Network Working Group
OBSOLETES RFC 1750

Expires June 2004

Donald E. Eastlake, 3rd
Jeffrey I. Schiller
Steve Crocker
December 2003

Randomness Requirements for Security

<draft-eastlake-randomness2-05.txt>

Status of This Document

Abstract

Security systems today are built on strong cryptographic algorithms that foil pattern analysis attempts. However, the security of these systems is dependent on generating secret quantities for passwords, cryptographic keys, and similar quantities. The use of pseudo-random processes to generate secret quantities can result in pseudo-security. The sophisticated attacker of these security systems may find it easier to reproduce the environment that produced the secret quantities, searching the resulting small set of possibilities, than to locate the quantities in the whole of the potential number space.

Choosing random quantities to foil a resourceful and motivated adversary is surprisingly difficult. This document points out many pitfalls in using traditional pseudo-random number generation techniques for choosing such quantities. It recommends the use of truly random hardware techniques and shows that the existing hardware on many systems can be used for this purpose. It provides suggestions to ameliorate the problem when a hardware solution is not available. And it gives examples of how large such quantities need to be for some applications.

Acknowledgements

Special thanks to
(1) The authors of "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security" which is incorporated as Appendix A.

(2) Peter Gutmann who has permitted the incorporation into this replacement for RFC 1750 of material from is paper "Software Generation of Practially Strong Random Numbers".

The following other persons (in alphabetic order) contributed to this document:

Tony Hansen, Sandy Harris

The following persons (in alphabetic order) contributed to RFC 1750, the predeceasor of this document:

David M. Balenson, Don T. Davis, Carl Ellison, Marc Horowitz, Christian Huitema, Charlie Kaufman, Steve Kent, Hal Murray, Neil Haller, Richard Pitkin, Tim Redmond, and Doug Tygar.

Table of Contents

More Table of Contents

1. Introduction

Software cryptography is coming into wider use and is continuing to spread, although there is a long way to go until it becomes pervasive.

Systems like SSH, IPSEC, TLS, S/MIME, PGP, DNSSEC, Kerberos, etc. are maturing and becoming a part of the network landscape [SSH, DNSSEC, IPSEC, MAIL*, TLS]. By comparison, when the previous version of this document [RFC 1750] was issued in 1994, about the only Internet cryptographic security specification in the IETF was the Privacy Enhanced Mail protocol [MAIL PEM].

These systems provide substantial protection against snooping and spoofing. However, there is a potential flaw. At the heart of all

cryptographic systems is the generation of secret, unguessable (i.e., random) numbers.

For the present, the lack of generally available facilities for generating such unpredictable numbers is an open wound in the design of cryptographic software. For the software developer who wants to build a key or password generation procedure that runs on a wide range of hardware, the only safe strategy so far has been to force the local installation to supply a suitable routine to generate random numbers. To say the least, this is an awkward, error-prone and unpalatable solution.

It is important to keep in mind that the requirement is for data that an adversary has a very low probability of guessing or determining. This can easily fail if pseudo-random data is used which only meets traditional statistical tests for randomness or which is based on limited range sources, such as clocks. Frequently such random quantities are determinable by an adversary searching through an embarrassingly small space of possibilities.

This Best Current Practice describes techniques for producing random quantities that will be resistant to such attack. It recommends that future systems include hardware random number generation or provide access to existing hardware that can be used for this purpose. It suggests methods for use if such hardware is not available. And it gives some estimates of the number of random bits required for sample applications.

2. Requirements

A commonly encountered randomness requirement today is the user password. This is usually a simple character string. Obviously, if a password can be guessed, it does not provide security. (For re-usable passwords, it is desirable that users be able to remember the password. This may make it advisable to use pronounceable character

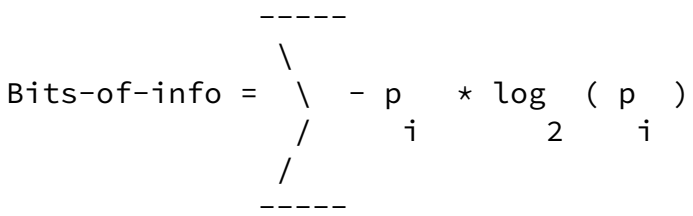
strings or phrases composed on ordinary words. But this only affects the format of the password information, not the requirement that the password be very hard to guess.)

Many other requirements come from the cryptographic arena. Cryptographic techniques can be used to provide a variety of services including confidentiality and authentication. Such services are based on quantities, traditionally called "keys", that are unknown to and unguessable by an adversary.

In some cases, such as the use of symmetric encryption with the one time pads [CRYPTO*] or the US Data Encryption Standard [DES] or Advanced Encryption Standard [AES], the parties who wish to communicate confidentially and/or with authentication must all know the same secret key. In other cases, using what are called asymmetric or "public key" cryptographic techniques, keys come in pairs. One key of the pair is private and must be kept secret by one party, the other is public and can be published to the world. It is computationally infeasible to determine the private key from the public key and knowledge of the public is of no help to an adversary. [ASYMMETRIC, CRYPTO*]

The frequency and volume of the requirement for random quantities differs greatly for different cryptographic systems. Using pure RSA [CRYPTO*], random quantities are required when the key pair is generated, but thereafter any number of messages can be signed without a further need for randomness. The public key Digital Signature Algorithm devised by the US National Institute of Standards and Technology (NIST) requires good random numbers for each signature [DSS]. And encrypting with a one time pad, in principle the strongest possible encryption technique, requires a volume of randomness equal to all the messages to be processed [CRYPTO*].

In most of these cases, an adversary can try to determine the "secret" key by trial and error. (This is possible as long as the key is enough smaller than the message that the correct key can be uniquely identified.) The probability of an adversary succeeding at this must be made acceptably low, depending on the particular application. The size of the space the adversary must search is related to the amount of key "information" present in the information theoretic sense [SHANNON]. This depends on the number of different secret values possible and the probability of each value as follows:



where i counts from 1 to the number of possible secret values and p sub i is the probability of the value numbered i. (Since p sub i is less than one, the log will be negative so each term in the sum will be non-negative.)

If there are 2^n different values of equal probability, then n bits of information are present and an adversary would, on the average, have to try half of the values, or 2^(n-1) , before guessing the secret quantity. If the probability of different values is unequal, then there is less information present and fewer guesses will, on average, be required by an adversary. In particular, any values that the adversary can know are impossible, or are of low probability, can be initially ignored by an adversary, who will search through the more probable values first.

For example, consider a cryptographic system that uses 128 bit keys. If these 128 bit keys are derived by using a fixed pseudo-random number generator that is seeded with an 8 bit seed, then an adversary needs to search through only 256 keys (by running the pseudo-random number generator with every possible seed), not the 2^128 keys that may at first appear to be the case. Only 8 bits of "information" are in these 128 bit keys.

