worst security consequences.

There is a tradeoff between Type I and Type II errors, and the experimenter can vary that by setting the significance level. Decreasing the significance from a typical scientific level of $p < .05$ to, say, $p < .20$ *decreases* the probability of a Type II error, but also *increases* the probability of a Type I error. Unfortunately, no setting will eliminate either possibility, and using a low significance is somewhat unusual because it means we are increasingly "finding" patterns which are not there.

Testing for Nothing is a Different Problem

Randomness testing is different from ordinary statistical testing, because common tests are designed for use in hypothesis significance testing (e.g., medical trials), which is a fundamentally different problem: Typically, medical trials use experiments to try and decide the extent to which a particular treatment is effective. The statistics thus seek to expose the *already known signal* of a controlled treatment from within the noise resulting from different individuals and other uncontrolled parameters. Success generally consists of finding the signal and so *rejecting* the null hypothesis. And medical trials are often costly and lengthy, so there is only a limited amount of expensive data with which to work. Medical hypothesis testing is thus willing to make an educated guess about the result of a particular treatment, and is also willing to sometimes guess wrong, as this generally would happen only with the least dramatic results anyway.

In contrast to medical tests, randomness testing generally does not know what signal might be present, but instead must test for one, then another, and there is no natural end to such testing. Success, such as it is, is a failure to find the signal and a *failure to reject* the null hypothesis, yet that does not testify that the null hypothesis is true, nor does having hundreds of similar results, it just means that nothing has been found.

Randomness testing also occurs in the context of easily repeated trials, and the more independent trials one performs, the more likely it is that one will get some uncommon results. Randomness testing is thus forced to recognize that even very unusual results are both possible and valid. When unusual results do occur, trials generally can be repeated until the result is clear, one way or the other. It is impressive to add trials and see the total distribution ever more closely approach the theoretical prediction. But if not, human analysis then seeks and finds the source of the problem. Often the problem lies in the test fixture, or in the way the sequence is measured, which does not indicate a problem with the generator at all. In randomness testing, it is usually fairly clear whether or not a particular test is finding something significant.

Experiment Design for Randomness Testing

Statistics depends upon having independent random samples. However, it is entirely conceivable that some bad generators will only produce that subset of sequences which test "good" on some particular test. Since this is not a random effect, I do not see it as a Type II error, but instead attribute it to bad test design.

One way to see the issue of randomness testing is how to design the statistical testing so as to expose "false positives" and their design equivalents. Rather than concentrating on test "pass" or "fail," a better approach is to run many trials and show that the results appear (or not) in the distribution predicted for random data.

The Recommended Process

The recommended process for many types of statistical test includes running many trials of each test, each time over a different part of the sequence, computing a p-value each time.

Exactly what "many" means here is part of the test design. I normally want to see at least one trial on each of the 5 percent tails. Ideally, we might expect to find this with somewhat more than 20 trials, although we probably should be ready to conduct 50 or so. The need for this many trials will influence the size of the trial we want to use, and thus, the test design.

1. In the end, before doing anything else, we *look* at the data. Many failures will be dramatic and those results will not need statistical analysis. Moreover, it is the human analysis of structure in the result values which develops insight into how those results came to be. That insight then can be confirmed by new experiments, if necessary.
2. If we do not find results on the tails, we call the generator "bad," and analyze it to hopefully find and correct the problem.
3. We also expect to find about a quarter of the results in each quarter of the distribution: 0-25, 25-50, 50-75 and 75-100 percent. If these results are terrible and do not get better, we also stop testing and look for a problem.
4. Otherwise, we can evaluate the distribution of test results with yet another statistical test (typically, Kolmogorov-Smirnov, but sometimes chi-square can be a good fit). Typically, we just accept those results, but we might do as much additional testing as is needed to be convincing, one way or the other. Then we do the whole thing over for every statistical test we want to use.

Seeking Quality and Understanding

In my view, the point of randomness testing is to find and fix intolerable systematic faults in the generation process (for a related idea, see quality management). The point of testing is to *characterize* what the results generally are, so as to better *understand* operation, and to support *improving* the design or implementation, when desired.

In practice, the random stream is manipulated by deterministic post-processing before use. (See really random and hash.) But if the system *always* decimates the sequence before testing, or shuffles the sequence, or even accumulates multiple sequence values in each test value, the user can expect to *not* see the problems that exist. Most statistical tests each detect one simple aspect of nonrandomness, and they are easy to fool. But fooling the tests is hardly enough to get an unpredictable sequence.

Seeking to hide problems, instead of finding and dealing with imperfection, may be working the wrong side of the issue. True randomness *cannot* be certified by mere testing. (In fact, most statistical RNG's are specifically designed to produce good randomness test results, and yet have little or none of the cryptographic strength of a really random sequence.) However, a particular physical really random source can be *verified* by tests which find a distribution similar to that anticipated by theoretical analysis. Then, *after* such verification, the source can be *assumed* random, and accumulated and flattened into an even distribution of symbols or values. However, users cannot perform a random source verification when the raw really random sequence is not made available.

Every statistical test is one measure of non-randomness, but the results are statistical instead of definitive. Testing simply cannot guarantee that a particular sequence has no useful pattern or predictability.

Random Number

A value, selected at random, generally from a given range. There is nothing "random" about the value; the randomness occurs in the selection process. See random number generator.

Random Number Generator

A random number generator (RNG) is a standard computational tool which creates a sequence of apparently unrelated numbers which are often used in statistics and other computations. (For some examples, see the Random Numbers in C conversation, locally, or @: http://www.ciphersbyritter.com/NEWS4/RANDC.HTM). Also see cryptographic random number generator.

RNG's as Computation

In practice, most random number generators are deterministic computational mechanisms (FSM's). Each result, and each next state, is directly determined by the previous state of the mechanism, all the way back to the original seed. Such a sequence is often called *pseudo*-random, to distinguish it from a really random sequence somehow composed of independent unrelated values. It is normally possible to set or reset an RNG to a particular state value with a seed.

A computational random number generator will always generate the same sequence if it is started in the same state. So if we initialize the state from a key, we can use the random number generator to shuffle a table into a particular order or permutation which we can reconstruct any time we have the same key. (See, for example: "A Keyed Shuffling System for Block Cipher Cryptography," locally, or @: http://www.ciphersbyritter.com/KEYSHUF.HTM.)

RNG's and Randomness Testing

Note that statistical random number generators are specifically *designed* to pass the various statistical tests of randomness, so we know that statistical tests do *not* detect many RNG patterns. The issue is not whether a particular RNG *has* patterns, because the generation pattern is the *definition* of a particular deterministic RNG. And since virtually the whole point of using an RNG is to produce long sequences, it is interesting to note that only a tiny fraction of all possible sequences can be produced after the sequence length exceeds the amount of internal RNG state.

Nor can statistical tests certify a really random sequence. If we define "random" as "the absence of any pattern," the only way we could validate a supposedly-random sequence is by checking for every possible pattern. But for long sequences there are far too many patterns, so "real" randomness would seem to be impossible to certify experimentally *by any means at all*. And one common characteristic of randomness is that various transient patterns do occur: Randomness is usually "lumpy," and only rarely "smooth." (Also see the discussions of randomness testing in statistics and null hypothesis.)

"Patterns" are *inherent* in random sequences. If we define "random" as an arbitrary selection among all possible sequences of a given length, that ideal universe will include "all zeros," alternating bit values, counting values, and all other patterns which fit in that length. In this case, what we really want is for all sequences (and all patterns) to occur with their appropriate statistical frequency, but that is also generally impossible to certify.

RNG's and Cryptographic Uncertainty

Normally, the uncertainty in any deterministic mechanism or FSM (such as an RNG) is limited to just the amount of state in the mechanism, plus knowledge of the mechanism itself (which is usually assmed). In the sense of unknowable randomness, the maximum amount of entropy in an RNG usually is just the entropy involved in the selection of a particular internal state.

Most RNG designs are intended for statistical work only, where nobody is actively trying to develop the internal state and predict the rest of the sequence. When used as a stream cipher confusion sequence generator, statistical RNG designs must be in some way "hardened" or "isolated." See

- jitterizer
- "The Cloak2 Cipher Design" locally, or @: http://www.ciphersbyritter.com/CLO2DESN.HTM
- "A Keyed Shuffling System for Block Cipher Cryptography" locally, or @: http://www.ciphersbyritter.com/KEYSHUF.HTM

Sometimes there is a desire to test for and remove so-called "weak" sequences (e.g., constants and simple patterns) in stream cipher keying. Doing that, however, seems somewhat analogous to the German WWII Enigma cipher, which did not allow a character to be enciphered into itself (since that "exposed" plaintext). Alas, that feature of Enigma created a statistical bias or unbalance that helped in breaking the cipher. By deleting "weak" sequences from stream cipher keying, we risk creating new exploitable structure where there once was none.

Also see:

- "The Efficient Generation of Cryptographic Confusion Sequences," locally, or @: http://www.ciphersbyritter.com/ARTS/CRNG2ART.HTM
- "RNG Implementations: A Literature Survey," locally, or @: http://www.ciphersbyritter.com/RES/RNGENS.HTM
- "RNG Surveys: A Literature Survey," locally, or @: http://www.ciphersbyritter.com/RES/RNGSURVE.HTM
- "Randomness Links" on the links page, locally, or @: http://www.ciphersbyritter.com/NETLINKS.HTM#RandomnessLinks

Random Sampling

In statistics, chance selection such that every element in the population has an equal chance of being selected, each selection being independent of every other. Also see sample and balance.

Random Variable

In statistics, a term or label for an unknown value. A variable which takes on a particular value due to chance.

A *discrete* random variable takes on a finite set of values. The probability of each value is the *frequency function* or *probability density function*, and the graph of the frequency function is the frequency distribution.

Random Walk

A classic simplified model of the statistical physics of particle diffusion and Brownian motion. In the random walk model, a particle is started at some position, and proceeds randomly to some adjacent location at each time step, with some given probability of choosing any particular direction.

In the simplified one-dimensional model, a particle must move either left or right with equal probability in steps of unit length. As a consequence, the probability of the particle being in any particular position at a given time takes on a normal distribution centered on the original position, and the variance increases linearly with time.

A random walk can be described as a Markov process, and inherently requires a memory or state variable for particle position.

Range

1. The distance from some base position in some direction.
2. All values between two given limits; the values allowed for a variable.
3. In descriptive statistics, the numerical difference between the largest and the smallest values in some grouping.
4. In abstract algebra, the set of the outputs or results from a mapping or function *f(x)* for all allowed input values or arguments *x*. Also see: domain.

Rationalization

The process of starting with a conclusion to be proved, and developing potentially false arguments to support that conclusion. As opposed to reasoning.

Ratio Transformer

An electronic instrument which divides the amplitude of an AC voltage by a selectable precise ratio. The transformer-based design is usually far more accurate than resistive voltage dividers. See

- "GERTSCH RATIO TRANSFORMERS" locally, or @: http://www.ciphersbyritter.com/RADELECT/RATIOTRN/RATIOTRA.HTM

RC Filter

Typically, a resistor and capacitor in series as a frequency-varying voltage divider. Signal is applied across the outside or non-joined terminals. The result across the resistor is high-pass filtered, while the result across the capacitor is low-pass filtered. RL (resistor-inductor) filters are similar but inverse. LC filters are resonant "tanks" that peak at a particular frequency.

In the RC low-pass case, the output is constant up to a turnover frequency where the "loss" is 3dB (the signals across R and C have the same amplitude), then the loss increases at 6dB/octave (20dB/decade) at higher frequencies. The high-pass case is similar, with high frequency signals being constant down to the turnover frequency where the loss is 3dB, with signal decreasing by 6dB/octave below that.

For resistance *R* in Ohms, capacitance *C* in Farads and frequency *f* in Hertz, RC filters have the following relations:

```
f = 1 / (2 Pi R C)
R = 1 / (2 Pi F C)
C = 1 / (2 Pi F R)
(2 Pi R C F) = 1
Pi = 3.14159...
```

RC filter part values typically are constrained by availability: For some fixed target frequency, we can pick an available capacitor (C), then compute the necessary resistor (R). Typically that will not be available, so we can check the closest R values we have and compute a resulting frequency (F) for each. If not satisfactory, other values for C can be checked. Alternately, a good approximation to a particular R value can be made from a series or parallel connection of multiple resistors. Some compromise to the desired frequency is common and typically innocuous, since the starting roll-off is gentle and covers a range of frequencies anyway.

RC filter part values are also constrained by the source and load impedances. Typically we expect:

- The input signal will come from a source with much lower internal impedance than R (say, R/10 or lower). If not, the source impedance must be considered part of R, up to the case where no external R is needed. If the source impedance is higher than that, the chosen C cannot produce the desired filter.
- The filtered output signal is used by something with a much higher impedance than R (say, 10R or higher). If not, the effective generator impedance is lowered, changing the filtering start frequency, and the signal is divided, reducing the output level.

It is very common to use multiple RC stages to increase the roll-off rate to 12dB/octave or 18dB/octave or more. However, unless subsequent stages have a higher impedance, or are isolated by an op amp buffer, they will load earlier stages and change the turnover frequency. Also, the sharpness of the starting rolloff cannot be improved by using multiple RC stages.

Reactance

In electronics, the frequency-sensitive analog to resistance produced by pure inductance and capacitance.

Really Random

A random value or sequence derived from a source which is expected to produce no predictable or repeating relationship between values. Also called physically random and truly random or TRNG. As opposed to pseudorandom (see random number generator).

Examples of a really random source might include radioactive decay, Johnson or thermal noise, shot noise from a Zener diode or reverse-biased semiconductor PN junction in breakdown, etc. Clearly, some sort of circuitry will be required to detect these generally low-level events, and the quality of the result is often directly related to the design of the electronic processing. Other sources of randomness might be an accumulated hash of text of substantial size, and precise keystroke timing.

Unfortunately, although we expect quantum-like effects to be "very random," science is not in the business of finding or guaranteeing physical "law" upon which we can base provably unpredictable generators. Instead, science generally develops quantitative models that are either confirmed or contradicted by experiment, but even repeated confirmation does not indicate a "law." In fact, it is not all that unusual for experiments to indicate that what previously was accepted as physical law will have to be updated and changed. That means even atomic physics is not an unarguable basis for unpredictable randomness as is often assumed. That said, quantum sources can avoid many problems inherent in other randomness techniques.

A prime example of a *non*-random generator is the ubiquitous digital computer, which unfortunately is also the main platform for modern cryptography. Technically, a computer is no more than a complex finite state machine, and is absolutely deterministic in everything it does. All a computer can do is follow a sequence of operations which, if not cryptographically created and protected, must be assumed to be exposed to the opponent. The computer, the operating system, disk devices and disk structure are all common production items which the opponent is expected to have and know much more about than we do. Granted, computer operations can be very complex. And anyone can *claim* that "nobody" could unravel the complexity, but proving that to cryptographic levels of certainty is another story. Mere complexity is not the same as cryptographic unpredictability or strength.

Really-random generators are the exact opposite of the highly-controlled software and digital hardware that we know how to design and test. Often, there are unexpected problems. For example:

- Timing keystrokes on an ordinary PC actually measures the result reported by the slow keyboard processor. Since the keyboard performs a periodic scan of keys, each keypress is quantized probably thousands of times more coarsely than implied by PC timing.
- Geiger-type radioactivity detectors have a sensitivity which depends upon bias voltage. So if the bias voltage changes, the probability of finding an event within some given time also changes. Because the events are random, the difference may not be obvious.
- Thermal noise is extremely small, making detection highly dependent upon particularly clean and wideband electronic amplification.
- Shot noise will vary with current, and so needs to be developed in some way which keeps current relatively constant over time.
- Zener diodes, which typically operate by avalanche multiplication, may have a disturbing and generally unnoticed tendency to oscillate, which would be unexpected predictability.
- Noise-based techniques can have substantial correlation between samples, starting with adjacent samples and extending on for perhaps tens of samples.
- Crystal oscillator phase noise is unlikely to be accurately measured by repeated software sampling on a PC platform. Phase noise is a tiny cycle-by-cycle effect which does not accumulate over multiple timing cycles.
- Crystal oscillators generate ultra-predictable waveforms. When combined or sampled, the result may be similar to a solvable linear equation.
- Sequences from various computer hardware sources may include a a complex but deterministic interaction between periodic interrupts and software timing, and so could be somewhat predictable while appearing random.
- Sampling asynchronous digital signals or analog noise directly using digital devices (including simply reading an input port) often will fail to meet the setup time and hold time requirements of digital logic. The result will be to bias the sampled data, possibly cause metastable operation, and operate differently on different machines.
- Collecting various external network timings is not particularly helpful if the network itself is not secure, since an opponent could collect the exact same timings.
- Collecting various internal computer timings and values, such as disk access and so on, ignores the fact

that these are all largely repeatable and predictable values when the state of the drive and program set are known. But that state is rarely protected, because cryptographic strength is generally assumed to reside in the cryptosystem, not the physical computer.

In general, a computer is the ultimate deterministic and predictable machine, even if that machine is very complex. Consequently, it is generally impossible to develop unpredictable random numbers from a computer alone. Any claims of unpredictable randomness from a computer must be analyzed extremely closely. Merely passing statistical randomness tests cannot certify that a generator produces unpredictable sequences. The real issue is knowing where any claimed randomness comes from, how much there is, and whether new randomness is developed over time.

Post-Processing

Many random generators include a post-processing step intended to improve the results of statistical randomness testing. Unfortunately, the processing prevents us from peering into and measuring the quality and quantity of the really-random input. Again unfortunately, most post-processing will produce statistically random results *even with* ***predictable*** *source values*. So from the outside we are generally unable to distinguish between a processed quantum unpredictable source and the deterministic RNG produced by post-processing alone. As a result, we are generally unable to certify that a quantum process is the ultimate source of the generated values. We are instead forced to trust what cannot be trusted: that the device designers, production, testing, and corporate sales have all worked for us and against our opponents to bring us a security device which cannot be influenced or predicted. Thus, it is vital to use really random generator designs which can expose the quantum data, and to bring that data out for analysis.

Yet another problem with post-processing is a peculiar desire for efficiency; that is, to produce for use every bit of randomness collected. Unfortunately, the more efficient post-processing is, the more likely it is to be reversible, which is exactly the wrong goal. An unprocessed random stream is likely to have a statistical bias or non-uniform distribution, and thus would be partly predictable. In cryptography, instead of *efficiency*, our first goal ought to be *security*, and in this case that implies "*in*efficiency." Post-processing should discard, or over-collect, at least as much information as it produces (and perhaps much more raw really-random data).

It is easy to hide the original data, because even a simple linear CRC hash is *guaranteed* to be non-reversible, *provided* only that far more information enters than is extracted. We do not have to simply believe that CRC is strong, because CRC is not strong. CRC does, however, absolutely limit the amount of information which can be represented in internal state. If we overload that state by at least a factor of two, we know absolutely that the function cannot be reversed to a unique value useful in analyzing the original source. Also see hash.

Really random values are particularly important as message key objects or other nonce values, or as a sequence for use in a realized one-time pad.

Also see:

- "Random Number Machines: A Literature Survey," locally, or @: http://www.ciphersbyritter.com/RES/RNGMACH.HTM
- "Random Electrical Noise: A Literature Survey," locally, or @: http://www.ciphersbyritter.com/RES/NOISE.HTM
- "The Hardware Random Number Generator" conversation, locally, or @: http://www.ciphersbyritter.com/NEWS4/HARDRAND.HTM
- "The Pentium III RNG" conversation, locally, or @: http://www.ciphersbyritter.com/NEWS4/PENTRAND.HTM
- "Randomness Links" on the links page, locally, or @: http://www.ciphersbyritter.com/NETLINKS.HTM#RandomnessLinks

Real Number

Numbers which can represent a continuous quantity. Denoted **R**. As distinguished from imaginary numbers, which with reals form complex numbers.

Reasoning

The process of using known facts to disclose previously unknown facts; the process of using known statements to form new, correct statements and expose existing false statements. As opposed to rationalization. See: logic, scientific method, inductive reasoning, deductive reasoning and fallacy. Also see: argumentation.

Red

In TEMPEST, electrical wiring or signals carrying plaintext information which must be protected and can **not** be exposed.

Redundancy

In information theory, the measured relative entropy subtracted from 1.0, the maximum possible value.

Relation

A set of ordered pairs.

Relative Entropy

In information theory, the ratio of the actual or measured entropy to the maximum possible with the same set of symbols. Also see redundancy.

Relatively Prime

In number theory, two positive integers which have 1 as their only common factor. Two integers *x*, *y* with the gcd(*x*,*y*) = 1, written as (*x*,*y*) = 1. Also see: congruence and prime.

Relay

Classically, an electro-mechanical component consisting of a mechanical switch operated by the magnetic force produced by an electromagnet, a conductor wound around an iron dowel or core. A relay is at least potentially a sort of mechanical (slow) and nonlinear amplifier which is well-suited to power control.

Reliability

1. The extent to which some one or some thing will perform in the future as they have in the past.
2. The ability to perform a function under given conditions for a given period.
3. The *probability* of a system performing correctly as opposed to failing.

Reliability is a engineering quality that can be measured after the fact, and then predicted in a statistical sense. Unfortunately, cryptographic strength is neither measurable nor predictable. Also see trust.

Re-Originating Stream Cipher

My term for a stream cipher which re-uses confusion sequence.

Even though the confusion is probably generated by a keyed generator, re-starting that sequence without changing the key usually is A Very Bad Thing. The potential advantage would be the ability to eliminate key-change overhead on every packet ciphered, and to allow packets to be deciphered independently without attaching a "packet key" to each one.

In a conventional additive stream cipher, two uses of the same confusion stream allow the confusion to be "cancelled" leaving only two mixed plaintexts, which is horribly weak. Thus, reasonable re-originating stream ciphers would seem to be the exclusive province of autokey operation with keyed nonlinear and dynamic combiners. But even in the best case such a cipher is likely to be weak against defined plaintext attack and must be analyzed very, very carefully.

Research Hypothesis

In statistics, the alternative hypothesis H_1. The research hypothesis is the nonrandom pattern which we seek in randomly-sampled results. We formulate the research hypothesis so that if it is wrong, the null hypothesis is not discarded except by the small chance of the significance level. The research hypothesis is supported when the null hypothesis is shown to be improbable.

Frequently, the research hypothesis will just indicate that something nonrandom has happened; that some signal or predictable data have been found. The problem is that any unexpected happening including experimental design error can provide such an indication. Care must be taken that the signal found is what was sought.

Sometimes, the research hypothesis is formulated such that if the proposed model is correct, the test statistic distribution will be have some unique but predictable structure. To find a particular statistical distribution, many things have to work as expected, including the conduct of the experiment and the functioning of the theoretical model. When the experimental results have to match a *particular* expected distribution, this formulation has the potential to produce a strong quantitative validation of the model.

Resistor

A basic electronic component which slows or obstructs current flow and dissipates energy in the form of heat. Measured in Ohms, denoted with a sort of upside-down U which is a Greek capital omega: Ω (see Greek alphabet).

In a resistor, voltage and current are linearly related by Ohm's Law: **E = IR**.

- Resistors can thus be used to limit current I given voltage E: (I = E/R), or to produce voltage E from current I: (E = IR).
- Since power in watts is related to voltage and current as P = EI, resistors dissipate heat according to P = I^2R (that is, P = I*I*R).
- Two same-value resistors in series can divide voltage to half the total. More generally, two resistors divide Ein to produce the output Eo = Ein(R1 / (R1+R2)). This is the basis for a wide range of electronic actions, from volume control to voltage amplification to filtering.

Resistors in series are additive. Two resistors in parallel have a total resistance which is the product of the resistors divided by their sum.

Precision

Precision relates an actual component value to the resistance value as marked. Classically, resistors were first manufactured and only then measured and marked with a value. Thus, every possible resistance value needed to be within a given precision of some standard value.

Classically, the common precision was 20 percent. Modern precision is typically 5 percent or even 1 percent, but that requires many more standard values. Retaining a stock of each value can be both expensive and difficult.

A better approach nowadays might be to collect a few particular values, repeated for at least 7 decades from ohms to megohms. One surface-mount assortment included the following values:

```
   1.00   1.21   1.50   2.00   2.43   3.01
   3.92   4.99   6.19   7.50   9.09
```

Marking

Modern surface-mount resistor values are often laser-etched in 3 or 4 digits. Classically, resistor values are indicated by color bands with the color code:

0	Black
1	Brown
2	Red
3	Orange
4	Yellow
5	Green
6	Blue
7	Violet
8	Grey
9	White

Normal resistors have 3 value bands, the first two being digit values and the third a multiplier. For example, a 100k resistor would be brown-black-yellow, 1-0-4, for 10 with 4 zeros (0000) or 100000.

Resistors also may have a precision band:

20 pct	none
10 pct	silver
5 pct	gold

Noise

All resistors produce thermal noise proportional to absolute temperature and resistance. Some resistors (e.g., the old carbon composition and modern thick-film surface mounts) also produce additional excess noise proportional to current, resistance, and probably temperature as well. Excess noise is not a problem when system noise is larger, or when negligible current is involved (as in most op amp applications). In very low-noise circuits that carry current, metal-film resistors will be a better choice.

Also see capacitor, inductor, conductor, negative resistance and resistor excess noise.

Resistor Excess Noise

In electronics, all resistors generate thermal noise, but some generate noise beyond that. Few if any resistors generate shot noise since that requires electrons to have independent timing, which typically comes from traversing a semiconductor junction. (Once electrons collect into a conductor or resistor, they become correlated and are no longer independent.) The remaining or excess is usually 1/f noise and is pink (predominantly low-frequency), instead of white (flat).

Resistor excess noise is generated in non-homogenous resistances, such as the typical thick-film surface-mount resistor which is a mixture of conductor particles and insulating glass. It is thought that the current path through the conductive grains can vary dynamically at random, thus modulating the resistance. Excess noise is generally proportional to resistance and current, and is insignificant when current flow is negligible.