3. Traditional Pseudo-Random Sequences

Most traditional sources of random numbers use deterministic sources of "pseudo-random" numbers. These typically start with a "seed" quantity and use numeric or logical operations to produce a sequence of values.

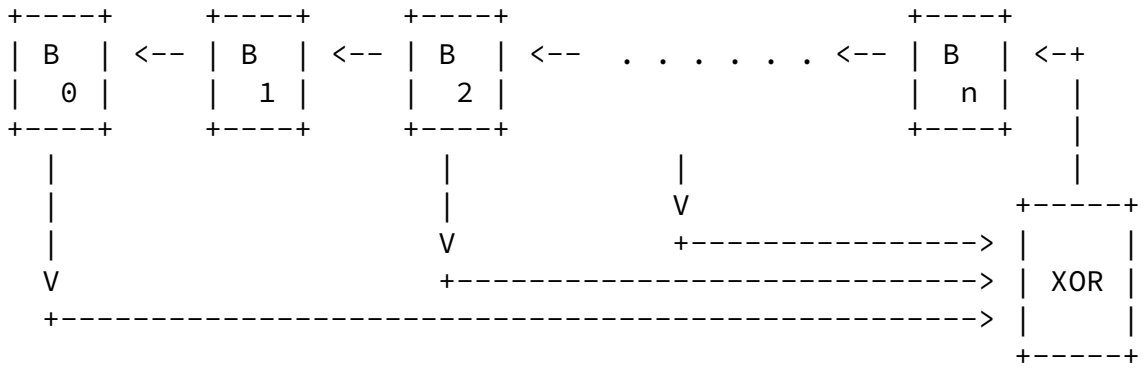
[KNUTH] has a classic exposition on pseudo-random numbers. Applications he mentions are simulation of natural phenomena, sampling, numerical analysis, testing computer programs, decision making, and games. None of these have the same characteristics as the sort of security uses we are talking about. Only in the last two could there be an adversary trying to find the random quantity. However, in these cases, the adversary normally has only a single chance to use a guessed value. In guessing passwords or attempting to break an encryption scheme, the adversary normally has many, perhaps unlimited, chances at guessing the correct value because they can store the message they are trying to break and repeatedly attack it. They should also be assumed to be aided by a computer.

For testing the "randomness" of numbers, Knuth suggests a variety of measures including statistical and spectral. These tests check things like autocorrelation between different parts of a "random" sequence or distribution of its values. But they could be met by a constant stored random sequence, such as the "random" sequence printed in the CRC Standard Mathematical Tables [CRC].

A typical pseudo-random number generation technique, known as a linear congruence pseudo-random number generator, is modular arithmetic where the value numbered N+1 is calculated from the value numbered N by

$$V_{N+1} = (V_N * a + b)(\mathrm{Mod}\ c)$$

The above technique has a strong relationship to linear shift register pseudo-random number generators, which are well understood cryptographically [SHIFT*]. In such generators bits are introduced at one end of a shift register as the Exclusive Or (binary sum without carry) of bits from selected fixed taps into the register. For example:



$$V_{N+1} = ((V_N * 2) + B_0 .xor. B_2 ...)(Mod\ 2^n)$$

The goodness of traditional pseudo-random number generator algorithms is measured by statistical tests on such sequences. Carefully chosen values of the initial V and a, b, and c or the placement of shift register tap in the above simple processes can produce excellent statistics.

These sequences may be adequate in simulations (Monte Carlo experiments) as long as the sequence is orthogonal to the structure of the space being explored. Even there, subtle patterns may cause problems. However, such sequences are clearly bad for use in security applications. They are fully predictable if the initial state is known. Depending on the form of the pseudo-random number generator, the sequence may be determinable from observation of a short portion of the sequence [CRYPTO*, STERN]. For example, with the generators above, one can determine V(n+1) given knowledge of V(n). In fact, it has been shown that with these techniques, even if only one bit of the pseudo-random values are released, the seed can be determined from short sequences.

Not only have linear congruent generators been broken, but techniques are now known for breaking all polynomial congruent generators. [KRAWCZYK]

4. Unpredictability

Randomness in the traditional sense described in section 3 is NOT the same as the unpredictability required for security use.

For example, use of a widely available constant sequence, such as that from the CRC tables, is very weak against an adversary. Once they learn of or guess it, they can easily break all security, future and past, based on the sequence. [CRC] Yet the statistical properties of these tables are good.

The following sections describe the limitations of some randomness generation techniques and sources.

4.1 Problems with Clocks and Serial Numbers

Computer clocks, or similar operating system or hardware values, provide significantly fewer real bits of unpredictability than might appear from their specifications.

Tests have been done on clocks on numerous systems and it was found that their behavior can vary widely and in unexpected ways. One version of an operating system running on one set of hardware may actually provide, say, microsecond resolution in a clock while a

different configuration of the "same" system may always provide the same lower bits and only count in the upper bits at much lower resolution. This means that successive reads on the clock may produce identical values even if enough time has passed that the value "should" change based on the nominal clock resolution. There are also cases where frequently reading a clock can produce artificial sequential values because of extra code that checks for the clock being unchanged between two reads and increases it by one! Designing portable application code to generate unpredictable numbers based on such system clocks is particularly challenging because the system designer does not always know the properties of the system clocks that the code will execute on.

Use of a hardware serial number such as an Ethernet address may also provide fewer bits of uniqueness than one would guess. Such quantities are usually heavily structured and subfields may have only a limited range of possible values or values easily guessable based on approximate date of manufacture or other data. For example, it is likely that a company that manfactures both computers and Ethernet adapters will, at least internally, use its own adapters, which significantly limits the range of built in addresses.

Problems such as those described above related to clocks and serial numbers make code to produce unpredictable quantities difficult if

the code is to be ported across a variety of computer platforms and systems.

4.2 Timing and Content of External Events

It is possible to measure the timing and content of mouse movement, key strokes, and similar user events. This is a reasonable source of unguessable data with some qualifications. On some machines, inputs such as key strokes are buffered. Even though the user's inter-keystroke timing may have sufficient variation and unpredictability, there might not be an easy way to access that variation. Another problem is that no standard method exists to sample timing details. This makes it hard to build standard software intended for distribution to a large range of machines based on this technique.

The amount of mouse movement or the keys actually hit are usually

easier to access than timings but may yield less unpredictability as the user may provide highly repetitive input.

Other external events, such as network packet arrival times, can also be used with care. In particular, the possibility of manipulation of such times by an adversary and the lack of history on system start up must be considered.

4.3 The Fallacy of Complex Manipulation

One strategy which may give a misleading appearance of unpredictability is to take a very complex algorithm (or an excellent traditional pseudo-random number generator with good statistical properties) and calculate a cryptographic key by starting with the current value of a computer system clock as the seed. An adversary who knew roughly when the generator was started would have a relatively small number of seed values to test as they would know likely values of the system clock. Large numbers of pseudo-random bits could be generated but the search space an adversary would need to check could be quite small.

Thus very strong and/or complex manipulation of data will not help if the adversary can learn what the manipulation is and there is not enough unpredictability in the starting seed value. Even if they can not learn what the manipulation is, they may be able to use the limited number of results stemming from a limited number of seed values to defeat security.

Another serious strategy error is to assume that a very complex pseudo-random number generation algorithm will produce strong random

numbers when there has been no theory behind or analysis of the algorithm. There is a excellent example of this fallacy right near the beginning of chapter 3 in [KNUTH] where the author describes a complex algorithm. It was intended that the machine language program corresponding to the algorithm would be so complicated that a person trying to read the code without comments wouldn't know what the program was doing. Unfortunately, actual use of this algorithm showed that it almost immediately converged to a single repeated value in one case and a small cycle of values in another case.

Not only does complex manipulation not help you if you have a limited range of seeds but blindly chosen complex manipulation can destroy the randomness in a good seed!

4.4 The Fallacy of Selection from a Large Database

Another strategy that can give a misleading appearance of unpredictability is selection of a quantity randomly from a database and assume that its strength is related to the total number of bits in the database. For example, typical USENET servers process many megabytes of information per day. Assume a random quantity was selected by fetching 32 bytes of data from a random starting point in this data. This does not yield 32*8 = 256 bits worth of unguessability. Even after allowing that much of the data is human language and probably has no more than 2 or 3 bits of information per byte, it doesn't yield 32*2 = 64 bits of unguessability. For an adversary with access to the same usenet database the unguessability rests only on the starting point of the selection. That is perhaps a little over a couple of dozen bits of unguessability.

The same argument applies to selecting sequences from the data on a publicly available CD/DVD recording or any other large public database. If the adversary has access to the same database, this "selection from a large volume of data" step buys very little. However, if a selection can be made from data to which the adversary has no access, such as system buffers on an active multi-user system, it may be of help.

5. Hardware for Randomness

Is there any hope for true strong portable randomness in the future? There might be. All that's needed is a physical source of unpredictable numbers.

A thermal noise (sometimes called Johnson noise in integrated circuits) or radioactive decay source and a fast, free-running oscillator would do the trick directly [GIFFORD]. This is a trivial amount of hardware, and could easily be included as a standard part of a computer system's architecture. Furthermore, any system with a spinning disk or ring oscillator and a stable (crystal) time source or the like has an adequate source of randomness ([DAVIS] and Section 5.4). All that's needed is the common perception among computer vendors that this small additional hardware and the software to access it is necessary and useful.

5.1 Volume Required

How much unpredictability is needed? Is it possible to quantify the requirement in, say, number of random bits per second?

The answer is not very much is needed. For AES, the key can be 128 bits and, as we show in an example in Section 8, even the highest security system is unlikely to require a keying material of much over 200 bits. If a series of keys are needed, they can be generated from a strong random seed using a cryptographically strong sequence as explained in Section 6.3. A few hundred random bits generated at start up or once a day would be enough using such techniques. Even if the random bits are generated as slowly as one per second and it is not possible to overlap the generation process, it should be tolerable in high security applications to wait 200 seconds occasionally.

These numbers are trivial to achieve. It could be done by a person repeatedly tossing a coin. Almost any hardware process is likely to be much faster.

5.2 Sensitivity to Skew

Is there any specific requirement on the shape of the distribution of the random numbers? The good news is the distribution need not be uniform. All that is needed is a conservative estimate of how non-uniform it is to bound performance. Simple techniques to de-skew the bit stream are given below and stronger techniques are mentioned in Section 6.1.2 below.

5.2.1 Using Stream Parity to De-Skew

Consider taking a sufficiently long string of bits and map the string to "zero" or "one". The mapping will not yield a perfectly uniform distribution, but it can be as close as desired. One mapping that serves the purpose is to take the parity of the string. This has the advantages that it is robust across all degrees of skew up to the estimated maximum skew and is absolutely trivial to implement in hardware.

The following analysis gives the number of bits that must be sampled:

Suppose the ratio of ones to zeros is 0.5 + e : 0.5 - e, where e is between 0 and 0.5 and is a measure of the "eccentricity" of the distribution. Consider the distribution of the parity function of N bit samples. The probabilities that the parity will be one or zero will be the sum of the odd or even terms in the binomial expansion of (p + q)^N, where p = 0.5 + e, the probability of a one, and q = 0.5 - e, the probability of a zero.

These sums can be computed easily as

$$1/2 * ((p + q)^N + (p - q)^N)$$

and

$$1/2 * ((p + q)^N - (p - q)^N).$$

(Which one corresponds to the probability the parity will be 1 depends on whether N is odd or even.)

Since p + q = 1 and p - q = 2e, these expressions reduce to

$$1/2 * [1 + (2e)^N]$$

and

$$1/2 * [1 - (2e)^N].$$

Neither of these will ever be exactly 0.5 unless e is zero, but we can bring them arbitrarily close to 0.5. If we want the probabilities to be within some delta d of 0.5, i.e. then

$$(0.5 + (0.5 * (2e)^N)) < 0.5 + d.$$

Solving for N yields N > log(2d)/log(2e). (Note that 2e is less than 1, so its log is negative. Division by a negative number reverses the sense of an inequality.)

The following table gives the length of the string which must be sampled for various degrees of skew in order to come within 0.001 of a 50/50 distribution.

Prob(1)	e	N
0.5	0.00	1
0.6	0.10	4
0.7	0.20	7
0.8	0.30	13
0.9	0.40	28
0.95	0.45	59
0.99	0.49	308

The last entry shows that even if the distribution is skewed 99% in favor of ones, the parity of a string of 308 samples will be within 0.001 of a 50/50 distribution.

5.2.2 Using Transition Mappings to De-Skew

Another technique, originally due to von Neumann [VON NEUMANN], is to examine a bit stream as a sequence of non-overlapping pairs. You could then discard any 00 or 11 pairs found, interpret 01 as a 0 and 10 as a 1. Assume the probability of a 1 is 0.5+e and the probability of a 0 is 0.5-e where e is the eccentricity of the source and described in the previous section. Then the probability of each pair is as follows:

pair	probability		
00	(0.5 - e)^2	=	0.25 - e + e^2
01	(0.5 - e)*(0.5 + e)	=	0.25 - e^2
10	(0.5 + e)*(0.5 - e)	=	0.25 - e^2

11	(0.5 + e)^2 = 0.25 + e + e^2

This technique will completely eliminate any bias but at the expense of taking an indeterminate number of input bits for any particular desired number of output bits. The probability of any particular pair being discarded is 0.5 + 2e^2 so the expected number of input bits to produce X output bits is X/(0.25 - e^2).