Excess noise occurs in proportion to the current flowing per unit area of resistive material, and can be reduced by expanding the resistive area. To reduce excess noise a larger (higher power) resistor can be used, or multiple resistors can be connected in parallel. All things being equal, smaller SMT resistors should be more noisy than larger ones, when carrying current in low-level circuits.

Metal film resistors based on a homogenous film construction generate little or no excess noise.

Resonance

A property of a physical system which can store energy in oscillations or vibrations at some natural frequency. The classic example is a pendulum, where energy moves between the potential energy of height against gravity, and the kinetic energy of motion in velocity with the inertia of mass.

Another example of resonance is a bell, which, when given energy from a single impulse, stores that energy at the natural resonant frequency of the bell and "rings" while dissipating that energy over time as sound. The sound vibrations correspond to the flexing and inertial motion of the physical bell material, the movement causing the displacement of air thus producing sound.

Yet another example of resonance is the quartz crystal, in which electrical energy is translated to and from physical vibrations in the crystal blank.

In electronics, inductors and capacitors form resonant systems which select or reject a particular frequency. In an ideal (non dissipative) resonant system, for *f* the frequency in Hertz, *L* the inductance in Henrys, and *C* the capacitance in Farads:

```
f = 1 / (2 Pi SQRT( L C ))
L = 1 / (4 Pi Pi f f C)
C = 1 / (4 Pi Pi f f L)
Pi = 3.14159...
```

At resonance, XL = XC, so (2 Pi f L) = 1 / (2 Pi f C).

For *f* in MHz, *L* in uH, and *C* in pF:

```
L = 25330 / (f f C)
C = 25330 / (f f L)
```

Because energy is stored in a resonant system over many cycles, small signals at the resonant frequency can accumulate and lead to large excursions of voltage or current; this is the resonant gain. And because energy is stored over many cycles, signals at other frequencies tend to be rejected, and small pulses or noise have a minimal effect on the pure resonant signal.

In practice, real resonant systems do dissipate energy. The most common reason for loss is resistance in the inductor, but capacitors can have resistance (or leakage) as well. This loss tends to broaden the resonant response and reduce resonant gain.

Reverse CRC

"Reverse" means that the CRC polynomial is intended to operate on bit-reversed data. This was needed for large 60's-style computer reel-to-reel tapes where the data could be read from either direction.

"Reverse" also means that the CRC polynomial is reversed from the original: If we write CRC-16 as 0x18005, CRC-16 reverse will be 0x14003, which is the same bit pattern read from opposite directions.

The way the reverse operation works depends upon the way CRC is developed and deposited; this is the forward direction. The most trivial way is to init the CRC register as zeros, CRC the data, then deposit the CRC result, MS byte first. Then, when we traverse the data plus the stored CRC result without error, the CRC result will be zeros.

To traverse the data in reverse direction, we again init the CRC register as zeros, use the inverse CRC polynomial, and process the data bits in reverse (starting with the lsb of the saved CRC value). Again, the CRC result will be zeros. Now we can error-check the data as it comes in, even if it comes in backwards.

It is also possible to init as ones (which is more desirable), and to save the complement of the CRC value (so the result will be a known non-zero value), and then also check in reverse, but doing that is more complicated.

Rhetoric

The art of persuasive speaking.

However, in *Gorgias*, Socrates appears to view rhetoric as a *habit* of "flattery" which persuades by creating belief (as opposed to imparting knowledge). Socrates also seems to group rhetoric with cookery, attire and sophistry as the persuasive tools. See: dialectic and argumentation.

Ring

In abstract algebra, a commutative group under (+), with a second dyadic operation (*), which is associative and distributive.

A nonempty set R with two dyadic (two-input, one-output) operations which we choose to call "addition" and "multiplication" and denote + and * as usual. If elements (not necessarily numbers) a, b are in R, then a+b is in R, and ab (or a*b) are also in R. The following properties hold:

1. **Addition is commutative:** a + b = b + a
2. **Addition is associative:** (a + b) + c = a + (b + c)
3. **There is a single "zero" or additive identity element:** a + 0 = a
4. **There is a corresponding additive inverse for every element:** for any a in R, there is an x in R such that a + x = 0
5. **Multiplication is associative:** (ab)c = a(bc)
6. **Multiplication is distributive:** a(b + c) = ab + ac and (b + c)a = ba + ca
7. **In a commutative ring, multiplication is commutative:** ab = ba
8. **In a ring with unity, there is a single "one" or multiplicative identity element e:** for e in R, ea = ae = a

The integers form a ring under addition and multiplication (Z,+,*). And Z/n, the set of integers modulo n is a ring under addition and multiplication mod n. A field is a ring where the second operation forms an abelian group. Any field contains a ring, and any ring contains a group.

Ringing

The decaying oscillation produced by the impulse excitation of a resonant system. In electronics, typically due to a resonant LC "tank" circuit. Analogous to the audible ringing of a cast metal bell after it has been hit.

Ringing often is apparent when testing an amplifier system with a square wave signal and viewing the result on an oscilloscope. Solving ringing problems generally involves identifying the resonant system, then adding resistance to dissipate energy stored in that system. The trick is to somehow retain efficient normal operation. This is easiest when the resonance is at least 100 times the highest desired frequency (thus, over 6kHz for 60Hz power and over 2MHz for hi-fidelity audio), so that the dissipating resistor can be enabled with a small capacitor. The resulting RC is called a "snubber," and the main role is to dissipate power stored in resonance, with a high-frequency enable. Snubbers are also used to protect switch or relay contacts from arcing when switching potentially-inductive loads such as transformers or motors.

Low-power signals allow other possibilities. One is to hide the ringing behind low-pass filtering. An option for high-frequency signals is to use ferrite beads or cores selected for loss above the desired band.

A related issue is that dynamic loudspeakers use voice coils which are inherently inductive, and at higher audio frequencies the inductance has a high impedance. That can emulate a dynamic loss of load, which might expose a resonance or instability in the amplifier. A common fix is to put a series RC across the speaker, much like a snubber. In this case the RC is called a "Zobel network," and one motivation is to shunt the rising speaker impedance with a power resistor to get a relatively constant load across frequency. A constant load impedance is also important for speaker crossover networks.

Risk

A remarkably ambiguous term typically used in reasoning about what *might* happen, typically negative outcomes. In contrast to evidence, which is used when reasoning about what *did* happen.

1. **An event** that could happen, presumably negative. Also called the hazard.
2. **The probability** of some hazard. In general, this will be more informative when cast as a statistical distribution across a range of time, or other variable. The distribution exposes the best case, the worst case, the most probable case, and everything else.
3. **The value** that may be lost as a consequence of a hazard. That which is "at risk." The *loss potential* or *possible cost*. Sometimes the *stakes* or *consequence*.
4. **The expectation** of loss. The product of event probability and loss value. Also called the exposure.

In cryptography, *risk* is often used with respect to the probability that a cipher may have been or will be broken by one or more opponents. Unfortunately, **nothing at all** can be known about such risk. The events of interest occur only in secret, and we will not be informed. Without results, there is no way to develop a probability or distribution, or to know if any estimate is right or wrong. There simply is **no way to know** the risk that a cipher is technically insecure. Nor is any valid comparison possible between an unknown quantity and other unknown quantities, such as the risks of operational error or human exposure, or other ciphers with no known break. All that we can do is say that a cipher break is always possible, and that no cipher whatsoever can be completely trusted. Because breaks occur in secret, even long, widespread use is not a rational basis for increased trust.

A standard cipher increases risk by having the highest possible stakes and limiting attack costs to a single cipher. Standard cipher *interfaces* (with a continually increasing set of ciphers) divide the stakes into separate channels and multiply the attack costs. Multiple encryption adds redundancy to counter the risk of single points of failure.

Risk is about potential loss. When we have a choice, the very idea of placing something at risk makes no sense at all unless there is a reward to be gained. All risks (and costs) of a bad outcome need to be seen in the context of the potential *benefit* to be obtained from the desired outcome. (Also see risk analysis and risk management.)

Risk Analysis

(Also called "risk assessment.") The process of identifying potential risks or hazards, finding their probability, consequences, and expectation, presumably so those can be addressed in risk management. Also called "failure analysis," "risk assessment" or many similar terms used in a wide variety of engineering areas, such as:

- Safety Engineering,
- Reliable System Design,
- Reliability Engineering,
- Fault Modeling,
- Fault Tolerance, etc.,

as well as public health and safety.

Two widely-used techniques for such analysis are: Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA).

In cryptography, we cannot know when our well-reviewed, highly-trusted and widely-used cipher has been broken by our opponents. Because we do not know when it happens, we also cannot develop a probability of that happening. We do know the risk exists, because no cipher has an absolute proof of strength which applies in practice. And if our cipher is broken, we will continue to use it and continue to expose our data because we will not know any better. (Also see threat model.)

Cipher failure is exactly the sort of single point of failure that aerospace engineering looks for and handles by adding redundancy. In cryptography, encryption with multiple ciphers could be used so the failure of one cipher would not affect system security. In addition, many different ciphers could be used, so that the failure of any one could expose only a small fraction of the total traffic. And ciphers could be changed frequently, thus terminating any existing break.

Risk Management

The planning activity that seeks to address risks identified in risk analysis, typically by reducing either the probability or the consequences.

Nobody can predict the future. But we can describe various possible outcomes, then imagine what each would cost and about how likely each would be. Then we can seek to direct the future toward the desirable outcomes, and consider how to mitigate the effects of the undesirable, should one of those occur despite our efforts.

Risk management is often applied in life-critical engineering areas like airplane design and construction, and in more general areas like public health and safety. Risk management could be described as:

1. continually seeking to identify, monitor, and reduce the probability of component or module or subproject failures, and
2. seeking to prevent any single failure from endangering the entire system or project.

Since unfounded beliefs or *assumptions* are inherent risks, risk management could be described as assumption management.

A significant recent book oriented toward risk in Software Engineering development projects is:

DeMarco, T. and T. Lister. 2003. *Waltzing with Bears.* Dorset House.

The best option would be to read the book, and perhaps give a copy to everybody on your development team. My interpretation of the process they describe for managing risk in software development goes something like this:

1. **Analyze plans to expose risks** ("hazard identification" in risk analysis).
   - Anything we *assume* will be available, may not be.
   - Anything we *assume* will happen in the future, may not, and that is a risk.
   - Anything we *assume* someone else will do, or will accept, or even be around to do, is also a risk.
   - Various parallel situations in the past can be examined to see what went wrong.

   In this way we can collect a list of potential problems. The risk identification and management processes should continue throughout the project. Some obvious common risks include:
   - unworkable schedule,
   - an expanding project spec,
   - employee loss, and
   - customer negotiation failure.

   (See *Waltzing with Bears* for discussion.)
2. **Quantify the likelihood of each problem occurring, and the cost if it does** ("risk assessment" or risk analysis). In most cases, there will be a distribution of probabilities and cost, as in the case of the probability of schedule overrun. That is good, because a distribution is more likely to be right than a prediction of any fixed, exact date. Moreover, completion probability versus time distributions for each of the various modules can be combined to give a similar distribution for the overall system.
3. **Assign responsibility.** The responsible party is whoever will have to pay if the risk occurs. Responsibility is often a contract issue, and, in general, there is no contract in which one party carries all risk. Certain risks are accepted and "owned."
4. **For each owned risk:**
   - abort, decline or otherwise "avoid" the activity which contains the possible problem, or
   - allocate extra resources sufficient to handle, correct or otherwise "contain" the problem if it occurs, or
   - additionally invest in and prepare a lower-cost alternative to "mitigate" the problem if it occurs,

or
- do nothing and hope to "evade" the problem by mere dumb luck.

5. **For each non-avoided risk**, find an indicator that will precede the negative situation, so the risk does not arrive unannounced.
6. **Monitor project progress** with some *quantitative* measure. The traditional software measure, lines of code (LOC), is not particularly useful because it scores large poorly-programmed routines better than small good ones. A better approach is to deliver a sequence of pre-release programs, with the hardest modules done first. Then a reasonable measure might be the simple percentage of routines finished, quality approved, and functioning in a pre-release. (Also see Software Engineering and Structured Programming.)

RMS

root mean square.

RNG

Random Number Generator. Or, for the pedantic among us, PRNG.

"RNG" is just the abbreviation for the classic computer term "Random Number Generator," used since at least the mid-1960's for the statistical sequence generators implemented on computers. Both the computer and statistics literatures call those objects "RNG's" -- not "PRNG's" -- despite knowing very well that they are deterministic finite state machines (like everything else on a computer) and so are *pseudo*-random.

If we wish to talk about actual, instead of pseudo, randomness, we can say "really random" or "physically random" or "noise based" or "radioactivity based," but a common abbreviation is TRNG for "truly random number generator." The cryptographic goal, however, goes beyond mere randomness toward a generator whose sequence is unpredictable. It is the unpredictability of a random source, and not just one-time use, which makes a one-time pad secure. Perhaps in the future "URNG" may make sense but, currently, CRNG or cryptographic random number generator is popular.

Root

A solution: A value which, when substituted for a variable in a mathematical equation, makes the statement true.

Root Mean Square

RMS. The square root of the integral of instantaneous values squared. Thus, when measuring voltage or current, a value proportional to the average power in watts, even in a complex waveform.

Round

In block cipher design, a sequence of simple operations repeatedly applied as a unit. The actual operations may include simple substitution, fixed permutation and additive combining with a key. A simple ciphering designed for use as multiple encryption.

In typical use, rounds are repeated multiple times with different keying. As opposed to a layer, which is not applied repeatedly. Used in product ciphers and iterated block ciphers.

RSA

The name of an algorithm published by Ron Rivest, Adi Shamir, and Len Adleman (thus, R.S.A.). The first major public key system.

Based on number-theoretic concepts and using huge numerical values, a RSA key must be perhaps ten times or more as long as a secret key for similar security.

Rule of Thumb

A likelihood. A rule that is normally true, but may be an approximation or simplification and may have exceptions. Not necessarily refuted by counterexample. See: accident fallacy and heuristic. Note that special pleading is the fallacy of claiming exceptions based on irrelevant issues.

A rule of thumb may seem to lack the purity one might expect in this day and age, but both cryptanalysis and, frankly, science itself are rarely pure. See scientific method, correlation and independence. Cryptanalysis seeks rules that *often* work, or even just *occasionally* work, and only rarely requires mathematical perfection.

For example, a cipher which is linear is usually weak, so the strict mathematical definition of linearity might seem useful. However, a cipher which is *almost-but-not-quite* linear can be weak in exactly that same way, because it is possible to approach linearity without actually achieving it. That means even a rigorous proof that a cipher is technically nonlinear is not particularly helpful in understanding linearity as a weakness. A reasonable alternative is to develop a *measure* of distance from linearity; see, for example, Boolean Function Nonlinearity.

Another absolutist quality which can be misleading is: invertible.

Running Key

The confusion sequence in a stream cipher. Also keystream.

SAC

Strict Avalanche Criterion.

Safe Prime

Prime P, where P = 2Q + 1 for some prime Q. Also see: special prime.

Salt

An arbitrary value, unique to a particular user, attached to a password before hash authentication. That means the exact same password will give different hash results for different users, which complicates password dictionary attacks: The salt reduces the advantage an opponent might otherwise have in pre-processing and storing the hash values for the most popular passwords. The salt also hides the use of the same password on multiple systems. Also see the short "Salts" conversation (locally, or @: http://www.ciphersbyritter.com/NEWS6/SALT.HTM).

Apparently "salt" is used in the sense of "salting" a mine with nuggets, or "salting" a dictionary with invented words. Salt is sometimes misused where IV or message key would be more appropriate.

Interestingly, there would appear to be some similarity between the use of a salt to create many different hash results for the same password and homophonic substitution which creates many different ciphertexts for the same plaintext.

Sample

In statistics, one or more elements, typically drawn at random from some population.

Normally, we cannot hope to examine the full population, and so must instead investigate *samples* of the population, with the hope that they represent the larger whole. Often, random sampling occurs "with replacement"; effectively, each individual sample is returned to the population before the next sample is drawn.

S-Box

Substitution box or table; typically a component of a cryptographic system. "S-box" is a rather non-specific term, however, since S-boxes can have more inputs than outputs, or more outputs than inputs, each of which makes a single invertible table impossible. The S-boxes used in DES contain multiple invertible substitution tables, with the particular table used at any time being data-selected.

One possible S-box is the identity transformation (0->0, 1->1, 2->2, ...) which clearly has no effect at all, while every other transformation has at least some effect. So different S-boxes obviously *can* contain different amounts of some qualities. Qualities often mentioned include balance, avalanche and Boolean function nonlinearity. However, one might expect that different ciphering structures will need *different* table characteristics to a greater or less degree. So the discussion of S-box strength should always occur within the context of a particular cipher construction.

S-Box Avalanche

With respect to avalanche, any input change -- even one bit -- will select a different table entry. Over all possible input values and changes, the number of output bits changed will have a binomial distribution, so, in this respect, all tables are equal. (See the bit changes section of: "Binomial and Poisson Statistics Functions in JavaScript," locally, or @: http://www.ciphersbyritter.com/JAVASCRP/BINOMPOI.HTM#BitChanges).

On the other hand, it is possible to arrange tables so that all single-bit input changes are guaranteed to produce at least two-bit output changes, and this would seem to improve avalanche. But what it *really* does is to *re-distribute* the probability of bit changes away from multi-bit-change cases, to enhance the single-bit-change case. (And I worry that such a change may add exploitable structure to what otherwise would be a random selection among all possible tables.) We note that increasing bit-changes in the single-bit-change case is *probable* even with a randomly-constructed table, so we have to ask just how much single-bit guaranteed expansion could improve things. In a Feistel cipher, it seems like this *might* reduce the number of needed rounds by one. But in actual operation, the plaintext block is generally *randomized,* as in CBC-mode. That means that the probability of getting a single-bit-change in plaintext is very low anyway.

It is true that cipher avalanche is tested using single-bit input changes, and that is the way avalanche is defined. The point of this is to assure that every output bit is "affected" by every input bit. But I see this as more of an experimental requirement than an operational issue that need be optimized.

S-Box Nonlinearity

With respect to Boolean function nonlinearity, the highest possible value is just half of the number of bits in the analyzed a sequence. In an "8-bit" table with 256 8-bit elements, the absolute maximum nonlinearity (for any particular output bit) would be 128, but no such table may exist. However, randomly-constructed 8 bit tables have surprisingly high values of nonlinearity on average.

For an "8-bit" table, the maximum possible nonlinearity is 128 for each of 8 output bits. When we take the minimum of all 8 results, we might think to find a much lower value, and so worry about random construction. But in fact, my extensive experiments show that we should expect a random 8-bit table to have a nonlinearity (the minimum over the 8 output functions) of about 100. That is, fully 100 bits must change in the weakest of the eight 256-bit output functions to reach the closest affine Boolean function. I generalize these results by saying that random 8-bit tables are "generally very nonlinear."

Experimental measurement shows that more than 99 percent of random 8-bit tables have a nonlinearity between 92 and 104. In an experiment covering 1000000 random 8-bit tables, exactly one table had a nonlinearity as low as 78 (that is one chance in 1000000, a probability of 0.0001 percent). The computed probability of an actually *linear* table (nonlinearity zero in any one of the 8 output functions) is something like 10^{-72} (that is, 1/10**72) which is about 2^{-242} (or 1/2**242). That figure is probably less than the chance of an opponent choosing the correct key by accident. Nevertheless, weak keying is a possibility in any random construction, so a cipher using keyed tables should use multiple tables to hide any accidental weakness.

In contrast, "4-bit" tables are surprisingly weak. With 16 elements, we might think to find some tables having a Boolean function nonlinearity value as high as 8 bits. Unfortunately, in my experiments, no 4-bit table was found to be more than 4 bits away from an affine function, and almost 50 percent of 4-bit tables were only 2 bits away. (This is in stark contrast to 8-bit tables, where 99.9999 percent of all tables were found to be at least 78 bits away from the closest affine function.) I generalize these results by saying that 4-bit tables are "almost linear." Now, clearly, 4-bit tables seem to be *nonlinear enough* for DES, but when table nonlinearity is important, the weakness of even the best 4-bit tables is very disturbing when compared to the strength of 8-bit tables.

As one reference point, the NSA-designed "8-bit" table in the Skipjack cipher has a computed nonlinearity of 104, which is in the top 2.5 percent of the random tables I have measured. (When checking 10000 random 8-bit tables I expect to find about 249 tables at nonlinearity 104, and 2 tables at 106, so just over 2.5 percent of the tables have a nonlinearity of 104 or better.) The abstract chance of a nonlinearity of 104 occurring at random is about 1 in 40. So, if a decision simply *must* be made on the basis of this single trial, we are forced to accept that good Boolean function nonlinearity (or something very much like it) was an intentional part of the NSA Skipjack S-box design.

Keyed S-Boxes

The tables I used in my Boolean function nonlinearity experiments were constructed by shuffling a standard table using a sequence from one of my cryptographic random number generators. My RNG's typically consist of a very large (and linear) additive RNG, and a jitterizer nonlinearizing filter. These generators are keyed by hashing a user key phrase, and then expanding that through multiple levels of increasing larger RNG. See my descriptions:

- "The Cloak2 Design," locally, or @: http://www.ciphersbyritter.com/CLO2DESN.HTM#Components
- "A Keyed Shuffling System for Block Cipher Cryptography," locally, or @: http://www.ciphersbyritter.com/KEYSHUF.HTM
- "Dynamic Transposition Revisited Again," locally, @: http://www.ciphersbyritter.com/ARTS/DYNTRAGN.HTM in the "KEY EXPANSION" section.

The result was simply the resulting box; the boxes were not examined before measurement; none were discarded for any particular characteristics.

Both key expansion and the table shuffling do take some time (although not *human* time; shuffling is fairly fast). So if we could measure the unknown strength of ciphers, presumably we might find that another approach would be both faster and better. As we cannot do that, however, time spent creating a huge amount of complex, keyed, internal state may be time well spent.

The point of shuffling is to create any possible invertible table with equal probability. This is essentially making a keyed selection among all possible tables. Keying is a very general concept in cryptography: After we first assure that we have "enough" possible keys, the next rule of thumb is that each key should be essentially the same as any other. Accordingly, it seems particularly unwise to talk about "improving" the table distribution by, say, eliminating linear tables:

- First of all, such tests would be an expensive waste of time; we will never find such a table at the 8-bit level.
- Next, linearity is hardly the only possible weakness in tables. It seems to me that damaging weakness would be far more likely to be something we do *not* know, and thus cannot check as part of the design process, than any weakness we might eliminate dynamically.
- Such a proposal seems eerily similar to the fatal weakness in the Enigma stream cipher, where a plaintext letter could not be enciphered into that same letter. Clearly, enciphering a letter into itself

provides no hiding at all, and if that were by chance to occur across an entire message, the opponents could simply read the ciphertext. But that sort of "exposure" it is just what we expect from random chance: if ciphertext can be any random value, sometimes the ciphertext will be the original plaintext. (Clearly, such "exposure" is less likely in block ciphers, where the enciphering alphabet is much larger.) The Enigma design prevented random "exposure," but doing that created a new exploitable structure which was in fact far, far weaker because it could be distinguished statistically.

In cryptography, anytime someone proposes to "correct" or "improve" a perfectly even distribution, everyone else's ears should perk way up.

We should remind ourselves that different cipher architectures use tables in different ways and thus have different requirements necessarily reported by different measures. Abstractly, there would seem to be no limit to the number of measures which might be used. And it seems unlikely that any table (or any large-enough keyable set of tables) can be simultaneously optimized for all known measures, let alone those measures of the future we do not now know.

While random tables cannot be expected to have *optimum* strength against any particular attack, they *can* be expected to have *average* strength against even unknown attacks. Where there is no systematic design, there is no systematic structure to be exploited as weakness. And when S-boxes are chosen at random, everyone can be sure that there is no S-box trap door. The point is to design keyed-table ciphers to accept and deal with the natural variations found in random table selection. Also see:

- "S-Box Design: A Literature Survey," locally, or @: http://www.ciphersbyritter.com/RES/SBOXDESN.HTM

Scalable

A cipher design which can produce both large real ciphers and tiny experimental versions from the exact same construction rules. Scalability is about more than just variable size: Scalability is about establishing a uniform structural identity which is size-independent, so that we achieve a strong cipher near the top, and a tiny but accurate model that we can investigate near the bottom.

While full-size ciphers can never be exhaustively tested, tiny cipher models *can* be approached experimentally, and any flaws in them probably will be present in the full-scale versions we propose to use. Just as mathematics works the same for numbers large or small, a backdoor cipher built from fixed construction rules must have the same sort of backdoor, whether built large or small.

For block ciphers, the real block size must be at least 128 bits, and the experimental block size probably should be between 8 and 16 bits. Such tiny ciphers can be directly compared to keyed substitution tables of the same size, which are the ideal theoretical model of a block cipher.

Potentially, scalability does far more than just *simplify* testing: Scalability is an enabling technology that supports experimental analysis which is otherwise *impossible.*

Scalar

In mathematics, a value of quantity only. As opposed to: vector.

Schematic Diagram

A graphic description of the interconnections between the components forming an overall system. Typically, different shapes or named boxes represent different kinds of component, and lines represent connections between components. In electronics, lines typically represent wires, or buses of multiple wires.

A schematic diagram is a visual depiction of a physical system. The schematic may then be used to understand, build or modify the real system, or to model the system in different ways to appropriate levels of precision. A schematic diagram is the single most common description of electronic systems for construction, debug and upgrade.

Science

1. Knowledge of reality.
2. The process of searching for structure in known facts, forming general laws and reaching rational conclusions. As opposed to mere authority.
3. The process of building numerical models which "predict" reality. (See scientific method.)
4. A system of study based on experimental evidence, systematic experimentation, hypothesis and rational conclusion.

Also see scientific method, scientific model, scientific paper, and scientific publication.

Scientific "Law"

Science is concerned with models of reality and experiments showing the models to be consistent with reality, or not. Science does not provide proof that models are "correct." That is why existing models are occasionally modified or even replaced by new models more consistent with reality.