This technique assumes that the bits are from a stream where each bit has the same probability of being a 0 or 1 as any other bit in the stream and that bits are not correlated, i.e., that the bits are

identical independent distributions. If alternate bits were from two correlated sources, for example, the above analysis breaks down.

The above technique also provides another illustration of how a simple statistical analysis can mislead if one is not always on the lookout for patterns that could be exploited by an adversary. If the algorithm were mis-read slightly so that overlapping successive bits pairs were used instead of non-overlapping pairs, the statistical analysis given is the same; however, instead of providing an unbiased uncorrelated series of random 1's and 0's, it instead produces a totally predictable sequence of exactly alternating 1's and 0's.

5.2.3 Using FFT to De-Skew

When real world data consists of strongly biased or correlated bits, it may still contain useful amounts of randomness. This randomness can be extracted through use of the discrete Fourier transform or its optimized variant, the FFT.

Using the Fourier transform of the data, strong correlations can be discarded. If adequate data is processed and remaining correlations decay, spectral lines approaching statistical independence and normally distributed randomness can be produced [BRILLINGER].

5.2.4 Using S-Boxes to De-Skew

Many modern block encryption functions, including DES and AES, incorporate modules known as S-Boxes (substitution boxes). These produce a smaller number of outputs from a larger number of inputs through a complex non-linear mixing function which would have the effect of concentrating limited entropy in the inputs into the output.

S-Boxes sometimes incorporate bent boolean functions which are functions of an even number of bits producing one output bit with maximum non-linearity. Looking at the output for all input pairs differing in any particular bit position, exactly half the outputs are different.

An S-Box in which each output bit is produced by a bent function such that any linear combination of these functions is also a bent function is called a "perfect S-Box". Repeated application or cascades of such boxes can be used to de-skew. [SBOX*]

5.2.5 Using Compression to De-Skew

Reversible compression techniques also provide a crude method of de-skewing a skewed bit stream. This follows directly from the definition of reversible compression and the formula in Section 2 above for the amount of information in a sequence. Since the compression is reversible, the same amount of information must be present in the shorter output than was present in the longer input. By the Shannon information equation, this is only possible if, on average, the probabilities of the different shorter sequences are more uniformly distributed than were the probabilities of the longer sequences. Thus the shorter sequences must be de-skewed relative to the input.

However, many compression techniques add a somewhat predictable preface to their output stream and may insert such a sequence again periodically in their output or otherwise introduce subtle patterns of their own. They should be considered only a rough technique compared with those described above or in Section 6.1.2. At a minimum, the beginning of the compressed sequence should be skipped and only later bits used for applications requiring random bits.

5.3 Existing Hardware Can Be Used For Randomness

As described below, many computers come with hardware that can, with care, be used to generate truly random quantities.

5.3.1 Using Existing Sound/Video Input

Increasingly computers are being built with inputs that digitize some real world analog source, such as sound from a microphone or video input from a camera. Under appropriate circumstances, such input can provide reasonably high quality random bits. The "input" from a sound digitizer with no source plugged in or a camera with the lens cap on, if the system has enough gain to detect anything, is essentially thermal noise.

For example, on a SPARCstation, one can read from the /dev/audio device with nothing plugged into the microphone jack. Such data is essentially random noise although it should not be trusted without some checking in case of hardware failure. It will, in any case, need to be de-skewed as described elsewhere.

Combining this with compression to de-skew one can, in UNIXese, generate a huge amount of medium quality random data by doing


```
cat /dev/audio | compress - >random-bits-file
```

5.3.2 Using Existing Disk Drives

Disk drives have small random fluctuations in their rotational speed due to chaotic air turbulence [DAVIS]. By adding low level disk seek time instrumentation to a system, a series of measurements can be obtained that include this randomness. Such data is usually highly correlated so that significant processing is needed, such as FFT (see section 5.2.3). Nevertheless experimentation has shown that, with such processing, most disk drives easily produce 100 bits a minute or more of excellent random data.

Partly offsetting this need for processing is the fact that disk drive failure will normally be rapidly noticed. Thus, problems with this method of random number generation due to hardware failure are unlikely.

5.4 Ring Oscillator Sources

If an integrated circuit is being designed or field programmed, an odd number of gates can be connected in series to produce a free-running ring oscillator. By sampling a point in the ring at a precise fixed frequency, say one determined by a stable crystal oscialltor, some amount of entropy can be extracted due to slight variations in the free-running osciallator.

Such bits will have to be heavily de-skewed as disk rotation timings must be (Section 5.3.2). An engineering study would be needed to determine the amount of entropy being produced depending on the particular design. It may be possible to increase the rate of entropy by xor'ing sampled values from a few ring osciallators with relatively prime lengths or the like. In any case, this can be a good, medium speed source whose cost is a trivial number of gates by modern standards.

6. Recommended Software Strategy

What is the best overall strategy for meeting the requirement for unguessable random numbers in the absence of a reliable hardware source? It is to obtain random input from a number of uncorrelated sources and to mix them with a strong mixing function. Such a

function will preserve the randomness present in any of the sources even if other quantities being combined happen to be fixed or easily guessable. This may be advisable even with a good hardware source, as hardware can also fail, though this should be weighed against any increase in the chance of overall failure due to added software complexity.

6.1 Mixing Functions

A strong mixing function is one which combines two or more inputs and produces an output where each output bit is a different complex non-linear function of all the input bits. On average, changing any input bit will change about half the output bits. But because the relationship is complex and non-linear, no particular output bit is guaranteed to change when any particular input bit is changed.

Consider the problem of converting a stream of bits that is skewed towards 0 or 1 to a shorter stream which is more random, as discussed in Section 5.2 above. This is simply another case where a strong mixing function is desired, mixing the input bits to produce a smaller number of output bits. The technique given in Section 5.2.1 of using the parity of a number of bits is simply the result of successively Exclusive Or'ing them which is examined as a trivial mixing function immediately below. Use of stronger mixing functions to extract more of the randomness in a stream of skewed bits is examined in Section 6.1.2.