Surely, if models can change, we cannot depend upon a specific model as a basis for absolute certainty. We can prove that the equations work, of course, but that is far different than proving results about reality. Science generally does not develop absolute "laws." Science generally provides no basis for the absolute certainty a cipher user expects.

Scientific Cryptography

As a "science," cryptography has developed both "bottom up," from real, practical implementations, and "top down," from mathematical theories. Unfortunately, those theories depend upon certain definitions and assumptions, and it is not always possible to guarantee those assumptions in practice (see proof). The existence of two different bodies of work has produced a "disconnect" in the general development. In both cases, cryptography is an arbitrary design, but the "bottom up" case is at least built on machines which can be realized.

One commonly ignored possibility is that implemented systems can be measured. But that insight is most useful when the systems are scalable, which unfortunately is not a feature of most designs.

Scientific Method

The basic function of science is to explain why things happen as they do. A comforting response is to develop quantitative models which predict observable outcomes. A satisfactory model implies a sense of understanding useful for explaining the past and for developing future devices which use the modeled properties. An important part of science is the development of a taxonomy to describe how different devices or principles fit into the general model.

A very basic approach to science is to simply:

1. collect facts (from library research, observation and experimentation),
2. reason from facts to new conclusions,
3. verify conclusions (by experimentation and observation).

Reality is known by raw observations which are presumed facts. A scientific theory must use known facts to predict outcomes which might be shown false by observation (if the theory is wrong). A better theory explains more facts, or does so more precisely. Models are often generalizations which attempt to explain only a small part of reality, and thus work only when properly understood and applied within assigned limits.

The grade-school description of the scientific method has five steps:

1. Observe some facts. Possibly set out to answer some specific question

2. Find an old scientific model or somehow create a new one which uses those facts and implies measurable or at least observable results. Creating a model is an inductive step, which typically involves both analysis and synthesis. The model is the hypothesis or conjecture or tentative theory. It is also reasonable to compare the results from multiple models.
3. Use the model to make some predictions.
4. **Conduct experiments** to see if the predictions occur. (However, some models are *generalizations* or even known *approximations* of reality, and measurements are always inaccurate to some extent. Statistics can provide a way to see beyond measurement error into the hidden reality.)
5. If the predictions do not occur, improve the model. (An experiment is not a "failure" simply because it shows an unexpected outcome; in fact such experiments may be the most important kind. A flat contradiction shows the model is wrong. An unpredicted result may show how to improve the model, and model improvement is scientific progress and "success.")

If we have multiple models, it is convenient to test (and use) the simplest one first. See: Occam's razor.

Scientific Method and Math

Unlike a physical science, mathematics is not locked to reality. Mathematical development differs from scientific development in that one is expected to prove or disprove a conjecture or hypothesis with straight deductive reasoning. However, previously unknown mathematical relationships can be exposed and demonstrated experimentally, which thus becomes the inductive step for future proof.

A mathematical axiomatic approach to science is actually fairly unusual: Most science is based on generalizations or rules of thumb, and inaccuracies "at the edges" are often expected and handled. In contrast, a strict deductive approach depends heavily on a correct choice of original assumptions or axioms, and often does not behave the way we expect from other sciences.

For example, in cryptography it is very common to define some interesting cipher property in strict mathematical terms (e.g., linear, or invertible). Accordingly, if it can be shown that even one case exists where the terms are not met, the mathematical definition is not achieved. Mathematical cryptography thus differs vastly from reality-based science which often treats properties as generalizations. In contrast, cryptanalysis operates much more like conventional science: Typically, if a property is available *in general*, a cryptanalysis technique based on that property can succeed. Often, more important than a strict mathematical definition are *measures* which tell us how much of some property is present (e.g., Boolean function nonlinearity). So, in the definition of absolutes, as well as in other more-subtle ways, it is possible for even a correct mathematical proof to deceive and lead an argument away from insight.

Valid Scientific Questions and Cryptography

A valid scientific question must be "falsifiable" by logical argument or experiment (see hypothesis). But not all questions can be investigated scientifically, and cryptographic strength may be just such a question. One problem is that the strength we care about in practice is not absolute but is instead contextual: Practical cryptographic strength depends upon the intelligence, experience, knowledge and equipment of our opponents. But because our opponents work in secret, their context is simply unknown. Practical cryptographic strength simply appears not to be a valid scientific question.

Even if we could find an absolute mathematical proof that a given cipher was secure, that would still not be enough: Consider the man-in-the-middle attack on public key ciphers, an attack in which no cipher is broken, yet which still exposes all information protected by cipher.

A field which cannot make verifiable predictions is not a science. The essence of cryptography is to protect our information from our opponents, but we have no way to know when we are successful, and so cannot verifiably predict success. Nor do we have any scientific grounds to go beyond mere naive belief that our current ciphers are successful in protecting our secrets.

Also see: scientific publication, quality management, and extraordinary claims.

Scientific Model

1. A quantitative description of reality that can explain or predict observations, based on one or more parameters. Also see hypothesis, null hypothesis and scientific method.
2. A useful structure or relationship between ideas. Multiple models can relate exactly the same ideas in different ways, and each model may be most useful for particular insights.

In science, the general character of most models is a mathematical relationship, but models only *approximate* reality, and are not reality itself. Indeed, models often are specifically designed to ignore details which *apparently* are not directly on topic. Models also tend to idealize and simplify relationships that may be complex in physical reality. Such simplification supports the development of relatively simple equations which describe a simplified and idealized reality.

To be able to reliably draw correct conclusions from a mathematical model, that model must have been constructed to include *all* significant aspects of the selected area of reality. Unfortunately, that is something we generally do not know until after we realize the model is flawed. Assuming a model is being used correctly, if there is a conflict between a model and reality, the model is simply *wrong*, no matter how mathematically consistent or proven it may be. Statistics can be very helpful in validating a theoretical model to practice.

A good model of reality often is a generalization, and thus has limits of applicability. Moreover, recognizing and accurately measuring the quantities predicted from such a generalization may take considerable experience. Interpreting what a scientific model means, where and how it does or does not apply, and how it can be used to predict reality, is one of the major skills of science. It is easy to misinterpret what a model really means. Part of scientific training is to understand that models have limits, and to learn how to deal with models when simplifying assumptions break down.

Often, the importance of a model is not in the predictions made, but in the insight the model brings by linking different areas of understanding. However, sometimes, multiple distinct models can explain exactly the same facts from different points of view. It may not be important to know which model is "correct," as long as each model predicts useful results. It is at least conceivable that known facts will never distinguish between models, so there may not *be* one "correct" answer.

One of the better examples of a modern scientific model is semiconductor in electronics:

1. At the most basic level, the model is that of atoms arranged in the form of a crystal, which is perhaps the simplest stable and easy-to-analyze form of matter.
2. At a more precise level, the crystal is made of a semiconductor, which says something about the availability of free electrons for conducting electricity.
3. So far we envision the "body" material of unlimited extent in all directions, but of course the inside of a crystal is not the same as the edge of a crystal, since crystalline bonds do not exist on the outside. This is another level of precision.
4. At yet another level, we can talk about "doping," or adding tiny amounts of specific other atoms to create more free electrons or holes *in particular areas* inside the crystal. Only now can we start to predict semiconductor action.

In cryptography, one of the best examples of a scientific model is that of keyed Simple Substitution, which mathematics likes to call a bijective mapping. Although Simple Substitution with a small alphabet is very weak, Simple Substitution with a *huge* alphabet is not so easily attacked and broken. But we cannot store or build a huge substitution table, so we have to innovate systems which emulate such a table. These emulated huge keyed Simple Substitutions are what we call conventional block ciphers, like DES and AES. And the keying part of this can be studied by considering the table contents to be a key-selected or key-constructed permutation. But that part of the model does not properly represent reality, because in reality, only a tiny, almost infinitesimal fraction of all possible permutations can be key-selected. (Also see block cipher and stream cipher models.) At another level cryptographers may try to create a threat model to capture the interaction between a cipher design and attacking opponents, but such models are very limited.

Eventually those in the field come up with equations which each relate to some important part of design by which useful devices can be made. For any particular part of the design, most equations will be ignored, and many parameters in the applicable equation also may be ignored. Knowing what one can afford to ignore is an important part of reducing complexity to manageable levels.

Scientific Paper

The publication of a scientific advance in sufficient detail to be replicated by others. Modern scientific papers generally follow a fairly fixed form, but simply following that form does not make a paper scientific. The point is to have something new, to clearly reveal what has been done, to "prove" or "show" that it is true, and teach the new things the reader must know to understand it.

Because science covers a wide range of activity and insight, it is difficult to find simple rules of thumb for writing that are appropriate in every case. Here an attempt is made to describe the essence of what is required, and one process for getting there:

The Story is the single most important aspect of any written article. To build a scientific paper the authors first need to know and be able to expose their story in a few clear sentences:

- Start with a sentence or two about what the author believes has been achieved and why that is not obvious.
- In another sentence or two, argue how, starting with previously known work, new experimental evidence or reasoning leads to the new or more refined results.
- Then present the experimental plan about what evidence is needed to establish proof and truth in this case, what relations need to be shown, and how those will be demonstrated or measured.
- End with a sentence or two of conclusions which are the new or improved scientific models.
- When the story is expressed well in a single short paragraph of a few clear sentences it becomes a basis for building the paper.

Teaching is the next most important aspect of a scientific paper:

- Teaching describes in clear language how the new work gets to new conclusions.
- Teaching provides any new techniques or new reasoning needed.
- Teaching provides a convincing proof or demonstration that each conclusion is true.

Classic scientific articles often are organized with the acronym IMRAD:

- Introduction: What issue was addressed?
- Methods: How was the issue exposed?
- Results: What was found?
- Discussion: What do the results mean?

The Methods section presents the experimental or development plan, along with any necessary tools. It might describe the various different results that could occur and what they each would imply. To confirm some effect it is necessary, in most cases, to find results that would not otherwise occur.

The Results section typically presents summarized or properly representative selected data. Any long or detailed experimental data should be placed in an Appendix.

Ideally, the Discussion section presents testable scientific models, with the argument and evidence that each is true, and compares each to other published work and reasoning. Evidence not supporting the conclusions must be exposed and argued.

Beyond the IMRAD science content are various support sections or subheadings:

- Title: The fewest words that describe specifically what has been done.
- Authors: The author who had the original idea or did the most important work is "senior" and listed first.
- Date: When publication occurred.
- Abstract: A one-paragraph general and non-detailed summary.
- Background (if needed beyond the Introduction): What previous work does this work build on?
- Comments (if needed beyond the Discussion): A place for speculation, but probably not a good idea.
- Acknowledgments: Recognize any significant technical or financial help received.
- References:
  - To refer the reader to published results, or
  - To provide sources so readers can learn already-known techniques needed to understand the paper.
- Appendix: Detailed experimental data.
- Biographical Sketch: When appropriate.

In scientific writing, it is important to use the proper tense:

- When quoting or discussing published work, use present tense.
- When describing this new work or the results, use past tense.

Also see Scientific Publication.

-- adapted and extended from: Day, Robert. 1988. *How to Write and Publish a Scientific Paper.* Oryx Press, New York.

Scientific Publication

Originally, a well-prepared report of new research, published in a primary journal, thus representing an archived and permanent advance in science. Also called "primary publication" or "valid publication." As distinguished from review papers, conference reports and meeting abstracts.

In particular:

- The first publication of original research results,
- With sufficient experimental detail to allow readers to repeat the experiments and test the conclusions,
- In a peer-reviewed journal readily available to the scientific community.

All intellectual work is always required to "give credit where credit is due": using someone else's ideas without acknowledgment is theft (typically plagiarism). Those listed as authors should include only those who actually have done the work; all others can be recognized in footnotes. First authorship should go to whoever made the primary contribution; conflict for credit between multiple authors can be addressed by describing the contribution of each author in a footnote.

Even though a published scientific paper presumably has surmounted peer review, research details are required specifically to support questioning both the results and the conclusions. Mere publication does not represent some sort of approval by "the scientific community." The scientific method does not require scientists to believe papers which seem to them to be either experimentally or logically flawed.

Classically, conference reports and meeting abstracts are preliminary and partial works and so do not normally qualify as scientific papers. Accordingly, they do not prevent later publication in a primary journal.

Classically, a review paper does not provide new results, and so is also not considered a scientific paper. Oddly, though, a review paper can be scientifically important, presenting "new syntheses, new ideas and theories, and even paradigms."

Also see Scientific Paper.

-- adapted from: Day, Robert. 1988. *How to Write and Publish a Scientific Paper.* Oryx Press, New York.

Scrambler

Some simple, fast process used to randomize data, often an LFSR or digital filter design. Frequently used in modems to reduce the probability of getting long runs of 1's or 0's. Not intended to produce security.

Secrecy

One of the objectives of cryptography: Keeping private information private. Also see: trust.

In a secret key cipher, secrecy implies the use of a strong cipher. Secrecy in communication requires the secure distribution of secret keys to both ends (this is the key distribution problem).

In a public key cipher, the ability to expose keys apparently solves the key distribution problem. But communications secrecy requires that public keys be authenticated (*certified*) as belonging to their supposed owner. This must occur to cryptographic levels of assurance, because failure leads to immediate vulnerability under a man-in-the-middle attack. The possibility of this sort of attack is very disturbing, because it needs little computation, and does not involve breaking any cipher, which makes all discussion of cipher strength simply irrelevant.

Secret Code

A coding in which the correspondence between symbol and code value is kept secret.

Secret Key Cipher

Also called a symmetric cipher or conventional cipher. A cipher in which the exact same key is used to encipher a message, and then decipher the resulting ciphertext. As opposed to a public key cipher. In practice, most so-called public key ciphers are hybrid cipher systems in which the public key component just transfers a key, and then a secret key cipher takes that key and actually protects the data.

Secret keys have a very strong analogy to the metal keys we all use and understand. For example, we do not need four different locks with four different key shapes so that four people can share the same house. In most cases, all those who are allowed access to some resource can and should share the same lock. Of course, the locks may be changed after somebody moves; similarly, key management is a central part of any cipher system, including public key ciphers. (When private keys may have been exposed or just used for a long time, it is important to change those keys as well.)

Many uses of cryptography involve securing files for storage, not transmission, where there is no need to transport a secret key, and no desire for each file clerk to have a different key. The idea that every person needs separate cryptographic access to every resource or even every person is just not realistic.

Security

Protection of a vital quality (such as secrecy, or safety, or even wealth) from infringement, and the resulting relief from fear and anxiety. The ability to engage and defeat attempts to damage, weaken, or destroy a vital quality. Security, in the form of assuring the secrecy of information while in storage or transit, is the fundamental role of cryptography, and is described as strength. Information security is often broken into:

- confidentiality or secrecy,
- integrity and
- availability.

A secure cryptosystem physically or logically *prevents* unauthorized disclosure of its protected data. This is *independent* of whether the attacker is a government agent, a criminal, a private detective, some corporate security person, or a friend of an ex-lover. Real security does not care *who* the attacker is, or *what* their motive may be, but instead prevents the disclosure itself.

Absolute security seems unlikely. But it is possible to at least *imagine* absolute security *against particular threats*. Proven security may be possible with respect to specific, fully-defined cryptographic attacks. But even "proven" security can rarely, if ever, be assured in practice, and it seems unlikely that we can define all possible attacks. Also see hypothesis.

Limited security is the general case and is necessarily contextual, depending both on the capabilities of the opponent and unknown weaknesses in the defense structure. One possibility for improvement is "defense in depth," where multiple different defenses are layered, hopefully with different unknown weaknesses and some sort of penetration alarm. A penetration alarm is not possible in cryptography. But the simple use of multiple encryption can reduce the information available to the opponent, and thus oppose many of the more serious attacks which need such information.

A typical response in physical security seeks to guess the identity, capabilities and motives of the attackers, predict their future actions, and concentrate resources at those points (see threat model). However, attempting to predict the actions or limitations of a serious opponent will incur considerable risk of failure.

In a sense, cryptography is almost *too easy* to use: It is compelling to imagine that the simple purchase of a cipher program can correct an information security problem. But no mere computer program, by itself, can guarantee security. In fact, true information security almost requires a new and special user lifestyle. Sadly, the very best a cipher can do is to protect information *while the information is in encrypted form*. But documents generally are not "born encrypted"; moreover, they must be decrypted for people to read, and fallible people will read them. Secret documents might be printed out, filed, thrown away, talked about, and otherwise exposed; all of these possibilities must be addressed, mainly by training users to change their former actions. Only after these avenues (and many more) are closed can one hope that simply using a cipher will provide true security. (Also see trust.)

In the modern networking environment it is probably *impossible* to maintain cryptographic levels of security on any computer connected to The Net. No firewall of any sort can be guaranteed to anything like the same cryptographic levels of effort commonly discussed in cipher attacks. Accordingly, serious secrets must be restricted to computers which simply cannot connect to The Net.

Security Through Obscurity

A term of art which normally refers to designing a new cipher which is supposedly strong, then keeping the cipher secret so it "cannot be attacked." (This is a distinctly different concept than the mere use of secret or "obscure" keys, or dealing with secret or "obscure" messages.)

Secret Keys "Yes," Secret Ciphers "No"?

Clearly, "security through obscurity" prevents public review of a cipher design, and from that it is often implied that secret ciphers could have serious weaknesses. Unfortunately, the exact same implication holds for *any* cipher, even a standard cipher, even after the best possible review and cryptanalysis.

Abstractly, there would seem to be no more reason that a secret cipher would have a higher probability of exposure than a secret key. Of course, the more people who use the same cipher *or* key, the more likely it is that even an unlikely thing will happen. But if every connection used a *different* cipher, in the same way that every connection currently uses a different key, exposing a cipher would have both a probability and an effect similar to exposing a key.

If programmers had the ability to choose, modify, or completely re-design a cipher for their own use, we could expect many more cipher designs. As a guess, ten years after such a system became available we could have

10,000 distinct ciphers. But if those designs were secret, we might not even know about them. And secret designs would not inform other programmers about cipher modifications *they* could use.

Since the danger of exposure is related to the number of users using the same secret cipher, it is interesting to consider multiple encryption. By enciphering three times with different ciphers and keys, we get an overall "cipher stack" analogous to a big bad cipher. An interesting part of this is that the number of different cipher stacks would be the cube of the number of "atomic" (indivisible) ciphers. If we have 10,000 ciphers, we get 10**12 different cipher stacks. That seems sufficient to make any particular cipher stack rare, making cipher stack exposure relatively meaningless.

Perhaps with multiple encryption and enough atomic ciphers, the ciphers themselves need not be secret. Perhaps only the *selections* constituting a particular cipher stack would need to be secret. Exposing a cipher stack selection would be like exposing a key, which is to say, serious for that user until the key is changed, but not particularly damaging after that or for any other channel.

Transiently Unknown Ciphers

It does seem likely that even secret ciphers eventually could be "obtained" by an opponent. As a goal, that should not "inconvenience" users (see Kerckhoffs' second requirement). However, obtaining each secret cipher would cost the opponent some amount of effort, delay and cost, whereas standard ciphers are immediately available at no cost at all. By standardizing ciphers we make some things much easier for the opponent.

One can imagine cryptosystems which dynamically select from among an ever-increasing set of ciphers. The ciphers might be only transiently unknown. But each of those ciphers would have to be obtained and analyzed by the opponent before it could be attacked. That would force the opponent into new massive costs for much smaller benefit, since each break would expose only a small part of all protected information.

The Consequences of Mistaken Belief

The belief that all security rests in the keys apparently allowed U.S. naval secrets to be exposed for seventeen years, as a result of the Walker spy ring. From October 1967 to mid-1985, Navy Warrant Officer John Walker delivered crypto information, including KW-7 "keylist cards," to the Soviet Union. That meant Russia had the crypto keys, but not the machine. Then it appears that Russia arranged for North Korea to capture the USS Pueblo, which had KW-7 cryptomachines on board. Apparently, KW-7's continued to be used by the US Navy for some time thereafter because exposure of the machine without the keys was not thought to be a significant risk. That belief was false.

Evidently, having a secret cipher machine design **can** contribute to security, which means that standard ciphers are more of a risk than the conventional cryptographic wisdom seems to believe. When every cipher machine is accounted for, and designs really are closely held, even having the crypto keys may not be much help (see cryptanalysis). This argues for having cipher machines which are physical hardware and so *can* be issued and accounted for. Another possibility would be to not treat keys as a single entity, and to never allow any single individual to possess all parts of a key. Even that would not prevent anyone from simply stealing plaintext, however.

Seed

The value placed into the state of an RNG. The start of a particular RNG sequence.

Self Inverse

In mathematics, an involution.

In cryptography, typically a cipher in which the same cipher and key will decipher what it has previously enciphered. The self inverse property is typically related to the use of an XOR or other additive combiner in an stream cipher. It can be dangrous to use a self inverse cipher in multiple encryption, unless the key is known to be different each time.

Self-Synchronizing Stream Cipher

(Asynchronous Stream Cipher). A stream cipher which automatically re-synchronizes if the ciphertext stream has either position errors (either loses a byte or has a byte inserted), or value errors. As opposed to a synchronous stream cipher.

The ability to recover after ciphertext error would seem to imply that the cipher produces the confusion sequence from a limited amount of past ciphertext or plaintext, and so is an autokey stream cipher. In particular, a non-data-diffusing autokey cipher which uses a fixed amount of prior ciphertext (or initial state) to produce each confusion element.

Semiconductor

A material which is between conductor and insulator with respect to ease of electron flow. The obvious examples are silicon and germanium.

As a rule of thumb, a cubic centimeter (cc) of a solid has about 10^{24} or 1E24 atoms. In a semiconductor like germanium, about one atom in ten billion (10^{10} or 1E10) has a broken bond which creates a free electron, thus producing about 10^{14} (1E14) free electrons per cc. The relatively modest number of free electrons in a semiconductor (one billionth as many as a metallic conductor) has a sizable resistance to current flow of from 10^{-3} (1E-3) to 10^{5} (1E5) ohms across a centimeter cube.

Silicon is an atom with 14 electrons, 4 of which are in the outer or *valence* shell. It is a solid at room temperature, and as it hardens, the atoms form a *crystalline* structure. The comparative simplicity of the crystaline structure supports simple models and accurate predictions of electronic effects. Within a natural silicon crystal, most electrons are locked in place, but some are freed by thermal action and become relatively free to move within the crystal, which creates the semiconductor level of conduction.

To improve conduction, single crystal silicon can be *doped* with a tiny amount of arsenic, an atom with five valence electrons, thus forming a negative or N-type semiconductor crystal with many free electrons. Similarly, doping with aluminum or boron, atoms with three valence electrons, can form a P-type semiconductor crystal with an excess of *holes*. Clever doping can produce N and P layers *in the same crystal*, and then current can flow, but normally only from P to N; this is a diode PN junction. Also see transistor.

Semigroup

In abstract algebra, a nonempty set *S*, and a closed dyadic operation with associativity only. Whatever the operation is, we may choose to call it "multiplication" and denote it with * as usual. Closure means that if elements (not necessarily numbers) *a, b* are in *S*, then *ab* (that is, *a*b*) is also in *S*.

In a semigroup consisting of set S and closed operation * :

1. **The operation * is associative:** (ab)c = a(bc)

A set with a closed operation which is associative and has an identity element is a monoid. A set with a closed operation which is associative, with an identity element and inverses is a group.

Series

In electronics, components connected such that the same current flows through all, while voltage divides across each according to their resistance or impedance. As opposed to parallel.

Session

A logical connection between two elements of a network or system. Typically established for a period of multiple interactions and then terminated.

Session Key

A key which lasts for the period of a work session. A message key used for multiple messages.

Set

A well-defined collection of distinguishable elements, usually, but not necessarily, numbers. Typically described by a rule so that any particular element can be decided to be either "in" or "out of" the set. Explicitly denoted in math notation with values inside braces, for example: {0,1} is a set of two values.

Set Up Time

In electronic digital logic, the amount of time a signal voltage must be present and stable *prior to* the occurrence of a clock for the signal to be guaranteed to be recognized by all devices over all allowed variations in device processing, power supply, signal level, temperature, etc. Also see: hold time.

Shannon

In cryptography and information theory, Claude E. Shannon (1916-2001). A greatly respected and hugely influential Bell Labs theorist responsible for:

- The mathematical model linking bandwidth and communications data rate.
- Information Theory.
- The measure of coding efficiency called Shannon entropy.
- Experiments to establish the Shannon entropy of English (also see letter frequencies).
- Unicity Distance.
- Equivocation.
- The Algebra of Secrecy Systems, (including Product Cipher or multiple encryption).
- Perfect Secrecy.
- Ideal Secrecy.
- The Work Characteristic as a measure of *practical* secrecy.
- Pure Cipher.

Shielding

Typically, a conductive envelope around electronic devices, to constrain incidental electromagnetic radiation.

Ideally, equipment would be fully enclosed in an electrically unbroken conducting surface. In practice, the conductive enclosure may be sheet metal or screening, with holes for shielded cables. In general, shielding with holes can be effective up to some frequency where the wavelength becomes comparable to hole size.

Shielding occurs not primarily from metal *per se,* but instead from the flow of electrical current in a conductor. When an electromagnetic wave passes through a conductive surface, it induces a current, and that current change creates a similar but *opposing* electromagnetic wave which nearly *cancels* the original. The metallic surface must conduct in all directions to properly neutralize waves at every location and from every direction. The result is an electromagnetic mirror which prevents most signal from escaping, but can greatly increase the signal intensity inside the shield. This increased energy often dissipates on internal conductors or steel surfaces, but if not will increase until the shielding becomes ineffective.

Stock computer enclosures often have huge unshielded openings which are hidden by a plastic cover. Openings should be covered with a conductive surface, such as:

- metal plates,
- unetched copper-clad,
- copper screening, or
- copper or aluminum foil.

It is important that good electrical contact occurs at all places around the hole, which can be attained by soldering, conductive foil tape, or possibly improved with conductive grease or epoxy. Note that forming good electrical surface connections can be difficult with aluminum, which naturally forms a thin but hard and non-conductive surface oxide.

When adding shielding to an existing chassis, it is important to actually monitor emission levels with a receiver before and after any change. While extreme success can be very difficult, significant improvements can be fairly easy. Without a radio, we can at least make sure that the shielding is tight (that it electrically conducts to all the surrounding metal), that it is complete (that copper, or copper screen or aluminum foil covers all large holes), and that external cables are also shielded.

Cable shielding extends the conductive envelope around signal wires and into the envelope surrounding the equipment the wire goes to. Any electromagnetic radiation from within a shield will tend to produce an opposing current in the shield conductor which will "cancel" the original radiation. But if a cable shield is *not* connected at *both* ends, no opposing current can flow, and no electromagnetic shielding will occur, despite the metallic "shield." It is thus necessary to assure that each external cable *has* a shield, and that the shield is *connected* to a conductive enclosure at *both* ends. (Some equipment may have an isolating capacitor between the shield and chassis ground to minimize ground loop effects when the equipment at each end of the cable connects to different AC sockets.)

When shielded cable is not available (or not possible, as in the case of AC power lines), it can be useful to place ferrite beads or rings around cables. In effect, the ferrite magnifies the inherent self-inductance of the wire. Nevertheless, the resulting inductance values are usually quite small, and depend upon working against some capacitance to ground to roll-off high-frequency pulses. The ferrite also magnifies the mutual inductance between wires in the same cable to promote a balanced and non-radiating signal flow. That depends upon having a fairly low impedance termination for the cable, so that the balancing current can flow easily.

Also see: TEMPEST and electromagnetic interference.

Shift Register

An array of storage elements in which the values in each element may be "shifted" into an adjacent element. (A new value is shifted into the "first" element, and the value in the "last" element is normally lost, or perhaps captured off-chip.) (See LFSR.)

```
   Right-Shifting Shift Register (SR)

               +----+  +----+         +----+
   Carry In -->| A0 |->| A1 |-> ... ->| An |--> Carry Out
               +----+  +----+         +----+
```

In digital hardware versions, elements are generally bits, and the stored values actually move from element to element in response to a clock. Analog hardware versions include the charge-coupled devices (CCD's) used in cameras, where the analog values from lines of sensors are sampled in parallel, then serialized and stepped off the chip to be digitized and processed.

In software versions, elements are often bytes or larger values, and the values may not actually move during stepping. Instead, the values may reside in a circular array, and one or more offsets into that array may step. In this way, even huge amounts of state can be "shifted" by changing a single index or pointer.

Shot Noise

The random analog electrical signal caused when current flows as independent electrons instead of a correlated stream. Shot noise is a statistical effect of event "clumping" when the events have an expected rate but independent times. Shot noise does not occur in conductors, where electrons are correlated like water in a hose.

Instead, shot noise requires some sort of independent emission of discrete events, such as electrons through a semiconductor junction, or water drops through a spray nozzle. Normal resistors do not produce significant amounts of shot noise. Surface mount resistors made from conductive particles and glass do generate resistor excess noise proportional to both resistance and current. In contrast to the wide range of electron speeds and directions in thermal noise, shot noise consists of individual pulses, all similar except for timing, which may overlap. Shot noise is a very weak noise source and a white noise.

> "Electricity is not a smooth fluid; it comes in little pellets, that is, electrons. The flow of electrons in a vacuum is accompanied by a noise of the same nature as the patter of rain on a roof." -- Pierce, J. R. 1956. Physical Sources of Noise. *Proceedings of the IRE.* 44:601-608.
>
> "Another important instance of shot noise arises in . . . the motion of carriers across a high-field transition region, e.g., at a metallic contact or at a *p-n* junction." ". . . the effect of each electron transit is effectively to introduce a current impulse . . . ." -- Burgess, R. E. 1955. Electronic fluctuations in semiconductors. *British Journal of Applied Physics.* 6:185-190.

Fundamentally, shot noise is a variation in current proportional to square-root current when electrons move independently (that is, not in a conductor). This variation is very small: a current of 1A through a power diode has a shot noise of just 57nA rms with a 10kHz measurement bandwidth. However, since a current of 1A represents a flow of about 6.24 x 10^{18} electrons per second, the resulting 57nA rms shot noise still represents about 3.56 x 10^{11} electrons per second, still much more of a flow than individual pulses. In a bipolar transistor, the main source of shot noise is current in the emitter-base and collector-base junctions.

Shot noise is often seen as a voltage developed across the dynamic resistance of a semiconductor junction, and that resistance is inversely proportional to current. The resulting shot noise voltage across a junction is inversely proportional to square-root current through the junction. But in practice, device variations are much larger than the expected calculated tendencies. See "Measuring Junction Noise" (locally, or @: http://www.ciphersbyritter.com/RADELECT/MEASNOIS/MEASNOIS.HTM) and "Junction Noise Measurements I" (locally, or @: http://www.ciphersbyritter.com/RADELECT/MEASNOIS/NOISMEA1.HTM).

The amount of shot noise current and shot junction voltage can be computed as:

```
   i[n] = SQRT( 2 q I B )
   e[n] = k T SQRT( 2 B / I q )

where:
   i[n] = shot noise current through junction (rms amps)
   e[n] = shot noise voltage across junction (rms volts)
   q = electron charge (1.602189 * 10**-19 coulombs)
   I = current flow (amps)
   B = measurement bandwidth (Hz)
   k = Boltzmann's constant (1.380662 * 10**-23 joule/deg K)
   T = temperature (deg. Kelvin)
```

Or, for quick approximations at room temperature [i[n] from Ryan, Al and Tim Scranton. 1984. D-C Amplifier Noise Revisited. *Analog Dialog.* 18(1): 3-11]:

```
   i[n] = 0.566 SQRT( I B )
      with E = IR, for junction dynamic R = 25.86E3 / I,
   e[n] = 0.0146 SQRT( B / I )

where:
   i[n] = shot noise current through junction (rms pA)
   e[n] = shot noise voltage across junction (rms uV)
   I = junction current (uA)
   B = measurement bandwidth (Hz)
```

Also see: avalanche multiplication, Zener breakdown and "Random Electrical Noise: A Literature Survey" (locally, or @: http://www.ciphersbyritter.com/RES/NOISE.HTM).

Shuffle

Generally, the concept of "mixing up" a set of objects, symbols or elements, as in shuffling cards. Mathematically, each possible arrangement of elements is a particular permutation. Also see transposition and double shuffling.

Within a computer environment, it is easy to shuffle an arbitrary number of symbols using a keyed random number generator, and the algorithm of Durstenfeld, named SHUFFLE, which is also described in Knuth II:

> Durstenfeld, R. 1964. Algorithm 235, Random Permutation, Procedure SHUFFLE. *Communications of the ACM.* 7: 420.
>
> Knuth, D. 1981. *The Art of Computer Programming,* Vol. 2, Seminumerical Algorithms. 2nd ed. 139. Reading, Mass: Addison-Wesley.

Also see

- my article "A Keyed Shuffling System for Block Cipher Cryptography," locally, or @: http://www.ciphersbyritter.com/KEYSHUF.HTM

Sieve of Eratosthenes

A way to find relatively small primes. Although small primes are less commonly useful in cryptography than large (say, 100+ digit) primes, they *can* at least help to validate implementations of the procedures used to find large primes.

Basically, the "Sieve of Eratosthenes" starts out with a table of counting numbers from 1 to some limit, all of which are potential primes, and the knowledge that 2 is a prime. Since 2 is a prime, no other prime can have 2 as a factor, so we run though the table discarding all multiples of 2. This is easy because we accept 2, then step 2 places, discard that value, step 2 places, discard that value, and so on to the end of the table, so almost no computation is required.

The next remaining number above 2 is 3, which we accept as prime, then run through the table crossing off all multiples of 3. Since 4 has been discarded, the next remaining is 5, which we accept as prime, then run through the table crossing off all multiples of 5, and so on. After we cross-off the last prime below the square-root of the highest value in the table, the table will contain only primes.

A similar process works with small polynomials, and small polynomial fields, to find irreducible polynomials.

Significance

In statistics, the probability of committing a Type I error, the rejection of a true null hypothesis. This is also the *size of the critical region*.

Suppose we have the distribution for some statistic based only on random data (that is, random sampling). A 5 percent *level of significance* means that we consider 5 percent of the distribution (either on one tail, or both tails, depending on the test) so extreme as to be "something unusual found." Only statistic values in this extreme "critical region" cause us to reject the null hypothesis and accept the alternative hypothesis. However, that also means the null hypothesis will be rejected 5 percent of the time by chance alone.

Simple Substitution

A type of substitution in which each possible symbol is given a unique replacement symbol.

Perhaps the original classical form of cipher, in which each plaintext character is enciphered as some different character. In essence, the order of the alphabet is scrambled or permuted, and the particular scrambled order (or the scrambling process which creates that particular order) is the cipher key. Normally we think of scrambling alphabetic letters, but any computer coding can be scrambled similarly.

Small, practical examples of simple substitution are easily realized in hardware or software. In software, we can have a table of values each of which can be indexed or selected by element number. In hardware, we can simply have addressable memory. Given an index value, we can select the element at the index location, and read or change the value of the selected element.

A substitution table will be initialized to contain exactly one occurrence of each possible symbol or character. This allows enciphering to be reversed and the ciphertext deciphered. For example, suppose we substitute a two-bit quantity, thus a value 0..3, in a particular table as follows:

```
   2  3  1  0.
```

The above substitution table takes an input value to an output value by selecting a particular element. For example, an input of 0 selects 2 for output, and an input of 2 selects 1. If this is our enciphering, we can decipher with an inverse table. Since 0 is enciphered as 2, 2 must be deciphered as 0, and since 2 is enciphered as 1, 1 must be deciphered as 2, with the whole table as follows:

```
   3  2  0  1.
```

Mathematically, a simple substitution is a mapping (from input to output) which is one-to-one (injective) and onto (surjective), and is therefore bijective and invertible.

Sine Wave

The graphic waveform of amplitude versus time which has the rolling "hill-and-valley" outline produced by the analog sine or cosine function. This is the waveform of ordinary AC power distribution, as opposed to the square waves of digital logic.

A sine wave shape by definition represents just a single frequency, while any other shape represents a combination of frequencies. The cosine function also produces the sine shape, but with a 90-degree phase difference. The sine and cosine pair of functions are orthogonal, and thus can be used as a basis to describe any continuous waveform. This is the basis of the Fourier transform.

Single Point of Failure

In risk analysis, any point in a system where a single component failure can interrupt system services or operation. Perhaps the major issue in "reliable system design," where the goal is to eliminate all such points, and add redundancy to points which cannot be eliminated.

In cryptography perhaps the most obvious single point of failure is the cipher. Because no cipher is proven secure in practice, it may fail, and if it does all security is lost. One alternative is to use multiple encryption, so if a particular cipher has been broken in secret, the other ciphers used must also be broken to expose the information. An advantageous extension is to change ciphers frequently, so if failure does occur, it can be passed by as soon as the cipher is changed (see Shannon's Algebra of Secrecy Systems).

Another single point of failure in cryptography is the key.

Smooth Number

An integer with no prime factors smaller than some specified number.

Snake Oil

Cryptography which may be ineffective. Unfortunately, in practice, the *possibility* of ineffectiveness is the situation for each and every cipher currently known.

The term "snake oil" presumably comes from an era of American history when drugs were unregulated: At that time it was possible for a travling salesman to make outrageous medical claims simply to sell a product (which could be ineffective or even dangerous), and then move on before the results became apparent.

Perhaps the main goal of cryptography is to keep information secret. And to keep information hidden, our ciphers need the strength to rebuff attacks by our opponents. Unfortunately, cryptographic strength in ciphers is, in practice, a uniquely unscientific situation, which makes wild claims and counterclaims the natural state of the cryptographic market. (Also see: Kerckhoffs' requirements.)

The Snake Oil FAQ by Matt Curtin cites various snake oil warning signs, but the situation is frequently less clear than one might like:

Originally there was some thought that ciphers using huge keys indicated a lack of understanding of keying, and, thus, cryptographic design. But the real situation is that a large keyspace prevents a brute force attack on the keys. So, once we have a "large enough ideal key" (for secret keys, 128 or perhaps 256 bits), more key does not buy more security (i.e., other attacks will be more effective). So, long keys may be *inefficient*, but the cryptographic problem is *short* keys, not long ones! In fact, if the keying source has some sort of unknown bias, having a larger-than-strictly-necessary keyspace may provide useful additional protection. I have used 992-bit message keys (32 x 31) since 1991 (see, for example, "The Cloak2 Design": locally, or @: http://www.ciphersbyritter.com/CLO2DESN.HTM), a size which I consider to be a more than acceptable overhead.

1. **"Trust Us, We Know What We're Doing."** Buyers are told to be wary of cryptography suppliers who will not describe their product, presumably the inner workings in complete technical detail.

   Unfortunately, technical details would not help many buyers, because few are both able and willing to perform a detailed cryptographic review of a complete cipher system. Indeed, one might well think that *not* having cryptographic expertise for such a review or not wanting to make the design investment would be the main motives for *buying* a product instead of *building* one.

   "Who" and "How much" to trust are central issues of cryptography and security that no mere cipher is going to solve. For example, one alternative would be to hire a very expensive consultant to perform a detailed cryptographic analysis, which would of course mean *trusting* the consultant.

   And if buyers and consultants hope to predict practical strength on the basis of cryptanalysis, they are necessarily forced to believe that academic cryptanalytic expertise at least equals the expertise of our opponents. But that cannot be true, since academic expertise is published and open, whereas our opponents have that *plus* secret expertise of their own.

2. **Technobabble.** Buyers are told to avoid suppliers who offer confusing or nonsense descriptions of operation, or who use newly-invented terms without proper explanation. Unfortunately, cryptography is complex, so even appropriate scientific descriptions may seem to be technobabble to most buyers. Even more unfortunately, some of the apparently scientific descriptions in the academic literature might also be described as technobabble.

3. **Secret Algorithms.** Buyers are told to avoid products which use secret algorithms. Alas, this is a double-edged sword: Certainly, new cipher designs do need review, although academics often fail to address new architectures. On the other hand, serious military ciphers are *not* made available for general academic review, which does not mean that we trust them any less. Indeed, there is no scientific basis for trusting *any* cipher in practice. (Also see: security through obscurity.)

4. **Revolutionary Breakthroughs.** Buyers are told to avoid products which claim new insights. I note that this attitude is not just *un*scientific, but also actively *anti*-scientific. Not only does it mean that inventors have little motive for research, and that buyers cannot capitalize on recent discoveries, it also implies that we can trust a cipher which has been used for a long time without a known break.

   But when a cipher has been used for a long time, it probably has protected a lot of valuable information, which thus provides the motive for serious investment in a serious secret attack which would not otherwise occur. And when a cipher has been used for a long time, there has been time to create and field such an attack (also in secret). This is arguably worse than the original problem, which at least potentially can be solved (as much as possible) simply by academics actually performing analysis on claimed breakthroughs.

5. **Experienced Security Experts, Rave Reviews, and Other Useless Certificates.** Buyers are told to beware of claims of cryptanalysis made without full details, such as who the experts are and what their qualifications may be. Surely buyers do need to check that clams made for a product are true.

   On the other hand, cryptanalysis is *not* an algorithm: instead, cryptanalytic approaches must be *applied* and generally *customized* for each particular cipher. Consequently, even a cryptanalyst with the best of reputations can make a mistake (in either direction) in a relatively quick analysis. Furthermore, even the best cryptanalyst is limited to what is known in the open literature, while the opponents are *not* similarly limited. Even long-used widely-analyzed ciphers are not necessarily strong in practice.

6. **Unbreakability.** Buyers are told to mistrust any claim to unbreakability. Certainly the history of cryptography is the history of failed ciphers and failed claims of strength. Lacking a general ability to measure cipher strength, we have no scientific basis for trusting *any* cipher in practice, no matter how intensively reviewed or how long used. We must *always* consider the possibility and consequences of cipher weakness.

7. **One-Time-Pads (OTP's).** Buyers are warned that OTP's will not solve their problems. (Also see stream cipher.) Since OTP's need as much running key as the data, and since that key material must be created, transported to both ends, then stored, all with absolute security, OTP's have a tremendous operational burden.

   Because the academic texts are not clear on the implications of OTP's, many suppliers see academic security proofs for the OTP as a way of finessing and avoiding the complexity of cryptographic design. However, those proofs require that the system have a *provably* theoretically-random source of bits, which is probably impossible (see really random). In general, there is no provably strong OTP in practice, just systems which may be strong or not, just like other ciphers. And OTP's require the continuous maintenance of new key material.

8. **Algorithm or product X is insecure.** Buyers are warned to avoid suppliers who criticize products from other suppliers without describing the problem, since many academic attacks are impractical and, thus, only confuse the issue in practice.

   Product competition is the buyer's friend. One of the reasons for developing any new product is that existing products have problems. There should be no shame in pointing out the advantages of a new development including problems others have not overcome.

9. **Recoverable Keys.** Buyers are warned to consider the implications of any cipher system which can recover lost keys. But key loss is a major issue for all cryptosystems: If a substantial amount of information (or even just one contract) is lost due to a lost key, that organization will have much less need for cryptography in the future. Attempts to address key loss are a legitimate product issue, especially for business use.

10. **Exportable from the USA.** Buyers are warned to avoid suppliers who are unaware of export rules, and that exportable products may be "crippled." This is less true now than it was at one time.

11. **"Military Grade."** Buyers are warned to disregard the term as "meaningless." Surely the expression must have meaning to somebody, but those people are not talking. So anyone who claims "military strength" probably has no idea what the military really uses or what their criteria for strength might be. Thus, anyone making such a claim would be displaying a serious credibility problem.

As is often the case in cryptography, some of the snake oil "warning signs" have serious problems when they are themselves analyzed. This appears to be a sort of profiling which just seems to be ineffective at making the desired distinction between useless and worthwhile products. But "snake oil" products are common and some research is necessary to avoid them.

Socrates

(470-399 B.C.E.) A decorated Greek warrior and philosopher, Socrates apparently wandered around Athens having conversations with, and learning from, almost anyone. Some of the best of those conversations were recorded by his friends, Plato and Xenophon. Plato went on to found a school called the *Academy*, from which we apparently take the term academic.

Socrates somehow developed a skill in conversation beyond that of anyone known at that time, and, perhaps, our time as well. This skill is only described indirectly, by the recorded conversations which survive. Apparently, Socrates could lead the other person to see their own errors in thinking, *without* causing too much offense. It should be noted, however, that Socrates was eventually condemned to death, and drank hemlock rather than leave Athens.

Socrates specifically disclaimed being a teacher or an author or even knowing very much. Indeed, he implies that his great advantage was to know how *little* he knew, and then seek answers from experts (or those most concerned), based on dialectic and logic. Socrates' Method is to ask a question, get a response, and then see if that response works in various other areas. If not, dialog used to get at the root of the problem, leading to deeper insight on the question. The goal is for everybody to end up knowing things more deeply than when they started the conversation.

Socratic Method

In argumentation, a formalized process of interactive or dialectic reasoning for teaching or developing insight. Apparently originating from the Greek philosopher Socrates. Basically it goes like this:

1. **Question:** The question is stated, and terms used in the question defined by example and adduction.
2. **Reflex Answer:** The conventional wisdom is proposed as a hypothesis.
3. **Analysis:** In playful or ironic dialog, a *definition* hypothesis may be shown to apply to various things which are obviously something else, which are thus valid contradictions. A *solution* hypothesis may be shown to fail under reasonable conditions. Or the logical negation of a hypothesis may be shown to fail, thus perhaps *supporting* the original. Hypotheses are typically shown to be insufficient, contradicted, or even self-contradictory.
4. **Synthesis:** The hypotheses is modified to avoid contradiction, typically demonstrating a deeper understanding of the issue.

The sequence of: answer, analysis, and synthesis is essentially the same process by which system design is conducted. It is also how mathematical proofs (see: Method of Proof and Refutations) and scientific models are improved. And it would be the way cryptographic ciphers could similarly be improved, if those ciphers were not simply discarded the very minute weakness was found. To fit with scientific development, the main goal of cryptanalysis ought to be cipher *improvement*, as opposed to cipher *destruction*.

Fortunately, the Socratic Method does not require great intellect, but mainly the simple ability to reason and to recognize when some assertion does not really apply as well as was commonly thought. Such exposure occurs almost automatically when the question: "Why?" is asked and contradictions found in the response, based on analogy or other reasoning.

The process is also effective at conveying complex arguments, with the issue sketched in the first stage, various approaches proposed in the second, their problems exposed in the third stage, ending with an improved solution. Thus, it can be a way to help others understand your position.

In argumentation, it is typical for a contradiction to produce conflict over the meaning of various terms. In language, terms often have multiple meanings, and sloppy or malicious argumentation can choose different meanings at different times to respond to particular attacks (this is the equivocation fallacy). However, the situation is not hopeless if the common goal is to extend knowledge and reach a conclusion: Terms can be clarified by multiple examples of use, so that, within the argument, a single common meaning can be adduced and assigned. (Also see mere semantics.)

The overall goal of the Socratic Method is to improve understanding, to teach or learn, and has nothing to do with "winning." In scientific argumentation, *everybody* "wins" when ignorance is exposed. The deliberate or repeated use of purely rhetorical techniques (such as propaganda or fallacies) to "win" a scientific argument is not merely misguided, but wastes the time of every listener, and reveals a lack of scientific integrity in the speaker.

Software

The description of a logic machine. The original textual composition is called source code, the file of compiled opcode values is called object code, and the final linked result is pure "machine code" or machine language Note that, by itself, software does not and can not function; but instead relies upon hardware for all functionality. When "software" is running, there is no software there: there is only hardware memory, with hardware bits which can be sensed and stored, hardware counters and registers, and hardware digital logic to make decisions. See: computer, system, system design, Software Engineering, Structured Programming and debug.

Software Engineering

The engineering process of designing and developing a logic-machine to perform a desired task. The tools, methods and disciplines which first create an informal, conceptual model of a desired system and then transform that into a formal, executable model which *is* the desired system. The actual programming is generally seen as a minor part of the overall task.

One of the more interesting recent books on Software Engineering is:

Glass, R. 2003. *Facts and Fallacies of Software Engineering.* Addison-Wesley.

The book collects a sequence of software beliefs, and then presents the known facts and rational argument that typically contradict such belief. Perhaps if our beliefs become more realistic we may improve both our performance and satisfaction.

Estimation and Success

One of the many things wrong with current software development practices is that project cost estimation occurs at the *start* of the project. Granted, that is the traditional way to set up a contract or manage a project investment. But before the start of software development, *any* estimate is particularly *rough*. At that time, almost nothing will be known about the legion of difficulties which may occur. So it should be no surprise that development projects are often seen as "late" and/or "over budget," when compared to the original rough estimate.

Being late or over-budget does not mean the project will not complete, nor that the result will not be successful or worthwhile. It is instead the existence of the original uninformed estimate which leads to recriminations as in: "You did not do what you said you would."

Unless one is re-doing something very straightforward or similar to what one has done before, software development is largely *research* and should be budgeted as such. Nobody imagines that new concepts will appear on some arbitrary schedule. But one can expect that effort not be wasted and progress made. And, in software (as opposed to research), one can make increasingly better estimates as to when success will be achieved. (Also see risk management.)

Size

It is much easier to build small software systems than big ones. Individual programmers usually are much more effective on small projects than large ones, presumably because much less communication and coordination are required. However, our society sees many large construction projects, each of which is somehow decomposed into jobs one man can handle, either working with others, or independently, or sequentially. Software engineering normally deals with large, complex projects, and the engineering process of controlling the complexity of the work.

The Beehive Approach

Engineering control typically involves an in-itself complex system of incremental development documentation including overall engineering designs, detailed designs for each area, document control, change requests, management approvals, etc., etc. Ideally, an emphasis is placed on early design validation so that problems are exposed as early as possible, when they cost much less to repair.

Similarly, the code itself is kept in a central repository. Programmers may be easily drafted into working on the same code chunk as someone else, if that is behind schedule. That implies a need for a code change-control system to prevent concurrency problems, so that a chunk of code is "checked out," worked on, and then replaced.

The Tree Approach

My conception of Software Engineering is different: I work hard to leverage the ease of small-project development. The idea is to decompose complex projects into small, independent modules which can be programmed separately, basically by no more than two programmers each. Each chunk of code has an "owner," and only the owner changes or otherwise writes that code. That alone prevents concurrency problems and the need for code change-control software. (Version-control is still necessary as a repository for each production version of each module. But it is a task unto itself, and not something in which everyone must participate.)

If somebody needs to change something in someone else's chunk, they do not just "check out" the code and change it. Instead, they have to get *the owner* to change it, which forces at least a modicum of planning, all of which happens without committees, agendas, meetings, approvals, or other typically massive engineering overhead. Responsibility is clarified instead of being diluted by everybody working on everybody else's stuff. Good programmers thus stand out, and poor programmers either develop or move on.

Good project decomposition requires particular effort during the system design phase to identify modules, and if necessary, to even redesign the system concept to better support modules. More front-end effort is required as it is necessary for the system designers to specify in detail the data structures and assumptions each module will use. (Although it is unclear that all projects can be decomposed in this way, we sure can do much better at it than we have in the past.)

Project decomposition can occur in many different ways, many of which are surprisingly unhelpful. Just decomposing into modules is not helpful unless the modules do appropriate things inside an appropriate interface. In my experience, the most important rule is that each module (and each procedure or subroutine) be individually testable. "Everything" a module needs must be sent through parameters, with results returned as a function return value, and/or parameters. (An exception is that of error "exceptions," since they can be "caught" by a calling routine.) The goal is to encapsulate and hide the lower-level complexity which is the code inside each module. In addition, we actually do test all routines as part of development (see system design).

Note that design and re-design may occur throughout the software development process. It is generally a mistake to imagine that some group can produce a design so finished that coders can just "fill in the blocks." One cannot know a design is correct until *after* implementation, since it is the implementation which checks the design. Consequently, software design is likely to be an iterative process and subject to feedback from implementation about unexpectedly costly or ineffective or just plain wrong design decisions. (Also see debug.)