6.1.1 A Trivial Mixing Function

A trivial example for single bit inputs is the Exclusive Or function, which is equivalent to addition without carry, as show in the table below. This is a degenerate case in which the one output bit always changes for a change in either input bit. But, despite its simplicity, it will still provide a useful illustration.

input 1	input 2	output
0	0	0
0	1	1
1	0	1
1	1	0

If inputs 1 and 2 are uncorrelated and combined in this fashion then the output will be an even better (less skewed) random bit than the inputs. If we assume an "eccentricity" e as defined in Section 5.2 above, then the output eccentricity relates to the input eccentricity as follows:

$$e_{output} = 2 * e_{input\ 1} * e_{input\ 2}$$

Since e is never greater than 1/2, the eccentricity is always improved except in the case where at least one input is a totally skewed constant. This is illustrated in the following table where the top and left side values are the two input eccentricities and the entries are the output eccentricity:

e	0.00	0.10	0.20	0.30	0.40	0.50
0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.10	0.00	0.02	0.04	0.06	0.08	0.10
0.20	0.00	0.04	0.08	0.12	0.16	0.20
0.30	0.00	0.06	0.12	0.18	0.24	0.30
0.40	0.00	0.08	0.16	0.24	0.32	0.40
0.50	0.00	0.10	0.20	0.30	0.40	0.50

However, keep in mind that the above calculations assume that the inputs are not correlated. If the inputs were, say, the parity of the number of minutes from midnight on two clocks accurate to a few seconds, then each might appear random if sampled at random intervals much longer than a minute. Yet if they were both sampled and combined with xor, the result would be zero most of the time.

6.1.2 Stronger Mixing Functions

The US Government Advanced Encryption Standard [AES] is an example of a strong mixing function for multiple bit quantities. It takes up to 384 bits of input (128 bits of "data" and 256 bits of "key") and

produces 128 bits of output each of which is dependent on a complex

non-linear function of all input bits. Other encryption functions with this characteristic, such as [DES], can also be used by considering them to mix all of their key and data input bits.

Another good family of mixing functions are the "message digest" or hashing functions such as The US Government Secure Hash Standards [SHA*] and the MD4, MD5 [MD4, MD5] series. These functions all take an arbitrary amount of input and produce an output mixing all the input bits. The MD* series produce 128 bits of output, SHA-1 produces 160 bits, and other SHA functions produce larger numbers of bits.

Although the message digest functions are designed for variable amounts of input, AES and other encryption functions can also be used to combine any number of inputs. If 128 bits of output is adequate, the inputs can be packed into a 128 bit data quantity and successive AES keys, padding with zeros if needed, which are then used to successively encrypt using AES in Electronic Codebook Mode [DES MODES]. If more than 128 bits of output are needed, use more complex mixing. For example, if inputs are packed into three quantities, A, B, and C, use AES to encrypt A with B as a key and then with C as a key to produce the 1st part of the output, then encrypt B with C and then A for more output and, if necessary, encrypt C with A and then B for yet more output. Still more output can be produced by reversing the order of the keys given above to stretch things. The same can be done with the hash functions by hashing various subsets of the input data to produce multiple outputs. But keep in mind that it is impossible to get more bits of "randomness" out than are put in.

An example of using a strong mixing function would be to reconsider the case of a string of 308 bits each of which is biased 99% towards zero. The parity technique given in Section 5.2.1 above reduced this to one bit with only a 1/1000 deviance from being equally likely a zero or one. But, applying the equation for information given in Section 2, this 308 bit skewed sequence has over 5 bits of information in it. Thus hashing it with SHA-1 and taking the bottom 5 bits of the result would yield 5 unbiased random bits as opposed to the single bit given by calculating the parity of the string.

6.1.3 Diffie-Hellman as a Mixing Function

Diffie-Hellman exponential key exchange is a technique that yields a shared secret between two parties that can be made computationally infeasible for a third party to determine even if they can observe all the messages between the two communicating parties. This shared secret is a mixture of initial quantities generated by each of them [D-H]. If these initial quantities are random, then the shared secret contains the combined randomness of them both, assuming they are uncorrelated.

6.1.4 Using a Mixing Function to Stretch Random Bits

While it is not necessary for a mixing function to produce the same or fewer bits than its inputs, mixing bits cannot "stretch" the amount of random unpredictability present in the inputs. Thus four inputs of 32 bits each where there is 12 bits worth of unpredicatability (such as 4,096 equally probable values) in each input cannot produce more than 48 bits worth of unpredictable output. The output can be expanded to hundreds or thousands of bits by, for example, mixing with successive integers, but the clever adversary's search space is still 2^48 possibilities. Furthermore, mixing to fewer bits than are input will tend to strengthen the randomness of the output the way using Exclusive Or to produce one bit from two did above.

The last table in Section 6.1.1 shows that mixing a random bit with a constant bit with Exclusive Or will produce a random bit. While this is true, it does not provide a way to "stretch" one random bit into more than one. If, for example, a random bit is mixed with a 0 and then with a 1, this produces a two bit sequence but it will always be either 01 or 10. Since there are only two possible values, there is still only the one bit of original randomness.

6.1.5 Other Factors in Choosing a Mixing Function

For local use, AES has the advantages that it has been widely tested for flaws, is reasonably efficient in software, and is widely documented and implemented with hardware and software implementations available all over the world including open source code. The SHA* family are younger algorithms but there is no particular reason to believe they are flawed. Both SHA* and MD5 were derived from the

earlier MD4 algorithm. Some signs of weakness have been found in MD4 and MD5. They all have source code available [SHA*, MD*].

AES and SHA* have been vouched for the the US National Security Agency (NSA) on the basis of criteria that primarily remain secret, as was DES. While this has been the cause of much speculation and doubt, investigation of DES over the years has indicated that NSA involvement in modifications to its design, which originated with IBM, was primarily to strengthen it. No concealed or special weakness has been found in DES. It is very likely that the NSA modifications to MD4 to produce the SHA* similarly strengthened these algorithms, possibly against threats not yet known in the public cryptographic community.

AES, DES, SHA*, MD4, and MD5 are believed to be royalty free for all purposes. Continued advances in crypography and computing power have cast doubts on MD4 and MD5 so their use is generally not recommended.

Another advantage of the SHA* or similar hashing algorithms over encryption algorithms in the past was that they are not subject to the same regulations imposed by the US Government prohibiting the unlicensed export or import of encryption/decryption software and hardware.

6.2 Non-Hardware Sources of Randomness

The best source of input for mixing would be a hardware randomness such as disk drive timing effected by air turbulence, audio input with thermal noise, or radioactive decay. However, if that is not available there are other possibilities. These include system clocks, system or input/output buffers, user/system/hardware/network serial numbers and/or addresses and timing, and user input. Unfortunately, any of these sources can produce limited or predicatable values under some circumstances.

Some of the sources listed above would be quite strong on multi-user systems where, in essence, each user of the system is a source of randomness. However, on a small single user or embedded system, especially at start up, it might be possible for an adversary to assemble a similar configuration. This could give the adversary inputs to the mixing process that were sufficiently correlated to

those used originally as to make exhaustive search practical.

The use of multiple random inputs with a strong mixing function is recommended and can overcome weakness in any particular input. For example, the timing and content of requested "random" user keystrokes can yield hundreds of random bits but conservative assumptions need to be made. For example, assuming at most a few bits of randomness if the inter-keystroke interval is unique in the sequence up to that point and a similar assumption if the key hit is unique but assuming that no bits of randomness are present in the initial key value or if the timing or key value duplicate previous values. The results of mixing these timings and characters typed could be further combined with clock values and other inputs.

This strategy may make practical portable code to produce good random numbers for security even if some of the inputs are very weak on some of the target systems. However, it may still fail against a high grade attack on small single user or embedded systems, especially if the adversary has ever been able to observe the generation process in the past. A hardware based random source is still preferable.

6.3 Cryptographically Strong Sequences

In cases where a series of random quantities must be generated, an adversary may learn some values in the sequence. In general, they should not be able to predict other values from the ones that they know.

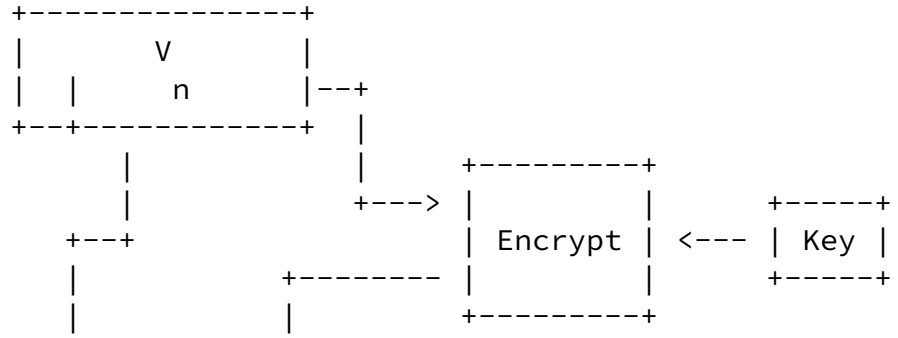
The correct technique is to start with a strong random seed, take cryptographically strong steps from that seed [CRYPTO2, CRYPTO3], and do not reveal the complete state of the generator in the sequence elements. If each value in the sequence can be calculated in a fixed way from the previous value, then when any value is compromised, all future values can be determined. This would be the case, for example, if each value were a constant function of the previously used values, even if the function were a very strong, non-invertible message digest function.

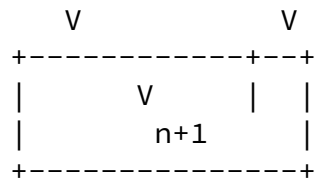
(It should be noted that if your technique for generating a sequence of key values is fast enough, it can trivially be used as the basis for a confidentiality system. If two parties use the same sequence generating technique and start with the same seed material, they will generate identical sequences. These could, for example, be xor'ed at one end with data being send, encrypting it, and xor'ed with this data as received, decrypting it due to the reversible properties of the xor operation.)

6.3.1 Traditional Strong Sequences

A traditional way to achieve a strong sequence has been to have the values be produced by hashing the quantities produced by concatenating the seed with successive integers or the like and then mask the values obtained so as to limit the amount of generator state available to the adversary.

It may also be possible to use an "encryption" algorithm with a random key and seed value to encrypt and feedback some or all of the output encrypted value into the value to be encrypted for the next iteration. Appropriate feedback techniques will usually be recommended with the encryption algorithm. An example is shown below where shifting and masking are used to combine the cypher output feedback. This type of feedback was recommended by the US Government in connection with DES [DES MODES] but should be avoided for reasons described below.





Note that if a shift of one is used, this is the same as the shift register technique described in Section 3 above but with the all important difference that the feedback is determined by a complex non-linear function of all bits rather than a simple linear or polynomial combination of output from a few bit position taps.

It has been shown by Donald W. Davies that this sort of shifted partial output feedback significantly weakens an algorithm compared will feeding all of the output bits back as input. In particular, for DES, repeated encrypting a full 64 bit quantity will give an expected repeat in about 2^63 iterations. Feeding back anything less than 64 (and more than 0) bits will give an expected repeat in between 2**31 and 2**32 iterations!

To predict values of a sequence from others when the sequence was generated by these techniques is equivalent to breaking the cryptosystem or inverting the "non-invertible" hashing involved with only partial information available. The less information revealed each iteration, the harder it will be for an adversary to predict the sequence. Thus it is best to use only one bit from each value. It has been shown that in some cases this makes it impossible to break a system even when the cryptographic system is invertible and can be broken if all of each generated value was revealed.

6.3.2 The Blum Blum Shub Sequence Generator

Currently the generator which has the strongest public proof of strength is called the Blum Blum Shub generator after its inventors [BBS]. It is also very simple and is based on quadratic residues. It's only disadvantage is that is is computationally intensive compared with the traditional techniques give in 6.3.1 above. This is not a major draw back if it is used for moderately infrequent purposes, such as generating session keys.

Simply choose two large prime numbers, say p and q, which both have

the property that you get a remainder of 3 if you divide them by 4. Let n = p * q. Then you choose a random number x relatively prime to n. The initial seed for the generator and the method for calculating subsequent values are then

$$s_0 = (x^2)(\mathrm{Mod}\ n)$$

$$s_{i+1} = (s_i^2)(\mathrm{Mod}\ n)$$

You must be careful to use only a few bits from the bottom of each s. It is always safe to use only the lowest order bit. If you use no more than the

$$\log_2(\log_2(s_i))$$

low order bits, then predicting any additional bits from a sequence generated in this manner is provable as hard as factoring n. As long as the initial x is secret, you can even make n public if you want.

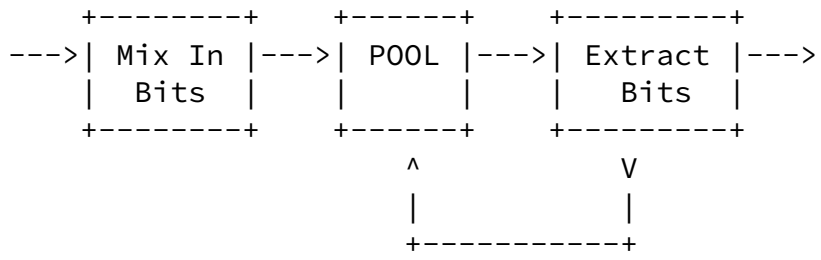
An intersting characteristic of this generator is that you can directly calculate any of the s values. In particular

$$s_i = (s_0 ((2^i)(\mathrm{Mod}\ ((p-1)*(q-1))))(\mathrm{Mod}\ n)$$

This means that in applications where many keys are generated in this fashion, it is not necessary to save them all. Each key can be effectively indexed and recovered from that small index and the initial s and n.

6.3.3 Entropy Pool Techniques

Many modern pseudo random number sources utilize the technique of maintaining a "pool" of bits and providing operations for strongly mixing input with some randomness into the pool and extracting psuedo random bits from the pool. This is illustred in the figure below.