Modeling

Software engineering is fundamentally a modeling activity. Models are used to predict and explain the system, and, eventually, to implement the system. In the first stage, a conceptual model or formal specification is created corresponding to the future real-world system. Of course, before one can create this model, it is first necessary to perform a requirements analysis and understand the problem. Only then can one come up with an apparent solution. All this is much easier if you have done something similar several times before. First-time development is essentially research and can involve substantial trial and error.

The specification communicates to the customer what is to be built, and describes to the developers specifically what needs to be done and so implies how much work will be involved. However, the specification necessarily will change somewhat over time as hidden needs and interactions are exposed. The specification provides the central concept around which the many developers in multiple groups can coordinate their activities.

Ideally, the specification is not simply re-interpreted but is instead formally modeled and then semi-formally *transformed* and expanded until it describes software which can be built. Properly done, this should force the resulting design to be correct (to the formal specification) by design. In that case, everything depends on how well the original vision modeled the desired system.

Programming

Programming is typically considered the least part of a software engineering project, which is probably wrong. The ultimate formal model of the system is the executable program. Ultimately, design modeling produces the program, and designers should be very well acquainted with the tools needed to reduce complexity and test thoroughly.

However, although most software engineers do program well, I have personally encountered those who were trying to develop software without any background at all in modern programming practices. Just using a modern language development system or using objects is not going to result in good programs if one does not know when and why to use the various language features. Programming style is widely discussed in books and manuals, and I present some of that here, in Structured Programming. Also see cipher system and system design.

Software Patent

There seems to be no one fixed definition of the term "software patent." Certainly, one would expect that it has something to do with patents on software. But there is no PTO certificate or grant which is called a "software patent." Nor is software a new "statutory class" for allowing patents. Instead, ordinary patents are just seen to cover software where appropriate. (Also see patenting software.)

Software patents are an issue of heated debate, often moving well beyond software to contest the worth of patenting itself. Sometimes the exact same argument will apply to software patents, patents in general, intellectual property, and private property. When that happens, the basic issue may not be software patents *per se* but instead the ideas of property and ownership, for which software patents can be a convenient whipping-boy. (Also see my article "The Politics of Software Patents" (locally, or @: http://www.ciphersbyritter.com/ARTS/POLIPAT4.HTM.))

Recall that a published patent *exposes* an inventive idea to society, and does not restrict reading about it or writing about it. Instead, a patent restricts the *application* or *use* of the invention in every implementation. The distinction between covering every possible implementation and covering just a single fixed form is the distinction between patent and copyright. Software may be seen as a way of implementing logic machines, some of which may be covered by patent.

It is often argued that patent law prevents granting patents on "mere algorithms," and that software is "nothing but" algorithms, implying that is something new. In contrast, I contend that ordinary physical machines *also* implement algorithms, and always have. Furthermore, the diagrams and text instructions which have always described how to construct a machine to perform certain actions in sequence are just as algorithmic as software text. Machines *always* implement and perform algorithms. Virtually all patents *always* have issued to cover "mere algorithms" in their implementation and use.

A "mere algorithm" that has no practical use has no business getting a patent, and is restricted from doing so. And it would be quite a stretch to imagine a patent covering mental processes and their results (although one might imagine cases where that would apply). But it is nowhere near as difficult to see an electronic computer as a steam engine under new technology, a real machine in every sense, with physical results and practical infringing use.

To the extent that software is seen as text, as characters on paper or on a screen, that software does not infringe a patent, at least in my view. Instead, the text form of software is protected by copyright which publication could infringe. But when software is used to create a logic machine which performs real computation and produces real results, that *can be* an infringing use, both of the machine, and of the software that created that particular logical machine within the physical machine or computer.

Sophist

Someone who believes that unarguable facts simply do not exist or cannot be known. That leaves rhetoric as the process of sophist proof and persuading the listener as the only goal. Sophists can feel free to convince anyone of anything without the tedious requirement of first acquiring knowledge.

Sophistry

The art of projecting the appearance of wisdom which one does not have (Aristotle, 350 B.C.E.).

Source Code

The textual representation of a computer program as it is written by a programmer. Nowadays, source is typically in a high-level language like C, C++ or Pascal, but inevitably some programmers must work "close to the machine" in assembly language. The "code" part of this is presumably an extension of the idea that, ultimately, all computer programs are executed as "machine code" or machine language. This consists of numeric values or "operation codes" ("opcodes") which select the instruction to be executed, and so represent a very public *code* for those instructions. Also see object code.

S-P

Substitution-permutation ciphering.

Special Prime

Prime P where P = 2*P1 + 1 and P1 = 2*P2 + 1, for odd primes P1 and P2. As defined in the original BB&S articles. The articles give 2879, 1439, 719, 359, 179, and 89 as examples of special primes (but 179 and 89 appear to not be special, while 167, 47 and 23 should be).

Specification

In engineering, stated limits within which a system or component must perform. For example, a bridge is specified to be limited to vehicles of some particular maximum weight, and if heavier vehicles use the bridge, it may not stay up. The specification of guaranteed properties is at the heart of hiding component complexity in a black box which "somehow" performs the specified operations. (Also see: quality management.)

The ideal specification document is:

- Absolutely clear, and preferably short;
- Specific and unambiguous;
- Absolute: measurable without reference to something else; and
- Correct.

In practice, these goals are difficult to achieve. It is often hard for the author to see that a specification is ambiguous or even wrong until an error actually is made in one or more implementations taken from that spec. There is a temptation to describe new systems by analogy to old ones, but the success of that often depends upon the reader having the same background as the author, which is never the case. One way to minimize ambiguity is to describe functions in several different ways, so that a misinterpretation of one description can be compared to another description and caught.

Good specifications tend to be full of dichotomy, so that any particular implemented feature can be seen to be either in, or out, of specification, with little or no in-between "gray area."

An electronic digital logic device is specified to produce correct results when it is operated in a given voltage range, in an environment of a given temperature range, with signals having appropriate setup and hold times, and so on, across the full range of production variation. When any of these parameters are violated in use, the device cannot be depended on to produce correct results; it may, or it may not. It is up to the using system designer to make sure component specs are obeyed.

A crystal oscillator is normally specified to operate on some frequency, to some given accuracy, over variations in time, temperature and supply voltage. In a real device we may expect frequency variation to occur, but when that is specified, probably it would be either a "typical" value or a "maximum." The device would be performing to spec if no variation occurred at all. So a maximum does not describe how the device actually performs. If we depend upon component action which is not specified, we may find a time when those actions do not occur.

Speculation

An unverified possibility or prediction. Often a basis for belief. Since speculations may be wrong, they are risks in any analysis.

Spin

In argumentation, a biased interpretation of a statement of fact.

We often see a complex system, such as life, as one or more simplified models of reality. Within any particular model, a fact may have various implications which we can call "good" or "bad." The model is thus the *context* for a value judgment on the data. However, by moving to a different or larger model, different and perhaps contradictory implications may be seen. This can be done honestly by exposing the various models to analysis and discussing which is appropriate and why.

However, a common method of deceit is to avoid mentioning that multiple viewpoints are possible, and that in judging a statement, some views may be more useful than others. It is instead common to "skip to the bottom line." In this way, a "success" can be branded a "failure" (and vise versa), simply by choosing the area or level of discourse. The fallacy is that there may *be* no single "bottom line," or the one implied may not be best or even appropriate. The logic fallacy might be called "Poor Context by Misdirection."

SPN

Substitution-Permutation Network. S-P.

Squares

Some discrete function objects used in cipher designs, including:

- Latin squares
- orthogonal Latin squares

And possibly related to:

- S-Boxes
- Substitution Tables

Square Wave

The graphic waveform of amplitude versus time which has only two amplitude values, and which changes between those values: A digital wave. Ideally, a "square" wave will spend the same amount of time at each level, but any digital waveform is going to be more "square" than a sine wave, which is the classic analog wave.

When viewed on an oscilloscope, a square wave may seem to have only horizontal lines representing the period at the high level and the period at the low level: In modern digital logic, the vertical transitions can be too quick to produce viewable lines.

SSL

Secure Sockets Layer.

Stability

The extent to which a device under mesure will remain invariant, given changes in time, temperature, power, etc.

Stake

A value at risk of loss.

Standard Cipher

A cipher standardized by some recognized group. The National Bureau of Standards, under the US Department of Commerce, first standardized the block cipher DES. The same organization, renamed NIST, more recently standardized the block cipher AES.

Normally, business loves standards so much that each company contributes many of its own. Oddly, that is not the case in cryptography, which is just where a multiplicity of standards could be most beneficial. (The earliest known source for this idea is Shannon's 1949 Algebra of Secrecy Systems.)

The usual reason for a standard is to allow separate development groups to build systems which interoperate. However, in cryptography, the whole purpose of a key is to *restrict* operation to only those who have the correct key. However and wherever we distribute a key, we could almost as easily distribute a complete cipher, independent of any standard.

The problem is that a standard cipher is the ideal target for our opponents: If all secret information in society is protected by one cipher, success in attacking it has the greatest possible payoff. And a single puzzle with the greatest possible payoff has the greatest possible risk of being solved. It is not as though we can avoid that risk by cryptanalysis, because even extensive analysis can at best show only that no weakness has been found, not that no weakness exists.

The worst thing for our opponents would be a need to identify, obtain and break an ever increasing stream of new ciphers. Not only is doing that billions of times more expensive than simply trying another key in a standard cipher, it also distributes the secret information of society in a multitude of fundamentally different channels. Breaking into any one or even a group of those channels would have a vastly smaller payoff than breaking any single standard. That vastly increases the cost of obtaining protected information.

Ideally, what we would have is not standard ciphers, but a few standard cipher *interfaces*, each of which could handle a broad class of different ciphers. Then we could select from among a continually increasing set of ciphers, perhaps on a message-by-message basis.

Even more risk protection is available using multiple encryption.

Standard Deviation

In descriptive statistics, the square root of the variance. A value which describes the "dispersion" or variability around the mean. If we have a normal distribution, about 68% of our data values should be within +/- 1 standard deviation about the mean. About 95% of the values should be within +/- 2 standard deviations, and over 99.7% should be within +/- 3 standard deviations.

State

Information storage, or "memory," or the contents of that memory. In abstract machine theory, retained information, generally used to influence future events.

In statistics, the current symbol from a sequence, or a value which selects or conditions possible outcomes (see: Markov process).

We normally measure "state" in units of information or bits, and 8 bits of "state" can support 2^8 or 256 different state-value combinations or *states*.

Also see: deterministic and keyspace.

State Machine

Finite state machine.

Static Electricity

Electrical energy which is not "moving." Stored electrical energy in which current is not flowing and the voltage is not changing direction or oscillating. Basically high-voltage electrical energy stored on the physical capacitance between objects.

Stationary Process

In statistics, a stochastic (random) process (function) whose general statistics do not change over time. Specifically, the expectation value of each sample is the same over all time, and the autocorrelation structure also does not change for any origin time. In a stationary process, every sub-sequence is equally representative of the whole. This may not be true of a Markov process. Also see: ergodic process.

Statistic

A computation intended to reduce diverse results into a one-dimensional ordering of values for better understanding and comparison. Also the value result of such a computation. See statistics.

A useful statistic will have some known (or at least explorable) probability distribution for the case "nothing unusual found." (For example, see the "Normal, Chi-Square and Kolmogorov-Smirnov Statistics Functions in JavaScript" page, locally or @: http://www.ciphersbyritter.com/JAVASCRP/NORMCHIK.HTM.) This allows the statistic value to be interpreted as the probability of finding that value or less, for the case "nothing unusual found." Then, if improbable statistic values occur repeatedly and systematically, we can infer that something unusual *is* being found, leading to the rejection of the null hypothesis.

It is also possible to explore different distributions for the same statistic under different conditions. This can provide a way to guess which condition was in force when the data were obtained.

Statistics

1. The mathematical science of using probability to extract meaning from data.
2. The analysis of a large population based on a limited number of random samples from that population.
3. The ability to state probability bounds for the correctness of certain types of inductive reasoning.

Statistical Limitations

When extracting meaning from data, statistics rarely if ever can provide the absolute certainty we often associate with mathematical proof. Statistics often provides a numerical answer to the question: "How frequently would random sampling of the null distribution produce the observed results?" If the answer is "almost never," we might *assume* the results were due to something else. But the assumption can be wrong. Since certainty is not possible, we should not expect it, and cannot rely upon it.

When random sampling is used to estimate a population as a whole, any possible values can occur, no matter how rare they are on average. This effect can produce wide variations in the predicted results which may be surprising to those who expect only the most probable outcomes. Only when many trials are conducted can we reasonably assume that the experimental results reflect reality instead of chance illusion.

Statistics and Cryptography

Statistics normally does not address the context of deliberate puzzles and the dynamic nature of cryptographic conflict (the cryptography war). Statistical comments made about the quality of a RNG simply do not consider the cryptographic requirement for unpredictability. Even a very good statistical RNG is unlikely to be good enough for cryptography, simply because it has not been deliberately constructed with the goal of being unpredictable.

Statistics in General

Statistics supports two kinds of scientific investigation:

- *Surveys*, based on observation only, and
- *Experiments*, which set up conditions and observe results.

Also see: statistic and random variable.

Experiments are *comparisons* conducted to assign differences in results to different conditions. Statistical experiments are intended to show that something happens not just at random, but as a consequence of known conditions. Normally, experiments define a null hypothesis that nothing will happen, or that the normal or random result will occur, or that multiple things each produce the same result. Typically, statistics provides a numerical probability for the observed events occurring as a result of only pure chance, *assuming* the null hypothesis is true. When the observed results seem sufficiently improbable under that assumption, the null hypothesis is rejected.

A common role of statistics is to identify particular systematic events in the context of expected random variations that may conceal such events. This often occurs in a context of difficult and costly experimentation, and there is a premium on results which are so good that they stand above the noise; it may be that not much is lost if a weak positive is ignored.

In contrast, cryptography and randomness testing generally support vast amounts of testing at low cost, we seek weak indications, and false positives are a particular danger. In this context, we may find it more useful to conduct many trials and collect many statistic values, then visually and mathematically compare the experimental distribution to the expected or null distribution for that statistic.

How Statistical Distribution Testing Works

Population statistics depend upon random sampling, then computing a statistic result from the samples, then computing a p-value for that result. For any particular range of sampled items, mere chance can yield all big ones, or all small ones, or anything in between. But, taken over many different trials, a mix of items will be far more likely, and the statistic values corresponding to mixed results generally occur more often.

For any particular experiment, there is some set of expected innocuous data which, when sampled in many different trials, will produce an expected distribution of statistic values. This is the "nothing unusual found" null distribution, typical of the null hypothesis, that we can compare against future results.

Most statistics have a related computation which produces a p-value, which is just the probability that statistic value (or less) has in the null distribution. The computation thus generally converts a raw statistic distribution with a large hump in the middle to a flat or uniform probability distribution. So, when sampling innocuous data, we should get an even distribution of p-values over the entire range from 0 to 1. A very *high* p-value will thus indicate that such a statistic value is unlikely when sampling innocuous data. And very *low* p-values are also unlikely.

To reject the null hypothesis, all we have to do is show that we get a distribution of statistic values which is significantly different from the null distribution. That is, we just show that the p-values do not occur in a uniform distribution. The problem is that random sampling itself can produce very unusual statistic values even with innocuous data. Thus, simply showing that we got one unusual statistic value normally is not enough, unless further testing is too expensive or even impossible, and then we live with the possibility of error.

The alternative is to run enough trials to characterize the experimental distribution well enough so it can be compared to the null distribution in many different places. Often the difference will be clear to the eye after just a few trials and will get worse, so we can discontinue testing. In some cases, when detecting a small signal in large random data, the difference will be minor, and that difference will need many trials to be evaluated by some other statistical test. But if we are actually rooting for a null result (as in randomness testing), we have to *actually run* all those trials and make the full statistical comparison before we can say that nothing is there. And then we have to do all that again for every pattern or relationship we want to test.

Summary of Randomness Testing

An expected statistical distribution usually represents what we should expect from random data or random sampling. If we have random data, statistic values exceeding 95% of the distribution (often called *failure*) *should* occur about 1 time in 20. And since that one time may happen on the very first trial, it is only prudent to conduct many trials and accumulate results which are more likely to represent reality than any one result from a single trial.

In statistical randomness testing, we are concerned not only with having too many "failures," but also with having too few. "Failure" *must* occur with the appropriate probability: The failure to fail is itself a failure! (Note that the very concept of statistical "failure" may be inappropriate for cryptographic use. Grading a result as "pass" or "fail" discards all but one bit of information and converts it into a Bernoulli trial, which would take many, many similar trials to properly characterize.) It is thus more appropriate to conduct 20 or more trials, collect those statistic probability values, and then compare them to the expected flat p-value distribution. Those results will provide a substantial basis for asserting that the sampled process either did or did not produce the same statistic distribution as a random process.

Steganography

Greek for "sheltered writing." Methods of cryptology which seek to conceal the *existence* of a message. As opposed to cryptography which seeks to hide the *information* in the message, even if the message itself is completely exposed.

Stochastic

In statistics, random; involving a random variable. A process which is predictable only in a statistical sense. As opposed to deterministic.

Stochastic Process

In statistics, a function which produces symbols at random, each with an individual probability or frequency of occurrence. One specific type is the Markov process.

Stream

A sequence. A continuing flow. The concept of one thing, then another, then another. A sequence of individual elements, as opposed to collections of two or more elements, as in blocks.

Stream Cipher

A cipher which directly handles messages of arbitrary size by ciphering individual data elements, such as bits or bytes or characters. Conventionally, some form of keyed random number generator (RNG) is used to produce a confusion sequence or running key or keystream. That sequence is then combined with plaintext data to produce ciphertext. In contrast to a block cipher. (Also see stream cipher classes, such as synchronous, self-synchronizing (asynchronous) and re-originating.)

A wide range of different RNG designs can be virtually "dropped into" a general stream cipher design. Unfortunately, most common statistical RNG's are very weak cryptographically, and so need some form of nonlinear isolation (see, for example: jitterizer). Conventionally, the combiner is just exclusive-OR or some other additivive combiner which adds no strength because it is known, linear and easily reversed. However, a wide range of keyed nonlinear and/or dynamic combiners (such as Dynamic Substitution and Latin square combiners) can be virtually "dropped in" to a general stream cipher design.

Enciphering individual characters allows ciphering to begin immediately, avoiding the need to accumulate a full block of data before ciphering, as is necessary in a conventional block cipher. But note that a stream cipher can be seen as an operating mode, a streaming of a tiny transformation. Stream ciphers can be called "combiner-style" ciphers. Also see: a cipher taxonomy.

Some academics insist upon distinguishing *stream* versus *block* ciphering by saying that block ciphers have no retained state between blocks, while stream ciphers do. Simply saying that, however, does not make it true, and only one example is needed to expose the distinction as false and misleading. A good example for that is my own Dynamic Transposition cipher, which is a block cipher in that it requires a full block of data before processing can begin, yet also retains state between blocks. (This discussion is continued in Block Cipher and Stream Cipher Models.)

Stream Cipher Diffusion

In a conventional stream cipher, each element (for example, each byte) of the message is ciphered independently, and does not affect any other element.

In a few stream cipher designs, the value of one message byte may change the enciphering of *subsequent* message bytes; this is forward data diffusion. But a stream cipher **cannot** change the enciphering of *previous* message bytes. In contrast, changing even the last bit in a block cipher block will generally change about half of the earlier bits within that same block. Changing a bit in one block may even affect later blocks if we have some sort of stream meta-cipher composed of block cipher transformations, like CBC.

Note that a stream cipher generally does not need data diffusion for strength, as does a block cipher. In a block cipher, it may be possible to separate individual components of the cipher if their separate effects are not hidden by diffusion. But a stream cipher generally re-uses (or, perhaps, re-keys) the same transformation, and has no multiple data components to hide.

Stream Cipher Construction

When an exclusive-OR combiner is used, exactly the same construction will also decipher the ciphertext; this is a self inverse or involution. But if the opponents have some known-plaintext (with the associated ciphertext), they can easily produce the original confusion sequence just by doing an XOR. This, along with their expected knowledge of the cipher design, may allow them to attack and expose the confusion generator or RNG. If this is successful, it will, of course, break the system until the RNG is re-keyed.

The ultimate stream cipher is the one-time pad, in which a really random sequence is never re-used. If a sequence *is* re-used, the opponent generally can combine the two ciphertexts, eliminating the confusion sequence, and producing the combined result of two plaintexts, which is normally "easy" to attack and penetrate. However, the main issue is not really "re-use" as much as it is "predictability": The data are exposed as soon as the sequence is predicted, whether that sequence has been "reused" or not. This highlights the need for cryptographic strength (unpredictability) in a stream cipher RNG.

The re-use of confusion sequence is extremely dangerous in a stream cipher design (see re-originating stream cipher). In general, stream cipher designs *must* use a message key (or some similar facilty) to assure that the cipher is keyed with a random value for every new ciphering. That does, of course, expand the ciphertext by the size of the message key.

Another alternative in stream cipher design is to use a stronger combiner, such as Latin square or Dynamic Substitution combining. This can drastically reduce the complexity required in the confusion generator, which normally provides all stream cipher strength. Each of these stronger combiners is nonlinear, with substantial internal state, and the designer may elect to use multiple combinings in sequence, or a selection among different combiners. Neither a sequence of combinings nor selection among combiners makes much sense with an additive combiner.

Strength

The ability of a cipher to resist attack and maintain secrecy. The overall "strength" of a cipher is the minimum effort required to break the cipher, by any possible attack. But our *knowledge* of cipher "strength" is necessarily contextual and subjective, much like *unpredictability* in random sequences. Although "strength" would seem to be the entire point of using a cipher, cryptography has no way to measure strength. Also see: scientific method, military grade and threat model.

Cipher "strength" is often taken as an absolute universal *negative,* the simple *non-existence* of any attack which could succeed, assuming some level of attack resources. But this means that overall "strength" may be forever impossible to measure, because there is no hope of enumerating and evaluating *every possible* attack.

In practice, cipher "strength" is inherently contextual, and depends upon the knowledge, experience, and intelligence of the attacker, as well as the available equipment and time. The contextual nature of strength is well-known: Clearly, academics do take pride in being the special individuals who can break particular ciphers, and if strength were not contextual, everyone would get the same result. The problem is that our opponents might be even *more* special, with even *better* attacks. Because practical strength is contextual, we cannot know the strength unless we know the context,and we cannot know the context because that is secret.

Strength and Cryptanalysis

Because we have no tools for the discussion of strength under all possible attacks, cipher "strength" is normally discussed in the context of particular attacks. Each known attack approach can be elaborated for a particular cipher, and a value calculated for the effort required to break the cipher in that way; this may set an "upper bound" on the unknown strength of the cipher (although some "elaborations" are clearly better than others). And while this is certainly better than not knowing the strength with respect to known attacks, such attacks may not represent the actual threat to the cipher in the field. (A cipher may even be said to have *different* "contextual strengths," depending on the knowledge available to different opponents.) In general, we never know the "lower bound" or "true" strength of a cipher. So, unless a cipher is shown to be weaker than we can accept, cryptanalysis provides no useful information about cipher strength.

It is sometimes argued that "our guys" are just as good as the opponents, who thus could not break a cipher with less effort than we know. Or it is said that if a better break were known, that secret necessarily would get out. When viewed in isolation such statements are clearly false reasoning, yet these are the sort of assumptions that are often implicitly used to assert strength after cryptanalysis.

Since we cannot know the true situation, for a proper security analysis we must instead assume that our opponents have more time, are better trained, are better equipped, and may even be *smarter* than our guys. Further, the opponents are quite likely to function as a well-motivated group with a common goal and which can keep secrets; clearly, this is a far different situation than the usual academic cryptanalysis. So, again, cryptanalysis by "our guys" provides **no** information about the strength of the cipher as seen by our opponents.

No Expertise on Strength

If cipher strength is the ability to protect information from opponents, that is a goal which can neither be tested nor verified. There are no tests for strength, only particular attacks which, if they succeed, show weakness. Although we seek strength, the only thing we can know as fact is that a cipher is "not known to be weak."

Without experimental confirmation, virtually no science remains. Nobody can be an expert if the results of their answers cannot be checked against reality; in other fields, such people are called "quacks." Without tests, there can be no expertise on cryptographic strength, anywhere, by anybody, in any organization, no matter how well educated or experienced. Unfortunately, strength is what people expect from a cipher, and expertise on strength is what many customers and users expect from a crypto expert.

Naturally, after weakness is found and defined, then we have a testable and verifiable issue. When a cipher can be made to expose information, knowing how to do that can be expertise on weakness. But even that expertise is necessarily limited, because there is no end to ciphering techniques, each of which might be attacked in essentially unlimited ways.

Increasing Probable Strength and Reducing Possible Loss

Cipher weakness is just one of the many possible problems in a cipher system that includes users. For example, users must be educated in security, and must actively change their practices to keep secrets or there will be nothing to protect. Getting people to change the way they do business is always difficult.

Even if we assume the ciphers are not weak, it is surprisingly difficult to construct a cipher system without "holes," and the opponents get to exploit any overlooked weakness without telling us about it. Many people look to cryptanalysis to expose weakness, but cryptanalysis is very expensive and always incomplete. Actual problems may be more system specific than generic.

It is a disturbing fact that we do not know and cannot guarantee a "true" strength for any cipher. But there *are* approaches which may reduce the probability of technical weakness and the extent of any loss:

1. We can *extrapolate* various attacks beyond weakness levels actually shown, and thus possibly avoid some weak ciphers.
2. We can use systems that allow us to switch to new ciphers when the old ones are shown weak.
3. We can use systems that change ciphers periodically. This will reduce the amount of information under any one cipher, and so limit the damage if that cipher is weak. (See Shannon's Algebra of Secrecy Systems.)
4. We can use multiple encryption with different keys and different ciphers as our standard mode. In this way, not just one but multiple ciphers must each be penetrated simultaneously to expose the protected data. (Again, see Shannon's Algebra of Secrecy Systems.)