Bits to be feed into the pool can be any of the various hardware, environmental, or user input sources discussed above. It is also common to save the state of the pool on shut down and restore it on re-starting, if stable storage is available.

In fact, all of the [MD*] and [SHA*] message digest functions are implemented by internally maintaining a pool substantially larger than their ultimate output into which the bytes of the message are mixed and from which the ultimate message digest is extracted. Thus the figure above can be implemented by using parts of the message digest code to strongly mix in any new bit supplied and to compute output bits based on the pool. However, additional code is needed so that any number of bits can be extracted and appropriate feedback from the output process is mixed into the pool so as to produce a strong pseudo-random output stream.

Care must be taken that enough entropy has been added to the pool to support particular output uses desired. See Section 7.3 for for more details on an example implementation and [RSA BULL1] for similar suggestions.

7. Key Generation Standards and Examples

Several public standards and widely deplyed examples are now in place for the generation of keys without special hardware. Two standards are described below. Both use DES but any equally strong or stronger mixing function could be substituted. Then a few widely deployed examples are described.

7.1 US DoD Recommendations for Password Generation

The United States Department of Defense has specific recommendations for password generation [DoD]. They suggest using the US Data Encryption Standard [DES] in Output Feedback Mode [DES MODES] as follows:

```
use an initialization vector determined from
     the system clock,
     system ID,
     user ID, and
     date and time;
use a key determined from
     system interrupt registers,
     system status registers, and
     system counters; and,
as plain text, use an external randomly generated 64 bit
quantity such as 8 characters typed in by a system
administrator.
```

The password can then be calculated from the 64 bit "cipher text" generated in 64-bit Output Feedback Mode. As many bits as are needed can be taken from these 64 bits and expanded into a pronounceable word, phrase, or other format if a human being needs to remember the password.

7.2 X9.17 Key Generation

The American National Standards Institute has specified a method for generating a sequence of keys as follows:

s is the initial 64 bit seed
 0

g is the sequence of generated 64 bit key quantities
 n

k is a random key reserved for generating this key sequence

t is the time at which a key is generated to as fine a resolution as is available (up to 64 bits).

DES (K, Q) is the DES encryption of quantity Q with key K

g = DES (k, DES (k, t) .xor. s)
 n   n

s = DES (k, DES (k, t) .xor. g)
 n+1   n

If g sub n is to be used as a DES key, then every eighth bit should be adjusted for parity for that use but the entire 64 bit unmodified g should be used in calculating the next s.

7.3 The /dev/random Device under Linux

The Linux operating system provides a Kernel resident random number generator. This generator makes use of events captured by the Kernel during normal system operation.

The generator consists of a random pool of bytes, by default 512 bytes (represented as 128, 4 byte integers). When an event occurs, such as a disk drive interrupt, the time of the event is xor'ed into

the pool and the pool is stirred via a primitive polynomial of degree 128. The pool itself is treated as a ring buffer, with new data being xor'ed (after stirring with the polynomial) across the entire pool.

Each call that adds entropy to the pool estimates the amount of likely true entropy the input contains. The pool itself contains a accumulator that estimates the total over all entropy of the pool.

Input events come from several sources:

1. Keyboard interrupts. The time of the interrupt as well as the scan code are added to the pool. This in effect adds entropy from the human operator by measuring inter-keystroke arrival times.

2. Disk completion and other interrupts. A system being used by a person will likely have a hard to predict pattern of disk accesses.

3. Mouse motion. The timing as well as mouse position is added in.

When random bytes are required, the pool is hashed with SHA-1 [SHA1] to yield the returned bytes of randomness. If more bytes are required

than the output of SHA-1 (20 bytes), then the hashed output is stirred back into the pool and a new hash performed to obtain the next 20 bytes. As bytes are removed from the pool, the estimate of entropy is similarly decremented.

To ensure a reasonable random pool upon system startup, the standard Linux startup scripts (and shutdown scripts) save the pool to a disk file at shutdown and read this file at system startup.

There are two user exported interfaces. /dev/random returns bytes from the pool, but blocks when the estimated entropy drops to zero. As entropy is added to the pool from events, more data becomes available via /dev/random. Random data obtained /dev/random is suitable for key generation for long term keys.

/dev/urandom works like /dev/random, however it provides data even when the entropy estimate for the random pool drops to zero. This should be fine for session keys. The risk of continuing to take data even when the pool's entropy estimate is small is that past output

may be computable from current output provided an attacker can reverse SHA-1. Given that SHA-1 should not be invertible, this is a reasonable risk.

To obtain random numbers under Linux, all an application needs to do is open either /dev/random or /dev/urandom and read the desired number of bytes.

The Linux Random device was written by Theodore Ts'o. It is based loosely on the random number generator in PGP 2.X and PGP 3.0 (aka PGP 5.0).

8. Examples of Randomness Required

Below are two examples showing rough calculations of needed randomness for security. The first is for moderate security passwords while the second assumes a need for a very high security cryptographic key.

In addition [ORMAN] and [RSA BULL13] provide information on the public key lengths that should be used for exchanging symmetric keys.

8.1 Password Generation

Assume that user passwords change once a year and it is desired that the probability that an adversary could guess the password for a particular account be less than one in a thousand. Further assume that sending a password to the system is the only way to try a password. Then the crucial question is how often an adversary can try possibilities. Assume that delays have been introduced into a system so that, at most, an adversary can make one password try every six seconds. That's 600 per hour or about 15,000 per day or about 5,000,000 tries in a year. Assuming any sort of monitoring, it is unlikely someone could actually try continuously for a year. In fact, even if log files are only checked monthly, 500,000 tries is more plausible before the attack is noticed and steps taken to change passwords and make it harder to try more passwords.

To have a one in a thousand chance of guessing the password in 500,000 tries implies a universe of at least 500,000,000 passwords or about 2^29. Thus 29 bits of randomness are needed. This can probably be achieved using the US DoD recommended inputs for password generation as it has 8 inputs which probably average over 5 bits of randomness each (see section 7.1). Using a list of 1000 words, the password could be expressed as a three word phrase (1,000,000,000 possibilities) or, using case insensitive letters and digits, six would suffice ((26+10)^6 = 2,176,782,336 possibilities).

For a higher security password, the number of bits required goes up. To decrease the probability by 1,000 requires increasing the universe of passwords by the same factor which adds about 10 bits. Thus to have only a one in a million chance of a password being guessed under the above scenario would require 39 bits of randomness and a password that was a four word phrase from a 1000 word list or eight letters/digits. To go to a one in 10^9 chance, 49 bits of randomness are needed implying a five word phrase or ten letter/digit password.

In a real system, of course, there are also other factors. For example, the larger and harder to remember passwords are, the more likely users are to write them down resulting in an additional risk

of compromise.