Kinds of Cipher Strength

In general, we can consider a cipher to be a large key-selected transformation between plaintext and ciphertext, with two main types of strength:

- One type of "strength" is an inability to extrapolate from known parts of the transformation (e.g., known plaintext) to model -- or even approximate -- the transformation at new points of interest (message ciphertexts).
- Another type of "strength" is an inability to develop a particular key, given the known cipher and a large number of known transformation points.

Views of Strength

Strength is the effectiveness of fixed defense in the cryptography war. In real war, a strong defense might be a fortification at the top of a mountain which could only be approached on a single long and narrow path. Unfortunately, in real military action, time after time, making assumptions about what the opponent "could not" do turned out to be deadly mistakes. In cryptography we can at least imagine that someday we might *prove* that all approaches but one are actually *impossible,* and then guard that last approach; see mathematical cryptography.

The Future of Strength

It is sometimes convenient to see security as a fence around a restricted compound: We can beef up the front gate, and in some way measure that increase in "strength." But none of that matters if someone cuts through elsewhere, or tunnels under, or jumps over. Until we can produce a cipher design which reduces all the possible avenues of attack to exactly one, it will be very difficult to measure "strength."

One possibility might be to construct ciphers in layers of different puzzles: Now, the obvious point of having multiple puzzles is to require multiple solutions before the cipher is broken. But a perhaps less obvious point is to set up the design so that the solution to one puzzle requires the opponent to *commit* (in an information sense) in a way that *obscures* the solution to the next puzzle.

Also see design strength, Perfect Secrecy, Ideal Secrecy, and security.

Strict Avalanche Criterion (SAC)

A term used in S-box analysis to describe the contents of an invertible substitution or, equivalently, a conventional block cipher. If we have some input value, and then change any bit in that value, we expect about half the output bits to change; this is the avalanche effect, and is caused by an avalanche process. The *Strict Avalanche Criterion* requires that each output bit change with probability one-half (over all possible input starting values). This is stricter than avalanche, since if a *particular* half of the output bits changed *all* the time, a strict interpretationist might call *that* "avalanche." Also see complete.

As introduced in Webster and Tavares:

> "If a cryptographic function is to satisfy the strict avalanche criterion, then each output bit should change with a probability of one half whenever a single input bit is complemented." [p.524] -- Webster, A. and S. Tavares. 1985. On the Design of S-Boxes. *Advances in Cryptology -- CRYPTO '85.* 523-534.

Although the SAC has tightened the understanding of "avalanche," even SAC can be taken too literally. Consider the scaled-down conventional block cipher model of a small invertible keyed substitution table: Any input bit-change thus selects a different table element, and so produces a random new value (over all possible keys). But when we compare the new value with the old, we find that typically half the bits change, and sometimes *all* the bits change, but *never* is there no change at all. This is a tiny bias toward change.

If we have a 2-bit (4-element) table, there are 4 values, but after we take one as the original, there are only 3 possible *changed* values, not 4. We will see changes of 1 bit, 1 bit, and 2 bits. But this is a change expectation of 2/3 for each output bit, instead of exactly 1/2 as one might interpret from SAC. Although this bias is clearly size-related, its source is *invertibility* and the definition of *change.* Thus, even a large block cipher *must* have *some* bias, though it is unlikely that we could measure enough cases to see it. The point is that one can extend some of these definitions well beyond their intended role.

Structured Programming

Originally, programming with looping structures instead of GOTO. Now, the general body of programming style which simplifies developing, reading and understanding a program.

Old-Style Programming

Early in the *mini*computer era (and presumably before), it was common to manually write routines directly into CPU memory. When supported with some routines that move code and automatically correct for address changes, and other routines which document the current system, machine-code programming still can be a useful approach. But without such support, routines tend to be fixed in place, and expanding those routines means jumping to unused areas and then jumping back. Eventually, this leads to a complex mass of intertwined code which is particularly difficult to debug at the machine language level because various aspects of a single conceptual routine are found in multiple areas.

Similarly, early in the *micro*computer era, many people used (or had learned on) the original BASIC language, in which each line of code or statement had a global line number. When new code was added, it could only expand between existing line numbers, and when no room was left between lines, the only thing to do was to GOTO some other place. The resulting programs were aptly described as "spaghetti code," as the execution path jumped back and forth in no way related to the problem at hand.

The alternative to spaghetti code is to eliminate line numbers and allow new code to be placed anywhere. (We might now consider that an editing issue, but in the beginning there were no editors, and languages had minimal editing support.) But even without spaghetti code, we can still end up with a huge, monolithic program, in which many complex things occur in one main execution path. Even worse, the only thing we can test in such a structure is the whole program itself, and there is no way we can test every combination of every parameter and option and use to verify that the program works.

The alternative to having a monolithic program structure is to collect each task into an individual routine, and then call or *invoke* those routines. The idea of collecting a sequence of instructions and invoking that sequence with a single command exists at the hardware level. However, the subroutine call in original BASIC (and in the hardware) only transferred control, not data. Nor could original BASIC subroutines declare and use variables which existed only in the subroutine. In practice, that meant original BASIC subroutines had to use program global variables for subroutine local work, and any part of the program could change those values. Moreover, subroutines could only work on particular named global data, and so were difficult to use to do the exact same thing on different data, or in a different program.

Modern Programming

Modern languages (including updated BASIC) support passing values or parameters to subroutines or procedures or functions. Parameters allow the same operations to be performed on different data. That means we can have one version of the code, instead of multiple slightly-different versions, some of which may not get updated when changes are made.

Modern languages also allow the declaration and use of local variables which are only valid within those routines. That means a procedure can be tested without having to set up a suite of named variables in the higher-level program. Setting up such variables can be particularly difficult if the program is already using variables with those names for different things. Having local variables also means the values cannot be changed by every line of code in the whole program. Only the code in the local procedure (or perhaps variables passed to other procedures) could possibly make such changes. So if we find that *something* is changing a variable to a bad value, we can confine our search for the problem. That error-localization is a crucial advantage in debugging.

The combination of parameters and local variables allows a subroutine to be considered an engineering black-box operation that does something, without making us understand *how* it does it. Moreover, the subroutine becomes stand-alone, and so can be tested in a completely different environment. Testing a procedure thus does not require running the whole larger program with vastly more complexity and interactions. This ability to encapsulate, test, and hide complexity, sometimes called "modularity," is the basis for modern programming. This goes far beyond the simple non-use of GOTO.

Personally, I find that the cause of modular programming is sometimes best served by the infrequent and judicious use of GOTO. Sometimes a routine has explicit entry and exit code, with exit code that should be executed even when an error occurs. Detecting an error and using GOTO to jump to the exit code which leads out is clear and easy. Modern "exceptions," while sometimes useful, are less clear (an error is often not handled where it occurs, but at some *unknown* higher level, which then must be emulated in testing), add disjoint complexity, and are more costly.

The exact details of good modern programming vary somewhat according to personal style. Nevertheless, there is broad, general agreement that some things are worthwhile, even if a particular programmer has another solution. For example:

Project

- Plan to release multiple, incremental versions, each with complete tests for that version. While the early versions may not do much, they will demonstrate investment results.
- Plan to do the hardest things first, so any problems become evident early.
- Schedule the project by module.
- Follow progress by completed, tested and accepted module count, not lines of code (LOC).
- After the hard things are handled in the first few releases, the rest of the project should be more predictable.

System Design

- Think through what you want to do before you do it. For anything larger than a single routine, do some design before writing code. You need to know where you are, where you are going, and what rules will guide your development.
- To reduce development complexity, simultaneously both *decompose* the problem into testable, information hiding routines, and also *compose* low-level routines for needed tasks, until development meets in the middle.
- Ideally, each procedure or function interface should be specified in detail before the code body is written. (Nevertheless, expect some changes.)
- When several similar functions pop up in the design, try to combine them into one common concept for single implementation.
- Program in small stand-alone units which can be individually understood and tested (e.g., procedures, functions, subroutines, etc.)
- Pass all needed information to the routine as parameters, and take results the same way. Minimize the number of parameters, but also minimize the use of globals, and "never" access globals directly.

Formatting

- Program for simplicity, clarity and ease of understanding.
- When the code to implement a function gets substantially longer than a page or a screen, start thinking about breaking it into several smaller routines.
- Describe the purpose of a routine in comments either immediately before or immediately after the declaration. The idea is to read *what* a routine does, without having dissect *how* it does it. In this way, complexity is encapsulated and unnecessary information hidden from the higher level.
- Choose and use meaningful and unambiguous names in all cases, including routines, variables, structures, classes, etc.
- Use whitespace (blank lines) to group code lines as appropriate.
- Use comments to describe grouped code (*not* what the code does, but instead what the code was *intended* to do).
- Do use appropriate looping structures for all loops and minimize the use of GOTO.
- Choose and use an indention scheme which reflects loop and decision levels.

Logic

- Always check and handle every possible failure return, especially when calling OS routines. Assume that if something can possibly fail, someday it will. Consider using the function return value for a failure code, and return values by parameter.
- Write a simple but comprehensive set of tests for "each" routine programmed. Run the tests and change the routine until it works in all cases. Extend the tests whenever the routine is extended, or whenever an

error is found. Run the tests frequently. (Programmers need their own tests to verify that their constructions work. The project, however, should have a separate testing group for serious system testing.)

- First make the routine work; only then make changes for efficiency.
- Only worry about efficiency if that will be important in the final program, *and* if you will instrument the routine to measure efficiency improvements.
- Assembly language is almost never worth the trouble anymore, except for:
    - Code which will be in almost continual execution, or which will be in an environment with resource constraints, or
    - Access to useful CPU instructions not available in a high level language (HLL).
- Eliminate or minimize the possibility of external scrambled data causing the routine to fail. The routine should identify bad input and if necessary fail in a safe way. One option is to have routines return a value indicating success (or failure); another option is to pull an exception to be handled at a higher level. But no routine should ever lock up no matter what data it gets.
- Use parenthesis freely in numerical expressions to clarify and force the desired evaluation order. (Some people say that unnecessary parenthesis are unprofessional, but *I* say it is unprofessional to demand from the reader both a knowledge of operator precedence and a correct interpretation of the evaluation order in a complex expression. Trying to make code be as readable as possible, even to people across a wide range of skills, is a Structured Programming issue.)

Also see: software, system design, Software Engineering and cipher system.

Subjective

In the study of logic, a particular *interpretation* of reality, rather than objective reality itself.

Substitution

The concept of replacing one symbol with another symbol. The cryptographic strategy of changing message-element *values* instead of changing message-element *positions*, as is done in transposition. Also see code.

Substitution might be as simple as a grade-school lined sheet with the alphabet down the left side, and a substitute listed for each letter. In computer science this might be a simple array of values, any one of which can be selected by indexing from the start of the array. See substitution table.

Cryptography recognizes four types of substitution:

- Simple Substitution or Monoalphabetic Substitution,
- Homophonic Substitution,
- Polyalphabetic Substitution, and
- Polygram Substitution.

The concept of substitution is extremely general and can seem to encompass virtually all of cryptography. However, substitution alone does not hide plaintext frequencies, which makes relatively small substitutions fairly weak. Moreover, modern cryptography is not *just* substitution, but also keying, in this case the arbitrary selection of one particular substitution table from among all possibilities.

Substitution Cipher

The classic cipher in which individual plaintext elements are substituted or replaced by ciphertext values, typically taken from a substitution table. The particular ordering of that table is the key.

Modern block ciphers are emulated huge substitution ciphers where the substituted value is computed, instead of being available in a table.

Also see: substitution, simple substitution and a cipher taxonomy.

Substitution-Permutation

(S-P and SPN). A method of constructing block ciphers in which data elements are substituted, and the resulting bits are transposed into a new arrangement or permutation. This would be one of several computation layers or rounds used in a complete cipher.

The big advantage of S-P construction is that the permutation stage is simply a re-arrangement or scrambling of connections rather than a new full component. And, in hardware, a scrambling of wires needs no devices and takes almost no time.

A disadvantage of the S-P construction is the need for special substitution table contents: S-P ciphers diffuse bit-changes across the block round-by-round. So if one of the substitution table output bits does not change, then no change may be conducted to one of the tables in the next round, which reduces the complexity of the cipher. Consequently, special tables are required in S-P designs, which means that keying can be very complex instead of an easy shuffle. But even special tables can only reduce and not eliminate the effect. See complete, ideal mixing, Balanced Block Mixing and Mixing Cipher.

Substitution Table

(Also S-box.) A linear array of values, indexed by position, which includes any value at most once. In cryptographic service, we normally use binary-power invertible tables with the same input and output range. For example, a byte-substitution table will have 256 elements, and will contain each of the values 0..255 exactly once. Any value 0..255 into that table will select some element for output which will also be in the range 0..255.

For the same range of input and output values, two invertible substitution tables differ only in the order or permutation of the values in the table. There are 256 factorial different byte-substitution tables, which is a keyspace of 1648 bits.

A keyed simple substitution table of sufficient size is the ideal block cipher. Unfortunately, with 128-bit blocks being the modern minimum for strength, there would be 2^{128} entries in that table, which is completely out of the question.

A keyed substitution table of *practical* size can only be thought of as a weak block cipher by itself, but it can be part of a combination of components which produce a stronger cipher. And since an invertible substitution table is the ideal tiny block cipher, it can be used for direct experimental comparison to a scalable block cipher of that same tiny size.

Superencipherment

Typically, a relatively simple additive cipher intended to hide an otherwise unchanging code. Also see superencryption.

Superencryption

Usually, the outer-level encryption of a multiple encryption. Classically, superencryptions are relatively weak, relying upon the randomization effect of the previous encryption level.

Superposition

In a quantum computer, the property of quantum mechanics that allows a quantum computation to be in many different states simultaneously.

Surjective

Onto. A mapping f: *X* -> *Y* where *f(x)* covers all elements in *Y*. Not necessarily invertible, since multiple elements *x* in *X* could produce the same *f(x)* in *Y*.

Switch

Classically, an electro-mechanical device which physically presses two conductors together at a contact point, thus "making" a circuit, and also pulls the conductors apart, thus allowing air to insulate them and thus "breaking" the circuit. More generally, something which exhibits a significant change in some parameter between "ON" and "OFF."

Switching Function

A logic function.

Symmetric Cipher

A secret key cipher.

Symmetric Group

The symmetric group is the set of all one-to-one mappings from a set into itself. The collection of all permutations of some set.

Suppose we consider a block cipher to be a key-selected permutation of the block values: One question of interest is whether our cipher construction could, if necessary, reach every possible permutation, the symmetric group.

Synchronization

1. In digital electronics, causing signals to align in time so that reliable computation can occur. A digital "bit" is not only a logic voltage, but also a *time* at which that level is guaranteed to be steady and so can be reliably sampled for use.
2. In software stream ciphers, causing the exact same confusion sequence both when enciphering and deciphering. The two processes are said to be "synchronized" if they produce the same confusion value at each message position, even if the ciphertext is stored and deciphering deferred to a later time.

Digital signals which have some delay, but exactly the right data rate, are often synchronized by latching the signal to the desired clock with a flip-flop. But if the data source and sink operate from different (and, thus, asynchronous) clocks, a simple flip-flop will eventually sample the signal when the level is changing, resulting in a largely deterministic form of jitter and also potential metastability problems.

Digital communications generally require the system to produce stable bit-data, sometimes a "valid" indication that new data are present, *and* a corresponding clock-edge when the bit value is stable and can be sampled. Many serial communications systems *also* require synchronization with respect to byte or packet boundaries:

Byte synchronization can be accomplished in various ways:

- The communications hardware can *produce* and then (on the other end) *detect* a physical signal or modulation which cannot occur except at a desired time, such as the start of a packet of data.
- Produce particular bit sequence as a "start" sequence, and then (on the other end) detect that sequence only when awaiting synchronization.
- The classic asynchronous serial-line approach of adding some bits of unchanging value (the "start" and "stop" bits) against which mis-synchronization eventually will be noticed. The edge between "stop" and "start" also defines and starts bit-level timing which is used to synchronize bit-level sampling. This supports reliable communications even when the sending and receiving clocks have somewhat different rates, an issue now largely insignificant with the widespread use of crystal oscillator clocks.

It is possible for signals to be effectively synchronized even when multiple clocks are involved: The classic example is from war movies, where a commando team "synchronizes their watches." Even though two watches may not keep exactly the same time, they can be close enough for practical use. Similarly, different crystal oscillators also may maintain whatever frequency relationship they happen to have, over some amount of time.

Synchronous

Literally "together in time." Typically, digital logic systems in which most signals and state changes are aligned in time.

Typically based on clocked flip-flops, where the outputs change only as a result of a clock edge. Clock edges are spaced to provide time for subsequent combinatoric logic to function, and for signals to fully change to the appropriate logic level before use. This helps avoid metastability problems. As opposed to asynchronous.

Synchronous Stream Cipher

A stream cipher which garbles the rest of the message after a ciphertext position error (i.e., either loses a byte or has a byte inserted; an error affecting stream length). As opposed to an asynchronous stream cipher.

In most stream ciphers, if a byte of ciphertext is lost, or an extra byte somehow inserted, synchronization will be lost and subsequent deciphered data will be incorrect. However, garbling from the error position through the end of the message also can occur from a ciphertext *value* error when using a synchronous stream cipher which has forward data diffusion.

Synthesis

In the study of logic, the composition of parts to form a whole. As opposed to analysis. Also see: thesis, hypothesis, argumentation and scientific method.

System

An interconnecting network of components which coordinate to perform a larger function. Also networks of ideas or people. Also see system design and schematic diagram.

A functioning system is generally expected to provide various services in an available and reliable (dependable) manner. When a service is not provided to specification, that is a failure. (Also see quality.)

A typical cipher system is required to at least provide a security service, as defined by some (formal or informal) specification. Typically, the implied specification is for absolute security in the given environment. But there is no way to test such a claim, and the only way to refute it is to break the system. Breaking the system requires us to know as much as our opponents and also to be as effective as them, which may not be possible.

System Design

The design of potentially complex systems.

It is now common to construct large hardware or software systems which are almost unmanageably complex and never error-free. But a good design and development approach can produce systems with far fewer problems. One such approach is:

1. Decompose the system into small, *testable* components.
2. Construct and then *actually test* each of the components individually.

This is both easier and harder than it looks: there are *many* ways to decompose a large system, and finding an effective and efficient decomposition can take both experience and trial-and-error. But many of the possible decompositions define components which are less testable or even *un*testable, so the testability criterion greatly reduces the search.

Testing is no panacea: we cannot hope to find all possible bugs this way. But in practice we *can* hope to find 90 percent or more of the bugs simply by *actually testing* each component. (Component testing means that we are forced to *think* about what each component does, and about its requirements and limits. Then we have to make the realized component *conform* to those tests, which were based on our theoretical concepts. This will often expose problems, whether in the implementation, the tests, or the concepts.) By testing all components, when we put the system together, we can hope to avoid having to debug multiple independent problems simultaneously. Also see: quality management, engineering, Software Engineering and Structured Programming.

Other important software system design concepts include:

- Build in test points and switches to facilitate run-time inspection, control, and analysis.
- Use repeatable comprehensive tests at all levels, and when a component is "fixed," run those tests again.
- Start with the most basic system and fewest components, make that "work" (pass appropriate system tests), then "add features" one-by-one. Try not to get too far before making the expanded system work again.

Table Selection Combiner

A combining mechanism in which one input selects a table or substitution alphabet, and another input selects a value from within the selected table, said value becoming the combined result. Also called a polyalphabetic combiner.

Tautology

In the study of logic, an expression which is always true and cannot be false. In contrast to contradiction.

Taxonomy

Classification based on common characteristics. See A Cipher Taxonomy.

Temperature

Conversions:

```
C = Celsius   F = Fahrenheit   K = Kelvin

F = (C * 9 / 5) + 32      [ 212 deg F = 100 deg C ]
C = (F - 32) * 5 / 9      [ 176 deg F =  80 deg C ]
                          [ 140 deg F =  60 deg C ]
                          [  80 deg F =  27 deg C ]
                          [  77 deg F =  25 deg C ]
                          [  32 deg F =   0 deg C ]

C = K - 273.15            [  80 deg F ~ 300 deg K ]
K = C + 273.15            [  27 deg C ~ 300 deg K ]
```

TEMPEST

Supposedly the acronym for "Transient Electromagnetic Pulse Emanation Surveillance Technology." Originally, the potential insecurity due to the electromagnetic radiation which inherently occurs in electronic processing when a current flow changes in a conductor. Thus, pulses from digital circuitry might be picked up by a receiver, and the plaintext data reconstructed.

The general concept can be extended to the idea that plaintext data pulses may escape on power lines, or as a faint background signal to encrypted data, or blinking modem lights, or as the reflected light from a computer terminal, or in any other unexpected way. Also see: red, black, shielding and ground loop.

Some amount of current change seems inevitable when switching occurs, and modern digital computation is based on such switching. But the amount of electromagnetic radiation emitted depends upon the amount of current switched, the length of the conductor, and the speed of the switching (that is, dI/dt, or the rate-of-change in current). In normal processing the amount of radiated energy is very small, but the value can be much larger when fast power drivers are used to send signals across cables of some length. This typically results in broadband noise which can be sensed with a shortwave receiver, a television, or an AM portable radio. Such receivers can be used to monitor attempts at improving the shielding.

Another approach to minimizing undesired exposure is to generate balanced signals which radiate equally but with opposite polarity, and so cancel. Similarly, interconnections also may use balanced line, and even shielded balanced line.

Perhaps the most worrisome electromagnetic and optical emitter on a personal computer is the display cathode ray tube (CRT). Here we have a bundle of three electron beams, serially modulated, with reasonable current, switching quickly, and repeatedly tracing the exact same picture typically 60 times a second. This produces a recognizable substantial signal, and the repetition allows each display point to be compared across many different receptions, thus removing noise and increasing the effective range of the unintended communication. All things being equal, a liquid-crystal display should radiate a far smaller and also more-complex signal than a desktop CRT.

Temporal Average

In statistics, the average of symbols across time. The probability of each symbol in a given sample of sequence. Also see ergodic process.

Term

A part of a mathematical expression which is added to other parts. As opposed to factor.

Term of Art

A phrase whose meaning differs from that implied by the actual words. Typically a phrase which has been re-defined or whose meaning has changed over time, while the phrase itself has been retained unchanged. Typically a part of a skilled area of study and practice, such as law, where the classic literature or statutes use the old phrase.

A "term of art" can be very confusing, in that the ordinary language interpretation of the phrase may differ widely from the skilled interpretation. Because "terms of art" often occur in law, trying to understand the law will be almost impossible without the background to understand what each "term of art" really means.

Prior Art: One example is the term prior art as used in patents. Although the phrase "prior art" may seem to mean anything similar at an earlier time, the actual meaning is more like "prior publication." An invention which is not published or in some other way widely and publicly exposed in detail probably does not qualify as "prior art" in patent law. (But see a patent attorney for advice on any particular case.)

First To Invent: Another patent example is that, normally, a U.S. patent belongs to the first one to make an invention. However, if the "first to invent" has chosen to rely upon trade secrecy, he does not qualify for the patent at a later time. And an inventor who was first to invent but who was not first to reduce the invention to practice also may not be awarded the patent. (Also see the discussion: "What's the Meaning of 'Invent'?," locally, or @: http://www.ciphersbyritter.com/NEWS4/FSTINVNT.HTM.)

Proof: Another example is the term proof as used in mathematics. Instead of representing, as one might imagine, a final outcome of absolute truth, "proof" simply may represent "the process of correct reasoning," completely independent of whether or not the outcome can be achieved in practice. In this way, those who lack a substantial background in both cryptography and mathematics can be mislead and deceived by the phrase: "security proof."

Break: Yet another example is the term break as used in academic cryptanalysis. The ordinary interpretation of "broken" would imply that any such cipher was destroyed and thus of no further worth. However, in academic cryptanalysis, "break" merely means that the cipher key can be exposed with less effort than a brute force attack, even if that requires totally impractical amounts of effort and demands unreasonable conditions which are easily prevented at the cipher system level.

One-Time Pad: Another example in cryptography is the one-time pad, which is known to have a theoretical proof of strength. In this context, the ordinary interpretation of "one-time pad" might imply that a single use of any pad would produce proven strength. In fact, however, the relevant property is that the pad be unpredictable, with re-use being just one especially egregious example of predictability.

Block Cipher: One example from academic cryptography is the phrase block cipher. The ordinary interpretation of "block cipher" would simply be a cipher which operates on block values. However, the usual academic meaning is that particular type of cipher which emulates a huge, keyed substitution table or bijection. But not all ciphers operating on blocks are like that; for those that are, a less confusing phrase would be "conventional block cipher." As an example of an "unconventional" block cipher, see Dynamic Transposition, which neither has nor needs diffusion (nor S-boxes, nor rounds, for that matter), and so does not fit the Simple Substitution model. Although the substitution model is fine for a specific type of block cipher (see block cipher and stream cipher models), appropriating the general term "block cipher" for a particular specialized model, no matter how important, seems disturbingly confusing.

Random Number: Surely there *must* be something strange and mysterious about a random number, since no mere value has anything random at all! The term is a misdirection: The randomness (if any) occurs in the curiously unmentioned selection process.

Test

The engineering process of comparing an implemented design to the specification, as measured by coverage.

In Software Engineering there are several different levels of test. First, most subroutines or modules (see Structured Programming, black box and component) should have an individual test routine written by the programmer of the module. The goal of these tests is to check every required function of the module, to see that it works. As the module is implemented it is tested and debugged (or the tests improved) until the module passes all tests . This very basic level is sometimes called *unit testing*.

Next, there should be a group somewhere who also tests modules, including the top-level module which is "the program." This testing should be more comprehensive, and is sometimes called *quality assurance* or QA (also see quality). This testing hopes to identify special cases where routines will not work, or routines that do things they should not, so they can be corrected and not fail in use.

Before QA testing, a programmer can use the modules he implements. But only after a module passes QA tests should it be placed in a library for wider use. QA testing also verifies that the overall system meets specification and so does what it is supposed to do.

Theorem

A statement whose truth has been established beyond doubt. A theorem must be inductively guessed or conjectured before it can be proven. A theorem can be refuted by counterexample. Also see: lemma and corollary.

Theory

A model of reality, generally intended to predict reality or explain how reality works. As opposed to practice.

Typically mathematical, a theory may be correct or incorrect, and will cover only a limited subset of reality. A better theory models reality better (more precisely, or over a wider range), or provides new insight.

Thermal Noise

Johnson noise. The random analog electrical signal caused by the thermal movement of electrons in a resistance, even when not in a circuit or when no external current is applied. Thermal noise is the one-dimensional summation of the random movement of vast numbers of individual electrons, each with a different three-dimensional speed and direction. The variation we see is a statistical effect of event "clumping" when events have an expected rate (here, expected voltage) but independent times. In contrast to shot noise.

Thermal noise is a very weak noise source and a white noise. At room temperature and with a measurement bandwidth of 10kHz, a 10k resistor generates about 1.3uV RMS, while a 1M resistor generates about 13uV RMS.

> "What is the source of Johnson noise? In an ordinary resistor, it is a summation of the effects of the very short current pulses of many electrons as they travel between collisions, each pulse individually having a flat spectrum. In this case the noise is a manifestation of the Brownian movement of the electrons in a resistor." -- Pierce, J. R. 1956. Physical Sources of Noise. *Proceedings of the IRE.* 44:601-608.

Thermal noise voltage is proportional to square-root resistance and square-root absolute temperature. In a bipolar transistor, thermal noise is typically due to the base spreading resistance, sometimes denoted r_b'.

The amount of thermal noise voltage can be computed as:

```
   e[n] = SQRT( 4 k T R B )

where:
   e[n] = Johnson noise voltage across R (rms volts)
   k = Boltzmann's constant (1.380662 * 10**-23 joule/deg K)
   T = temperature (deg. Kelvin)
   R = effective resistance (ohms)
   B = measurement bandwidth (Hz)
```

Or, for quick approximations at room temperature [Ryan, Al and Tim Scranton. 1984. D-C Amplifier Noise Revisited. *Analog Dialog.* 18(1): 3-11]:

```
   e[n] = 0.129 SQRT( R B )
   i[n] = 0.129 SQRT( B / R )

where:
   e[n] = Johnson noise voltage (rms uV) in series with R
   i[n] = Johnson noise current (rms pA) in parallel with R
   R = resistance (megohms)
   B = measurement bandwidth (Hz)
```

Also see: "Random Electrical Noise: A Literature Survey" (locally, or @: http://www.ciphersbyritter.com/RES/NOISE.HTM).

Thesis

In the study of logic, a proposition to be proven. Also see: hypothesis, analysis, synthesis and argumentation.

Threat

A particular and specific potential danger to security.

In cryptography, although we may speculate forever on the threats posed by our opponents and their capabilities and potential, we can not *know*, because those facts do not require our knowlege. This situation is inherent in cryptography and leads to unfounded belief in the strength of our ciphers.

Threat Model

1. In security analysis, the set of set of threats or attacks which can be used to break security.
2. What the system is designed to protect, from whom, and for how long. (Schneier, 1996, *Why Cryptography Is Harder Than It Looks*). (Also see attack tree.)

In a conventional security sense, threat models can be very important. They can provide a framework for discussing various possible attacks, and remind us of their consequences. They can provide a rationale for distributing resources to counter the most important threats. And they can remind us of the cost of failure. When we know all possible attacks, we can investigate their costs and our potential damage and probability of use. We can investigate countermeasures and *their* costs and probable advantages, in the context of our limited resources.

Threat models can be useful to address overall cryptosystems. They can highlight areas which must be secure even if not normally considered part of "the system." Threat models are particularly useful for protocols, which sometimes assume conditions which are not in fact true on modern data networks.

In cryptography, we normally intend to protect information from anyone, for the foreseeable future, under any conceivable effort. We assume the opponents have unlimited amounts of ciphertext and the associated plaintext (known plaintext) ciphered under a single key. Sometimes we can prevent those things at the cipher system level, but that does not really change the analysis, since unknown attacks are a real possibility.

With respect to direct attacks on ciphers, the conventional threat model is rarely useful:

- **The Threats.** We do not know and so cannot list all possible attacks on our ciphers. We rarely need consider even the range of attacks we know, since, if our opponents have *any* effective attack, we have failed. Rarely are countermeasures possible, except in cipher design itself. Nor can we assume that any new attacks will be more costly and less effective than the ones we know:
    - A new idea may lead to easier attacks.
    - An attack might be made into a computer program that almost anybody could run, independent of resources, knowledge, experience or age.
- **What we protect.** In cryptography, we assume that a cryptosystem protects our information, and failing that, exposes it all. Unless we have multiple such systems of different capabilities, we do not care what the information is, because all our information is protected the same way. If our cipher is broken, all protected information of the present, past and future may be exposed.
- **From whom.** In cryptography, we do not, and *can* not identify our opponents:
    - We cannot know who or how many they are.
    - We cannot know their training or experience.
    - We cannot know their abilities or resources.
    - We cannot know their motives or determination.
    - We cannot know when or how often they attack.
    - We cannot know whether they succeed.

  A successful attack can be made into a computer program, so the actual attacker may not need to know much, or invest much, to break the cipher. The possibility of attack programs for sale vastly expands the number and type of potential attackers over the unlimited future.
- **For how long.** Normally, cryptography cannot model strength. In most cases, if we can predict how long information will be secure, the system is already weak beyond any reasonable use.

Cryptography differs from physical security:

- We cannot tell if cryptography actually does prevent exposure of our information.
- We cannot tell if cryptography fails, or when, or how often, or to whom, or how much effort that took.

It should come as no surprise that the utility of any particular analytical technique may differ as well. With respect to cipher strength, a threat model is just not the panacea it is sometimes implied to be.

Time Constant

In electronics, in simple resistor and capacitor charge and discharge circuits, the product of R in ohms and C in farads. Alternately, in simple resistor and inductor (RL) circuits, the product of R in ohms and L in henrys. This result is the time in seconds for the current or voltage to fall to 1/e or 36.8 percent of it initial value, or to rise to (1 - 1/e) or 63.2 percent of its final value. (Here e is the base of natural logarithms, or about 2.718). Also see RC filter.

Trademark

Symbols or graphics or phrases used to identify goods or services as the product of a particular maker or provider. An intellectual property right.

Trade Secrecy

The right to keep formulas or inventions secret, and nevertheless profit from the resulting products or use. An intellectual property right.

In the U.S., trade secrecy law is a matter for the individual states. In general, if one creates something in secret, one is entitled to keep it secret, as in a secret beverage formula. No application is needed, nor are there any fees to pay. But there are consequences: First, trade secrets can be lost without legal recourse in various ways, apparently including:

- reverse engineering a product containing the secret,
- independent discovery, and
- accidental disclosure.

Furthermore, when an invention is kept a trade secret, that inventor gives up his right to that patent. But *other* inventors do not give up *their* rights, and unless the invention is published and thus becomes public prior art which would prevent a patent, some other inventor may obtain a patent which covers the original invention. Surely, neither the PTO nor the patent applicant could be criticized for not knowing that the invention exists, when the earlier inventor has explicitly chosen to keep it secret.

Traffic Analysis

The use of information *about* messages, such as call sign and length, to expose otherwise hidden information. Sometimes called a branch of cryptology.

While not strictly cryptanalysis, traffic analysis can reveal critical information even when cryptanalysis is ineffective. For example, when planning an offensive operation, orders from a headquarters (thus indicating the type of attack) might go to several facilities (thus indicating those involved). Similarly, messages with the exact same content and length might be sent to several stations, and thus facilitate either cryptanalysis or offensive cryptographic attacks.

Information of interest for traffic analysis includes:

- originating call sign (thus relating previous activity, location and suspected purpose)
- recipient call sign (thus implying a chain of command and perhaps a purpose)
- sending date and time
- frequency and/or addressing information
- message length
- message sequence number
- location of origin (e.g., radio direction-finding)

Trajectory

A path. A sequence of states in FSM state-space.

Transformer

In electronics, a passive component consisting of magnetically-coupled coils of wire. Each coil can be considered an inductor.

When an electrical current flows through a coil it creates a magnetic field, although a static magnetic field does not induce power. However, when a *changing* current flows through one coil or "primary," it creates a *changing* magnetic field that can be coupled to another coil or "secondary." A changing magnetic field induces power in a coil, producing a voltage proportional to the number of turns in that coil.

Normally, little power is lost from primary to secondary, and power is voltage times current. So secondaries with many turns can produce higher voltages than the primary, but then are necessarily limited to lower currents. Similarly, secondaries with few turns can produce lower voltages than the primary, and then can source higher currents. A transformer thus both *isolates* power or signal, and can change the voltage-to-current ratio, which we can describe as impedance.

Voltage Change

Probably the most common use for old-style "heavy iron" is to "step down" AC line voltage for low-voltage use in transistorized or digital logic equipment. Another use is to "step up" low voltages for high-voltage devices such as vacuum tubes. The ability to convert to and from high voltage at reduced current allows long-distance power distribution, since transmission loss is proportional to current squared.

Power Isolation

Almost all power transformers isolate the equipment from the AC power line, and especially the AC safety ground. One exception is "autotransformers" with a single winding, as often used in Variac variable-voltage transformers.

External isolation (1:1 turns ratio) transformers can be used to isolate entire systems from safety ground, ironically to increase safety or to break signal ground loops. Isolation transformers with a center-tapped secondary can provide balanced AC power which is sometimes beneficial in reducing hum. Isolation transformers also can prevent ground fault circuit interrupters from tripping when outside lights leak small amounts of current to physical ground after rain.

Audio Isolation

Audio isolation transformers can pass audio signals mostly unchanged while isolating power and signal grounds between different pieces of equipment. The issue is often preventing ground loops, a source of hum and crosstalk. However, real transformers work best in a band of frequencies, and have both a low and high frequency beyond which their response drops off.

Balanced Lines

A transformer secondary easily provides the opposing signals usually expected for balanced line operation. A transformer primary also provides the floating differential detection needed to take advantage of balanced lines. The ability to transfer relatively low-level signals on wires necessarily near to AC power is a mainstay of professional audio.

Impedance Match

The concept of impedance "matching" has created unnecessary confusion almost from the beginning of electronics. Typically there is little motivation to match impedances (see impedance matching). Audio signals normally are produced by low-impedance sources (e.g., preamplifier outputs) which are not matched to high-impedance inputs (e.g., amplifier inputs). AC power is available as a low impedance source and is not matched to equipment power impedance.

Sensitive electro-mechanical devices, such as phonograph cartridges and professional microphones, may need a particular load impedance to achieve design performance. Nowadays the known load is often provided by a single resistor across the high-impedance input of the amplifier, as opposed to selecting a transformer with a particular impedance ratio.

On the other hand, preamplifier designs using bipolar input transistors require a low impedance signal source for best noise performance. A low impedance allows the signal source to "shunt away" some of the noise produced inside the input device itself. Each bipolar circuit has an optimal source impedance for lowest noise, and transformers sometimes are used to achieve that, although their support for balanced line and ground-loop isolation are probably more important.

Audio Response and Distortion

Audio signals cover a wide ratio of frequencies, and transformers can have trouble at both ends. The reason for transformer low-frequency response roll-off is the inductance of the transformer primary winding. (Small "600 ohm" audio line transformers can have an inductance of 3.5 Henrys.) As signal frequency goes down, the inductive reactance of the primary winding also goes down. This reduced impedance increases the load on the amplifier or other signal source and so reduces the voltage at the transformer primary. As the voltage across the primary goes down, the secondary must be similarly affected.

It is common to test audio circuits with square waves, which can be a serious problem for transformers. The flat parts of a square wave are essentially DC, and transformers cannot work on DC. Moreover, the fast transient when the wave changes from one polarity to the other can cause "overshoot" due to ringing, the excitation of an LC resonance. Of course, revealing a resonance allows it to be fixed. But a reasonable general alternative to square waves is "triangle waves."

Transformer distortion typically is low for small signals, but if large signals saturate the magnetic material the distortion will be massive.

Bidirectional Operation

Transformers are "bidirectional" and will transport signals in two directions simultaneously. That can be useful in "hybrid" transformer arrangements, which allow full duplex conversations over a single line. This is done by forming a "bridge" circuit to "cancel" the known local signal thus exposing the unknown remote signal.

Transformer Impedance

Generally speaking, transformers have *ratios*, and not fixed values. The impedance ratio is the square of the turns ratio, but the actual primary impedance is set by the load on the secondary. However, any transformer must have enough primary inductance at the operating frequency so as to not load the signal source or generator. It would be nice if the listed transformer impedance was always the primary inductance at the lowest design frequency, but there is ample financial motive to fudge this value when low frequencies may be rare.

Generator impedance and transformer inductance form a RL high-pass filter with a particular low-frequency response. The transformer manufacturer thus specifies frequency response assuming a given signal source or generator impedance, which may be indicated by the specified primary impedance. Secondary impedances then follow from the turns ratios (squared).

Often, a transformer primary is driven from a low impedance, which should improve low-frequency transformer response, at least to the point where the lamination material finally saturates. The secondaries follow the lower generator impedance, thus showing that transformer impedances are not fixed.

Many audio transformers operate well with essentially no load on the secondary, but some do require a load resistance to damp down resonant peaks and valleys in their transformed response. Unfortunately, it probably is necessary to "sweep" a signal through each transformer design to see how smooth it is under various loads.

Magnetic Saturation

In any transformer with a magnetic core, whether radio frequency (RF) or audio frequency (AF), a high-enough voltage on the primary will saturate the magnetic circuit and cause an immediate loss of primary inductance. As a result, the primary impedance transiently falls to the DC resistance of the primary wire. During saturation, the generator or amplifier output is shorted, which typically causes massive distortion. Saturation occurs most easily on low-frequency signals.

Alternately, if the generator has a very low impedance, it will transiently have to produce massively more power to maintain the signal level when saturation occurs. Amplifier fuses could blow, but the output transistors usually go first. A better amplifier may protect itself, but only at the cost of massive distortion.

Magnetic or core saturation is the result of high primary voltage at relatively low frequency. Saturation is easy to test since no secondary load is needed. Put a small resistor in series with the primary and monitor the resistor voltage as signal voltage is increased or frequency decreased. Observe an abrupt increase in resistor voltage as saturation starts to occur on each cycle.

For power transformers, saturation and eventual melt-down are possible for 60Hz power transformers connected to 50Hz mains or slow emergency generators. A bitter cure for a hot 60Hz transformer might be to somehow reduce the voltage on the primary, but that will also reduce the output. For AC generators, faster than 60Hz is much better than slower.

For signal transformers, the effect of saturation on the generator can be reduced by adding some resistance between the generator and the transformer primary.

The design cure is to acquire a transformer with more primary and secondary turns or more heavy iron which can handle more power at a low frequency. Or low frequencies might be filtered out of the signal to allow the use of cheaper transformers.

Also see ratio transformer.

Transistor

An active semiconductor component which performs analog amplification. Typically a 3-terminal device with input (base), common (emitter), and output (collector) terminals.

History

The transistor was invented at AT&T Bell Labs in late 1947. At that time, amplifying devices in the form of vacuum tubes were very well known. Tubes were characterized by *transconductance*, the variation in output or plate current in response to input or grid voltage. But since the new solid-state device seemed to respond to input *current*, it needed to be described differently.

The first transistors were "point-contact" devices, and followed the general construction of existing point-contact semiconductor diodes. In the early diode, a tiny block of single-crystal germanium was probed by a sharp and delicate wire to make the anode contact. In the transistor, *two* wires were emitter and collector contacts. Sometimes the collector was "formed" by capacitor discharge, presumably causing localized melting and creating a larger and better contact. Often the junctions were protected by a grease. Since the germanium block was the base for the construction, that terminal was reasonably called the "base."

Point-contact transistors had low gain and poor frequency capabilities, and were mainly operated in "common base" mode. In that mode, the signal is sent into the emitter and out the collector, and, thus, *through* the device. An early description of a transistor was a device that would amplify signals transferred through it. One name proposed was "trans-*resistor*," which J. R. Pierce quickly contracted to "transistor." (J. R. Pierce is also known for his work with information theory.)

Modern transistors are usually made from silicon, and are usually operated in common-emitter or common-collector modes, instead of common-base.

Modern bipolar transistor models show that transistor operation is *best* described as a *voltage*-controlled current source, instead of a current-controlled current source. Unfortunately, the bipolar base-emitter junction has a low and nonlinear impedance, similar to a forward-biased diode, which can require substantial input current. This is much different from the vacuum tube or FET model, where the input or grid or gate voltage generally requires almost no current at all. That means one cannot model bipolar transistor operation by base voltage without also considering the curent required to force the base to that voltage. So, while the voltage-controlled model is more accurate, it is also much more complex than the simple idea of amplified base current.

Operation

In normal analog use, a transistor is biased "partly on" so the output rests at about half the maximum output voltage (see voltage divider). That allows the transistor to respond to both positive and negative parts of an AC signal, such as audio or video. However, designing biasing that works across a wide range of devices, voltages and temperatures can be tricky. (See transistor self-bias.)

There are two forms of bipolar transistor: PNP and NPN, named for the two possible sequences of three opposing Positive and Negative semiconductor layers. The terminals connecting to each layer are called Emitter (e), Collector (c), and Base (b). The simple model is that a current flow through the base-emitter junction (Ibe) allows an amplified current to flow through the collector-emitter junction (Ice). (A more accurate but much more complex model expresses bipolar collector current as a function of base-emitter *voltage*, see Ebers-Moll.)

In a sense, a bipolar transistor consists of two back-to-back PN diodes in a single crystal: the base-collector junction (operated in reverse bias) and the base-emitter junction (operated in forward bias). Current through the base-emitter junction moves either free electrons or free "holes" from the emitter into the base where they are physically near the collector junction. Many of these "carriers" are lost to the collector due to the higher potential there, and so do not count as base current. Indeed, the base is made thin and somewhat resistive so there is a high loss of carriers from the base material, thus increasing the ratio of collector current to base current, which is current gain and amplification. A modern bipolar small-signal transistor generally has a gain between 50 and 200.

Field-Effect Transistor (FET) devices have an extremely high input impedance, taking essentially no input current, and can be more easily fabricated in integrated circuits than bipolars. In an FET, Drain (d) and Source (s) contacts connect to a "doped" semiconductor *channel* which contains the output current flow. There are P-channel and N-channel devices.

MOSFET (metal-oxide-semiconductor FET) devices have a metal Gate (g) contact over the channel, insulated by a very thin oxide or glass layer. Voltage on the gate creates an electrostatic field which interacts with current flowing in the drain-source channel, and can act to turn that current ON or OFF, depending on channel material (P or N), doping (enhancement or depletion), and gate polarity. In most power devices, the source terminal is connected to the substrate and there is a "body diode" across the drain and source. MOSFET devices are typically "enhancement mode" and are normally OFF. N-channel enhancement devices go ON with a positive voltage (0.5 to 5v) on the gate. Depletion mode N-channel MOSFET devices are possible, and presumably would be normally ON and would need to be biased OFF with a negative voltage. Some depletion-mode examples exist in small-signal dual-gate VHF amplifiers, but depletion mode MOSFETS are not common.

JFET (junction-FET) devices have a gate which is a reverse-biased PN junction instead of an insulated conductor. Often, the drain and source terminals are interchangeable. JFET's can *only* be "depletion mode," which means they are normally ON, and the gate must be biased to turn them OFF. N-channel JFET devices go OFF with a negative voltage (e.g. -3v) on the gate.

Transistor Saturation

The electronic circuit situation where a transistor attempts to conduct more current than the load resistor will allow. As a result, the transistor is "fully ON," which is a lack of amplification. In this situation, small signals are completely lost, and large signals are distorted.

Transistor Self-Bias

A common approach used to bias or set the nominal output voltage of a transistor at "half on." Typically, the voltage generated by current flowing in an emitter resistor is automatically adjusted to match a fixed base voltage. This is essentially a constant current circuit with a load resistor. As opposed to collector-base feedback, and relatively exotic biasing options like op amp circuits. (While op amp biasing may seem like overkill for small-signal transistors, op amps are now very cheap, and in any case may make sense for power RF devices that are hard to bias efficiently.)

The main difficulty in establishing an output bias is that transistor devices with the exact same part number may have any of a wide range of current gains, as we can see from device data sheets. Two different devices actually may have the specified minimum and maximum gain values, and a circuit using those devices generally must work in either case.

Moreover, gain is specified at a particular temperature, and at higher temperatures gain may double, while at lower temperatures gain may halve. Any bias technique which presents a relatively fixed base current to the circuit will have difficulty controlling the output voltage across various devices and temperatures. Usually the problem is massively too much bias, resulting in transistor saturation and almost no amplification at all, but too little bias also will cause distortion.

Normally, a bipolar transistor is self-biased by placing a relatively low impedance low voltage on the base, and a resistor in the emitter. By varying the emitter resistor, an appropriate current can be selected which, working with the collector load resistor, places the output bias at about half the difference between the emitter and the supply. If greater device gain causes more current to flow, the voltage on the emitter resistor rises, which tends to cut off the base current, thus reducing device current. This negative feedback tends to reduce current variation.

Typically, the base bias voltage will be a volt or two; it must be greater than the typical 0.6 volt V_{BE} of the emitter junction, but every volt used for bias is a volt not available for output. Also, even though the voltage source must have *relatively* low impedance, that will tend to shunt the input signal, which then must be of substantially *lower* impedance. Usually the emitter resistor is "bypassed" with a capacitor so that bias feedback is delivered only at DC, and full gain is available at AC.

While field-effect devices do not have the sharp bipolar "knee" in the (input voltage vs. output current) transfer curve and so less tightly control the device current, they can be easier to use. Typically, an N-channel JFET gate is biased to ground (generally through a high value resistor so that signal is not lost). Since a JFET is normally ON, current rises in the source resistor until the gate goes negative with respect to the source voltage, and the JFET starts to turn OFF, thus controlling device current. Again, varying the source resistor effectively varies the regulated current (and the gain).

Transposition

The cryptographic strategy of changing message-element *positions*, instead of changing message-element *values*, as is done in substitution.

In mathematics, transposition is the exchange in position of two elements. This is the most primitive possible permutation or re-ordering of elements. Any possible permutation can be constructed from a sequence of transpositions. Also see shuffle, transposition cipher, braid and Dynamic Transposition.

Transposition Cipher

A form of block cipher which first accumulates data in a block, and then permutes or re-arranges elements within that block. The permutation is thus the key. In general, the larger the block, the more possible permutations, and so the larger the keyspace. Transposition ciphering is "dynamic" when each block is permuted separately and, with high probability, differently.

Transposition can be a particularly attractive encryption transformation. In contrast to the tiny fraction of simple substitution tables emulated by conventional block ciphering, transposition can make *every possible* permutation available to the shuffling sequence. And, since the per-byte encryption effort for transposition is largely independent of block size, transposition block size can vary on a block-by-block basis.

Arbitrary permutations are efficiently produced in a computer by the well-known Shuffle algorithm, driven by a keyed large-state random number generator. (See my article "A Keyed Shuffling System for Block Cipher Cryptography," either locally, or @: http://www.ciphersbyritter.com/KEYSHUF.HTM). Shuffling the same block twice helps hide the shuffling sequence from external analysis (see double shuffling). The RNG needs a large amount of state so that the shuffling sequence will be independent for the length of two shufflings. While unusual in statistical service, large-state RNG's are relatively easy to design and implement, and can be efficient in operation (see additive RNG).

Classic transposition ciphers are extremely limited, in that:

- only whole letters or characters are permuted;
- only simple hand-permutation patterns are used;
- the permutation is fixed for many messages; and
- messages of the same length are generally permuted similarly.

The classic attack on classic transposition is called multiple anagramming, but only works when two messages are permuted in the same way.

All of the classic limitations are eliminated in modern transposition cipher designs that:

- permute bits instead of characters (see bit permutation cipher);
- use computer algorithms that support every possible permutation (with a sequence generator having sufficient state);
- construct a new permutation for each message; so that
- different blocks or messages are almost never permuted similarly.

For security even when the block is all-1's or all-0's, the data can be bit-balanced before transposition (see Dynamic Transposition).

Also see

- my article: "Dynamic Transposition Revisited Again," locally, or @: http://www.ciphersbyritter.com/ARTS/DYNTRAGN.HTM
- the Dynamic Transposition Ciphering conversation, locally, or @: http://www.ciphersbyritter.com/NEWS5/REDYNTRN.HTM
- and a cipher taxonomy.

Trap Door

A cipher design feature, presumably planned, which allows the apparent strength of the design to be easily avoided by those who know the trick. Also see: back door.

Tree Analysis

A formal method of analysis which models some property in the form of a network tree. (See fault tree analysis and attack tree.)

Tree analysis addresses goals which can be achieved in not just one, but in multiple different ways. A network in the form of a tree is used, with goals represented as nodes. Various possible ways to achieve a particular goal are represented as branches, which then can be taken as goals with their own branch nodes. *Provided* that nodes can be evaluated, a total result can be accumulated to describe each approach. Then, *provided* that every possible approach can be identified, the optimal approach becomes apparent.

A tree structure can be used to model various properties, and can be evaluated by various functions. One possibility is to accumulate some real quantity across all daughter nodes, perhaps as a sum, or the minimum value or the maximum. Other possibilities may consider each node to have a Boolean value, such as "activated" versus "inactive." With Boolean values, one accumulation function is to require *all* daughter nodes to be active before a particular node will be active (essentially, the AND logic function). Alternately, *only one* daughter node may need be active to activate the parent (an OR logic function). Or each arc could have a different meaning. The utility of the result depends not upon mathematics (which obviously has many options), but instead the extent to which the model relates to reality.

Tree analysis has a very serious problem: A tree analysis does not tend to expose unknown alternatives. Not highlighting the presence of unknown attacks is a serious problem for cryptographic analysis and attack trees.

In some applications, one could expect that summed probability of all options should be exactly 1.0, or perhaps exactly the same as the parent node. Then, to the extent that accurate measurements are available, it might become possible to infer that one or more alternatives exist which are not represented in the model. But even that would not help to specifically identify or describe those alternatives.

Trinomial

An expression with three terms, each a different power of some variable. Also see: monomial, and polynomial.

Triple DES

The multiple encryption consisting of three sequential cipherings with the block cipher DES, each time with an independent key. Often implemented as "EDE," or encipher-decipher-encipher (also see Pure Cipher). Despite using three keys, the true strength is argued to be more like two DES keys, or 112 bits. Keyspace is only one aspect of strength, and 112 bits is more than enough to discourage brute force attacks.

TRNG

True RNG or Truly Random Number Generator. (Presumably, "truly random" was taken from *Knuth II*, Section 3.5.) A non-deterministic really-random or physically-random RNG. As opposed to PRNG.

Trojan Horse

A deceptive program which receives privileges based on a false apparent role; those privileges are then misused.

Truly Random

A random value or sequence derived from a physical source. Also called really random and physically random; see: TRNG.

Trust

1. In a security context, confidence in the future based on negative consequences for abuse, misuse or betrayal. This is "Trust, but verify." This sort of trust is not mere unsupported belief, but is instead the knowledge that betrayal will cost the betrayer dearly. Without consequences, there is no secure basis for trust.
2. Confidence in the future based on past experience. That of course assumes that the past is correctly known and the thing being trusted is not deliberately deceptive. It also assumes that external conditions generally will not change (otherwise the response could not be predicted), an assumption which probably must fail eventually.
3. Unsupported belief in a particular outcome, in a dependence upon some one or some thing. Mere hopes and dreams.

In a true security sense, it is impossible to fully trust *anyone:* Everyone has their weaknesses, their oversights, their own agendas. But normally "trust" involves some form of *commitment* by the other party to keep any secrets that occur. Normally the other party is constrained in some way, either by their own self-interest, or by contractual, legal, or other consequences of the failure of trust. The idea that there can be any realistic trust between two people who have never met, are not related, have no close friends in common, are not in the same employ, and are not contractually bound, can be a very dangerous delusion. It is important to recognize that no trust is without limit, and those limits are precisely the commitment of the other party, bolstered by the probability of detection and the consequences of betrayal. Trust without consequences is necessarily a very weak trust indeed. (Also see proof.)

Trust in Secure Communications

Trust is the basis for communications secrecy: While secrecy can involve keeping one's own secrets, *communications security* almost inevitably involves at least a second party and the equipment itself. We thus necessarily "trust" that party with the secret, as well as cryptographic keys. It makes little sense to talk about secrecy in the absence of trust.

In communications security, the trust we have is that the other user will keep our secrets. We need not necessarily trust their ethics, morals, judgment, loyalty, or the quality of information they deliver. The trust needed by a security system is simply the trust to not violate security.

Trusting Computers and Ciphers

In security discussions, it is common to state that a particular computer is "trusted." But computers make only those decisions which have been programmed for them, cannot experience consequences, and are not themselves responsible. Since designers and implementors rarely experience consequences for design failure, consequences are not a basis for trusting the design.

Sometimes a computer is "trusted" in the sense of expecting a machine to behave in the future as it did in the past. That of course depends upon knowing exactly how the machine actually did perform, and that outside conditions will remain unchanged, which seems rather unlikely.

Computing hardware can be "trusted" to work if it was designed for test, and was exhaustively tested, which is rarely the case. Naturally, tests must be repeated periodically, because hardware can die. Abstractly, computer software could be trusted similarly, but only if all software in the system was designed for test and actually was exhaustively tested, which is almost never the case. Current computing systems are far, far too complex for exhaustive tests on the unit as a whole.

Other times a computer is said to be "trusted," with trust being taken as a mere marker, an indicator that the machine is inside the defenses and so is trusted by default. The idea that one can trust everything which happens to be inside our defenses deserves no further comment.

It is not possible to trust a cipher based on past performance, because it is not possible to know whether a cipher has been successful at keeping information secret from our unknown opponents. See scientific method.

Trusting Sources

Another aspect of trust in cryptography is the ability to trust technical pronouncements made in texts or in public statements. Most users simply do not themselves have the background to verify such comments. However, users should know that some texts do appear to slant issues in ways which glorify academic cryptography and minimize real problems. It is important that someone actually go to the source material and verify that it actually says what a summary text says it says.

In addition, there are academic papers whose titles imply results different from those actually found. It is important that someone go through the reasoning and verify that it means what it says it means, and that the results apply to a reasonable practical environment. (Also see proof.) Pretty much the only way for non-experts to do that, either for articles or claims, is to review the arguments made by those who assert that they know the issue. Thus, the concern is not the result *per se,* but the correctness of the argument, because it does not take an expert in cryptography to expose false argumentation.

In cryptography, the majority view is often incorrect (see, for example, Old Wives' Tale). My guess at the reason for this is that someone investing a lot of time and effort in cryptography has "bought in" to the accepted position and is unwilling to "make waves." (Also see cognitive dissonance.) Perhaps the largest such issue is that of cipher strength, which is not known scientifically, and is simply believed by sellers and users. Note that reality is irrelevant to selling the product! And, of course, some fraction of those in cryptography may have a professional interest in misleading everyone about cipher strength. It must not be forgotten that deception, misdirection, propaganda and conspiracy are natural and expected aspects of cryptography.

Truth Table

Typically, a Boolean function expressed as the explicit value it will produce for each possible combination of input values, expressed in order. See the examples in logic function.

Tunneling

1. In semiconductor electronics, the ability of electrons to traverse a potential barrier without needing to surmount that barrier, due to quantum-mechanical effects when the barrier is extremely thin.
2. In Internet communications, establishing a secure encrypted connection which is then used for the IP data stream.

Two-Sided Test

1. In statistics, a two-tailed test.
2. In statistics, a statistic computation sensitive to variation in both directions (e.g., "above" or "below"). See Kolmogorov-Smirnov. As opposed to one-sided test.

Ideally, "two-sided tests" are statistic computations sensitive to variations on both sides of the reference. The two "sides" are not the two ends of a statistic distribution, but instead are the two directions that sampled values may differ from the reference (i.e., *above* and *below*).

Using a "two-sided" test does *not* mean that a low p-value represents "below" and a high p-value represents "above" or vise versa: Instead, finding low p-values generally means that the sampled distribution is "too good to believe."

If we want to expose deviations *both* above *and* below the reference, we can use *two* appropriate one-sided tests, *or* a "two-sided" test intended to expose both differences. Then, finding high p-values generally implies that the statistic has detected "something unusual," something beyond the null hypothesis. With a two-sided test, this would mean that some difference, positive or negative, probably was found between the sampled distribution and the reference distribution.

Two-Tailed Test

In statistics, a hypothesis evaluation with a rejection region on both ends of the null distribution. The "test" part of this is the two critical values which mark the start of each rejection region, thus becoming the measure of the hypothesis. Sometimes called two-sided. In contrast to one-tailed test. Also see null hypothesis.

Fundamentally a way to *interpret* a statistical result. *Any* statistic value can be evaluated at either tail of the null distribution or even both tails simultaneously. "Two-tailed" evaluations use both tails, but each tail has a different meaning.

When comparing distributions, in general, repeated p-values near 0.0 generally mean that the distributions seem too similar, which could indicate some sort of problem with the experiment.

On the other hand, the meaning of repeated p-values near 1.0 depends on the test. Some one-sided tests may concentrate on sampled values *above* the reference distribution, whereas different "one sided" tests may be concerned with sampled values *below* the reference. If we want to expose deviations *both* above *and* below the reference, we can use *two* appropriate "one-sided" tests, *or* a two-sided test intended to expose both kinds of difference.

Both the chi-square test and the "two-sided" Kolmogorov-Smirnov (K-S) test detect both "above" and "below" differences between distributions. Alternately, we could just as well run *both* "one-sided" K-S tests and get the same information.

"Two-tailed" evaluations have "critical values" which mark off the "too-similar" critical region on one tail of the null distribution and the "too-different" critical region on the other. One problem with this is that the two tails of the distribution generally have very different experimental worth, yet, typically, the same amount of critical region is allocated to each tail. Thus, one may well question whether the usual "two-tailed" interpretation is appropriate for a particular experiment.

Type I Error

In statistics, the rejection of the null hypothesis even though it is actually correct. This can occur strictly by chance. Since even extreme statistic values do occur occasionally with normal random data, if we can afford multiple trials, we may want to have multiple rejections before we believe them. The probability of rejecting the null hypothesis by chance alone is the significance of the test. Also see Type II error and power.

Type II Error

In statistics, the acceptance of a null hypothesis which is actually false. Also see Type I error and power.

UFN

Unbalanced Feistel Network.

Unary

From the Latin for "one kind." Sometimes used to describe functions with a single argument, such as "the unary -" (the minus-sign), as opposed to subtraction, which presumably would be "binary," and *that* could get very confusing very fast. Thus, monadic may be a better choice. Also see: arity, binary and dyadic.

Unbalanced Feistel Network

(UFN). "Feistel networks where the source and target blocks are of different size." (Schneier) See Feistel construction.

Uncertainty

The extent to which a value is not known as an absolute fact. A base concept for cryptographic unknowability, when a value not only is not known, but also cannot be predicted (unpredictability). Also see entropy.

Unexpected Distance

My name for the values computed by a fast Walsh transform (FWT) when calculating Boolean function nonlinearity as often used in S-box analysis.

Given any two random Boolean sequences of the same length, we "expect" to find about half of the bits the same, and about half different. This means that the *expected* Hamming distance between two sequences is half their length.

With respect to Boolean function nonlinearity, the expected distance is not only what we *expect,* it is also *the best we can possibly do,* because each affine Boolean function comes in both complemented and uncomplemented versions. So if *more* than half the bits differ between a random function and one version, then *less* than half must differ to the other version. This makes the expected distance the ideal reference point for nonlinearity.

Since the FWT automatically produces the difference between the expected distance and the distance to each possible affine Boolean function (of the given length), I call this the *un*expected distance. Each term is positive or negative, depending on which version is more correlated to the given sequence, and the absolute value of this is a measure of linearity. But since we generally want *non*linearity, we typically subtract the unexpected value from half the length of the sequence.

Unicity Distance

The amount of ciphertext needed to uniquely identify the correct key and its associated plaintext (assuming a ciphertext-only attack and natural language plaintext). With less ciphertext than the unicity distance, multiple keys may produce decipherings which are each plausible messages, although only one of these would be the correct solution. As we increase the amount of ciphertext, many formerly-plausable keys are eliminated, because the plaintext they produce becomes identifiably different from the structure and redundancy we expect in a natural language.

> "If a secrecy system with a finite key is used, and *N* letters of cryptogram intercepted, there will be, for the enemy, a certain set of messages with certain probabilities, that this cryptogram could represent. As *N* increases the field usually narrows down until eventually there is a unique 'solution' to the cryptogram; one message with probability essentially unity while all others are practically zero. A quantity *H(N)* is defined, called the equivocation, which measures in a statistical way how near the average cryptogram of *N* letters is to a unique solution; that is, how uncertain the enemy is of the original message after intercepting a cryptogram of *N* letters." [p.659]
>
> "This gives a way of calculating approximately how much intercepted material is required to obtain a solution to the secrecy system. It appears . . . that with ordinary languages and the usual types of ciphers (not codes) this 'unicity distance' is approximately *H(K)/D.* Here *H(K)* is a number measuring the 'size' of the key space. If all keys are *a priori* equally likely, *H(K)* is the logarithm of the number of possible keys. *D* is the redundancy of the language . . . ." "In simple substitution with a random key *H(K)* is $\log_{10}$ 26! or about 20 and *D* (in decimal digits per letter) is about .7 for English. Thus unicity occurs at about 30 letters." [p.660]

"After the unicity point has been passed in intercepted material there will usually be a unique solution to the cryptogram. The problem of isolating this single solution with high probability is the problem of cryptanalysis." [p.702]

-- Shannon, C. 1949. Communication Theory of Secrecy Systems. *Bell System Technical Journal.* 28:656-715.

Also see: Ideal Secrecy and data compression.

Uniform Distribution

A probability distribution in which each possible value is equally likely. Also a flat or even distribution.

A uniform distribution is the most important distribution in cryptography. For example, a cryptographer strives to make every possible plaintext an equally likely interpretation of any ciphertext (see Ideal Secrecy and balance). A cryptographer also strives to make every possible key equally likely, given any amount of known plaintext.

On the other hand, a uniform distribution with respect to one quality is not necessarily uniform with respect to another. For example, while keyed random shuffling can provably produce any possible permutation with equal probability (a uniform distribution of different tables), those tables will have a Boolean function nonlinearity distribution which is decidedly not uniform. And we might well expect a *different* non-uniform distribution for every different quality we measure.

Universe

In statistics, the set of all possible relevant values. Also see alphabet and population.

Unknowable

A value which not only is not known (uncertain), but is also unpredictable by any means. See entropy.

Unpredictable

1. Something which cannot be anticipated. The supposed impossibility of creating a scientific model which can predict "future" results. The essence of cryptography. As opposed to predictable.
2. An alternate, confusing, and I believe inappropriate, meaning of entropy in cryptography. Also see unknowable, hypothesis and random.

Note that unpredictability simultaneously requires a high entropy as in *coding efficiency*, and also a high amount of unknowable information. It is common in really random generators to have some amount of unknowable information in a *non*-uniform distribution. As a result, such values will be somewhat predictable, if only to the extent that the coding is inefficient and redundant.

User Authentication

Assurance that a user is who he claims to be.

The classic approach to user authentication is a password; this is "something you know." One can also make use of "something you have" (such as a secure ID card), or "something you are" (biometrics).

The classic problem with passwords is that they must be remembered by ordinary people, and so carry a limited amount of uniqueness. Easy-to-remember passwords are often common language phrases, and so often fall to a dictionary attack. More modern approaches involve using a Diffie-Hellman key exchange, *plus* the password, thus minimizing exposure to a dictionary attack. This does require a program on the user end, however.

Also see: message authentication and key authentication.

Variable

In mathematics and computing, a symbol able to represent any element or value in some range or set. As opposed to a constant. Also see: expression and equation.

Variable Size Block Cipher

The ciphering concept described in U.S. Patent 5,727,062, see

- U.S. Patent 5,727,062 locally or @: http://www.ciphersbyritter.com/PATS/VSBCPAT.HTM
- the VSBC info and articles section on the main page: locally, or @: http://www.ciphersbyritter.com/index.html#VSBCTech

It appears that I originated the term "Variable Size Block Cipher" and introduced it to cryptography in my 1995 Aug 20 article:

- "VSBC Newsgroup Discussion," locally, or @: http://www.ciphersbyritter.com/NEWS/VSBCNEWS.HTM

A VSBC is a block cipher which supports ciphering in blocks of dynamically variable size. The block size may vary only in steps of some element size (for example, a byte), but blocks could be arbitrarily large.

Three characteristics distinguish a true variable size block cipher from designs which are merely imprecise about the size of block or element they support or the degree to which they support overall diffusion:

1. A variable size block cipher is indefinitely extensible and has no theoretical block size limitation;
2. A variable size block cipher can approach overall diffusion, such that each bit in the output block is a function of every bit in the input block; and
3. A true variable size block cipher does not require additional steps (rounds) or layers to approach overall diffusion as the block size is expanded.

Also see Dynamic Substitution Combiner, Dynamic Transposition, Balanced Block Mixing, Mixing Cipher and Mixing Cipher design strategy.

Variance

A descriptive statistic which attempts to capture the extent to which data vary around the sample mean or some other basis value. The common variance is the sum of the squares of each deviation from the basis value. The variance is the square of the standard deviation.

A related computation is the "mean deviation" or "absolute deviation" which is the sum of the absolute values of each deviation from the mean.

In contrast, the basis for Allan Variance is the value of the previous sample. One advantage of this is that the mean does not have to be computed (a summation across the entire sample) before variance computation can start.

VCO

Voltage controlled oscillator.

Vector

1. In software: A sequence of values, an array.
2. In mathematics: A value with both magnitude and angle; the polar form of a complex number. As opposed to: scalar.

VENONA

The NSA codename for a project which broke a hand cipher system used between the Russian KGB or GRU and their operatives in the United States from 1939 to 1946. That cipher system included a one time pad component, which is often considered to be proven unbreakable. The security failure of that system resulted in the *death by execution* of Julius and Ethel Rosenberg. VENONA has its own pages at http://www.nsa.gov/docs/venona/.

Vernam Cipher

The original stream cipher. Described by Gilbert S. Vernam in U.S. Patent 1,310,719 of July 22, 1919. Based on combining data with a confusion loop both in punched paper tape (teletype) form, the early form of machine stream cipher. (Interestingly, neither of the modern combining terms exclusive-OR or "mod-2 addition" are used in the patent.) Soon generalized to use as much confusion as data, which is the one-time pad.

Vet

To check and approve.

Voltage

The measure of electron "potential" in volts. Voltage is analogous to water *pressure,* as opposed to *flow* or current.

Voltage Controlled Oscillator

An oscillator in which the frequency can be controlled over some range by varying a control voltage.

Voltage Divider

In electronics, the principle that voltage divides proportionally across a resistance. So if a voltage is applied to two resistors of the same value in series, each sees half the total voltage.

The obvious application is in a potentiometer or mechanical "volume control," where a sliding tap moves up or down a fixed resistance.

More generally, most forms of amplifier use some form of active device (such as a transistor) as one of the resistors. By varying the effective resistance dynamically, the output voltage changes proportionally.

Even more generally, many filters use the impedance in frequency-varying components, such as inductors or capacitors, to divide the applied voltage at the right frequencies to give the desired response.

Vulnerability

A security weakness that an opponent could exploit by attack.

Walsh Functions

Walsh Functions are essentially the affine Boolean functions, although they are often represented with values {+1,-1}. There are three different canonical orderings for these functions. The worth of these functions largely rests on their being a complete set of orthogonal functions. This allows any function to be represented as a correlation to each of the Walsh functions. This is a transform into an alternate basis which may be more useful for analysis or construction.

Also see: fast Walsh transform.

Walsh-Hadamard Transform

(WHT). See: fast Walsh transform.

Weak Key

For a particular cipher design, a key value which provides poor security

One example might be a substitution cipher with the substitution table in numerical order, so that the plaintext is not changed at all. A similar example would be a transposition cipher in which all of the plaintext elements are left in their original position.

Weight

The weight of Boolean Function *f* is the number of 1's in the truth table of *f*.

Whitening

An overly-cute description of making a signal or data more like white noise, with an equal amount of energy in each frequency. To make data more random-like or balanced. Examples include:

- CBC where plaintext data are mixed with the preceeding ciphertext block, and
- LFSR's which produce random-like sequences to be mixed with data, which is approaching the concept of a stream cipher.
- Various forms of digital-filter-like scrambler.

White Noise

A random-like signal with a flat frequency spectrum, in which each frequency has the same magnitude. As opposed to pink noise, in which the frequency spectrum drops off with frequency. White noise is analogous to white light, which contains every possible color. Also see noise and the usual white noise sources thermal noise and shot noise.

White noise is normally described as a relative power density in volts squared per hertz. White noise power varies directly with bandwidth, so white noise would have twice as much power in the next higher octave as in the current one. The introduction of a white noise audio signal can destroy high-frequency loudspeakers.

WHT

Walsh-Hadamard transform.

Wide Trail Strategy

A cipher design strategy described in Joan Daemen's thesis:

> "In the wide trail strategy, the resistance against LC and DC is achieved by the iterated alternation of a nonlinear transformation and a transformation with high diffusion."
>
> -- Daemen, J. 1995. Cipher and Hash Function Design, Strategies Based on Linear and Differential Cryptanalysis. Thesis. Section 6.8.

This strategy produces a cipher with repeated rounds, which are necessary when the diffusion or mixing is biased or unbalanced. Because the mixing forms which Daemen considers are not guaranteed ideal, he must introduce various concepts and computations which bound mixing bias (e.g., branch number).

Presumably the advantage the "wide trail strategy" is to have a structure in which the computations bound Differential and Linear attacks to impractical levels. But of course there is no implication that the strategy addresses all possible attacks. And simply having keyed S-boxes prevents most Linear and Differential attacks, as does limiting the amount of data ciphered under one key, as does multiple encryption.

One might argue that my Fenced DES design (April 17, 1994) (locally, or @: http://www.ciphersbyritter.com/NEWS/94042901.HTM) fits the general "wide trail" scheme of alternating nonlinear transformations (my keyed S-boxes) with wide diffusion (my FFT-style linear mixing), but does so in several unique layers, instead of repeated rounds.

In comparison, my current mixing cipher design strategy has moved on, and uses keyed FFT-style Balanced Block Mixing (the butterfly operations are orthogonal Latin squares) and is distinguished from the "wide trail" scheme by:

- the use of layers instead of rounds,
- a nonlinear and keyed diffusion transformation,
- a diffusion guarantee (FFT-like connection patterns have exactly one path from input element to output element, and any input change of any sort to any one element will change all output elements), and
- an equal-power mixing guarantee (stepping any one input element through all possible values steps every output element through all possible values, so all inputs have the same amount of effect).
- Because of these guarantees of diffusion and power, every mixing (typically nonlinear and key-selected) will have the same overall effect as any other, so
- there is usually no advantage to selecting particular mixing functions because the mixing performance is already guaranteed, and
- the mixing and diffusion can be easily understood without complex analysis. Moreover,
- whatever analysis is performed generally applies to all possible mixings,
- thus providing an analytical basis for keying the mixing itself.

Wire

A thin, long conductor, often considered "ideally conductive" compared to other parts of a circuit.

Work Characteristic

A measure of practical secrecy:

> "[When it is] possible in principle to determine . . . solutions (by trial of each possible key for example), different enciphering systems show a wide variation in the amount of work required. The average amount of work to determine the key for a cryptogram of *N* letters, *W*(*N*), measured say in man hours, may be called the work characteristic of the system. This average is taken over all messages and all keys with their appropriate probabilities. The function *W*(*N*) is a measure of the amount of "practical secrecy" afforded by the system."
>
> -- Shannon, C. E. 1949. Communication Theory of Secrecy Systems. *Bell System Technical Journal.* 28:656-715.

XOR

The widely-used assembly language mnemonic for the exclusive-OR function as a computer operation or opcode. Exclusive-OR is basically a Boolean logic function which is bit-level addition without carry. Normally, however, a computer will do a full word-width of exclusive-OR's, instead of operating just one bit at a time. Also called: "addition mod 2."

The exclusive-OR function is the simple but strengthless additive combiner of a conventional stream cipher. Exclusive-OR is also used extensively in block ciphers and, indeed, throughout cryptography.

The exclusive-OR function can be generalized beyond simple 2-value bit-elements to elements of arbitrary range in the Latin square combiner. The Ls combiner supports a huge number of unique, nonlinear, combining tables, and, thus, also supports combiner keying, which is simply not available with exclusive-OR.

XOR Encryption

Usually, the simple stream cipher produced by XOR-combining *plaintext data* with a *keystream*. Commonly, the keystream is produced by a keyed random number generator. There is also a toy version which uses a short string of text or random values as the keystream. Both versions become insecure as soon as the keystream starts to repeat, so the toy version is insecure almost immediately.

In cryptanalysis we normally assume that the opponent has a substantial amount of both the original plaintext and the corresponding ciphertext (known plaintext). In that case, simply combining the plaintext and ciphertext in another XOR exposes the original keystream (a known plaintext attack). That means the keystream generator can be attacked directly, which implies that XOR has not protected the generator at all. Alternative combiners include a keyed Latin square or Dynamic Substitution.

Z

In math notation, the integers.

Zener Breakdown

In semiconductor electronics, a diode reverse-voltage breakdown mode. A source of shot noise. In contrast to the more common avalanche multiplication breakdown mode, true Zener breakdown is typically confined to voltage regulator diodes specifically designed to break down at applied voltages under 5 or 6 volts.

In diodes designed for low-voltage breakdown, heavy doping creates a particularly thin space-charge layer between P and N materials. Thus, even a modest applied voltage can create a field on the order of 10^6 volts/cm, which is enough to pull apart covalent bonds in the crystal lattice, causing current flow. In Zener junctions, the space-charge region is too thin to allow the charge carrier growth found in avalanche multiplication.

Apparently, every Zener diode has a *range* of discrete breakdown voltages, each at a separate physical site, each capable of conducting some limited small amount of current. Increasing the voltage across the diode causes more sites to break down, which draws more current. Different diodes have different distributions of breakdown voltage, current, and average voltage difference between sites. See

- "Measuring Junction Noise" locally, or @: http://www.ciphersbyritter.com/RADELECT/MEASNOIS/MEASNOIS.HTM
- "Junction Noise Measurements I" locally, or @: http://www.ciphersbyritter.com/RADELECT/MEASNOIS/NOISMEA1.HTM

In contrast to avalanche multiplication, which has a positive temperature coefficient, Zener breakdown has a negative temperature coefficient. Presumably this is due to heat causing increased activity in the crystal lattice, thus increasing the probability of breaking a bond under an applied potential.

Also see:

- "Random Electrical Noise: A Literature Survey" locally, or @: http://www.ciphersbyritter.com/RES/NOISE.HTM

Zeroize

The military term for erasing a key from equipment.

Z_n

In math notation, the ring of integers modulo n.

Z_n^*

In math notation, the multiplicative group of integers modulo n.

(Z,+)

In math notation, the group of integers under addition.

(Z,+,*)

In math notation, the ring of integers under addition and multiplication.

Z/n

In math notation, the ring of integers modulo n.

Z/p

In math notation, the field of integers modulo prime p.

Z/pZ

Same as Z/p.

(Z/pZ)[x]

In math notation, the ring of polynomials in x with coefficients in the field **Z/pZ**.

Terry Ritter, his current address, and his top page.