8.2 A Very High Security Cryptographic Key

Assume that a very high security key is needed for symmetric encryption / decryption between two parties. Assume an adversary can observe communications and knows the algorithm being used. Within the field of random possibilities, the adversary can try key values in hopes of finding the one in use. Assume further that brute force trial of keys is the best the adversary can do.

8.2.1 Effort per Key Trial

How much effort will it take to try each key? For very high security applications it is best to assume a low value of effort. This question is considered in detail in Appendix A. It concludes that a reasonable key length in 1995 for very high security is in the range of 75 to 90 bits and, since the cost of cryptography does not vary much with they key size, recommends 90 bits. To update these recommendations, just add 2/3 of a bit per year for Moore's law [MOORE]. Thus, in the year 2004, this translates to a determination that a reasonable key length is in 81 to 96 bit range.

8.2.2 Meet in the Middle Attacks

If chosen or known plain text and the resulting encrypted text are available, a "meet in the middle" attack is possible if the structure of the encryption algorithm allows it. (In a known plain text attack, the adversary knows all or part of the messages being encrypted, possibly some standard header or trailer fields. In a chosen plain text attack, the adversary can force some chosen plain text to be encrypted, possibly by "leaking" an exciting text that would then be sent by the adversary over an encrypted channel.)

An oversimplified explanation of the meet in the middle attack is as follows: the adversary can half-encrypt the known or chosen plain text with all possible first half-keys, sort the output, then half-decrypt the encoded text with all the second half-keys. If a match is found, the full key can be assembled from the halves and used to decrypt other parts of the message or other messages. At its best, this type of attack can halve the exponent of the work required by the adversary while adding a large but roughly constant factor of effort. To be assured of safety against this, a doubling of the amount of randomness in the very strong key to a minimum of 162 bits

is required for the year 2004 based on the Appendix A analysis.

This amount of randomness is beyond the limit of that in the inputs recommended by the US DoD for password generation and could require user typing timing, hardware random number generation, or other sources.

The meet in the middle attack assumes that the cryptographic algorithm can be decomposed in this way but we can not rule that out without a deep knowledge of the algorithm. Even if a basic algorithm is not subject to a meet in the middle attack, an attempt to produce a stronger algorithm by applying the basic algorithm twice (or two different algorithms sequentially) with different keys may gain less added security than would be expected. Such a composite algorithm would be subject to a meet in the middle attack.

Enormous resources may be required to mount a meet in the middle attack but they are probably within the range of the national security services of a major nation. Essentially all nations spy on other nations government traffic and several nations are believed to spy on commercial traffic for economic advantage.

It should be noted that key length calculations such at those above are controversial and depend on various assumptions about the cryptographic algorithms in use. In some cases, a professional with a deep knowledge of code breaking techniques and of the strength of the algorithm in use could be satisfied with less than half of the 162 bit key size derived above.

9. Conclusion

Generation of unguessable "random" secret quantities for security use is an essential but difficult task.

Hardware techniques to produce such randomness would be relatively simple. In particular, the volume and quality would not need to be high and existing computer hardware, such as disk drives, can be used.

Computational techniques are available to process low quality random quantities from multiple sources or a larger quantity of such low quality input from one source and produce a smaller quantity of higher quality keying material. In the absence of hardware sources of randomness, a variety of user and software sources can frequently, with care, be used instead; however, most modern systems already have hardware, such as disk drives or audio input, that could be used to produce high quality randomness.

Once a sufficient quantity of high quality seed key material (a couple of hundred bits) is available, computational techniques are available to produce cryptographically strong sequences of unpredicatable quantities from this seed material.

10. Security Considerations

The entirety of this document concerns techniques and recommendations for generating unguessable "random" quantities for use as passwords, cryptographic keys, initialiazation vectors, sequence numbers, and similar security uses.

Intellectual Property Considerations

pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Appendix: Minimal Secure Key Lengths Study

Minimal Key Lengths for Symmetric Ciphers
to Provide Adequate Commercial Security

A Report by an Ad Hoc Group of
Cryptographers and Computer Scientists

Matt Blaze, AT&T Research, mab@research.att.com
Whitfield Diffie, Sun Microsystems, diffie@eng.sun.com
Ronald L. Rivest, MIT LCS, rivest@lcs.mit.edu
Bruce Schneier, Counterpane Systems, schneier@counterpane.com
Tsutomu Shimomura, San Diego Supercomputer Center, tsutomu@sdsc.edu
Eric Thompson Access Data, Inc., eric@accessdata.com
Michael Wiener, Bell Northern Research, wiener@bnr.ca

January 1996

A.0 Abstract

Encryption plays an essential role in protecting the privacy of electronic information against threats from a variety of potential attackers. In so doing, modern cryptography employs a combination of _conventional_ or _symmetric_ cryptographic systems for encrypting data and _public key_ or _asymmetric_ systems for managing the _keys_ used by the symmetric systems. Assessing the strength required of the symmetric cryptographic systems is therefore an essential step in employing cryptography for computer and communication security.

Technology readily available today (late 1995) makes _brute-force_ attacks against cryptographic systems considered adequate for the past several years both fast and cheap. General purpose computers can be used, but a much more efficient approach is to employ commercially available _Field Programmable Gate Array (FPGA)_ technology. For attackers prepared to make a higher initial investment, custom-made, special-purpose chips make such calculations much faster and significantly lower the amortized cost per solution.

As a result, cryptosystems with 40-bit keys offer virtually no protection at this point against brute-force attacks. Even the U.S. Data Encryption Standard with 56-bit keys is increasingly inadequate. As cryptosystems often succumb to `smarter' attacks than brute-force key search, it is also important to remember that the keylengths discussed here are the minimum needed for security against the computational threats considered.

Fortunately, the cost of very strong encryption is not

---

significantly greater than that of weak encryption. Therefore, to provide adequate protection against the most serious threats --- well-funded commercial enterprises or government intelligence agencies --- keys used to protect data today should be at least 75 bits long. To protect information adequately for the next 20 years in the face of expected advances in computing power, keys in newly-deployed systems should be at least 90 bits long.

A.1. Encryption Plays an Essential Role in Protecting the Privacy of Electronic Information"

A.1.1 There is a need for information security

As we write this paper in late 1995, the development of electronic commerce and the Global Information Infrastructure is at a critical juncture. The dirt paths of the middle ages only became highways of business and culture after the security of travelers and the merchandise they carried could be assured. So too the information superhighway will be an ill-traveled road unless information, the goods of the Information Age, can be moved, stored, bought, and sold securely. Neither corporations nor individuals will entrust their private business or personal data to computer networks unless they can assure their information's security.

Today, most forms of information can be stored and processed electronically. This means a wide variety of information, with varying economic values and privacy aspects and with a wide variation in the time over which the information needs to be protected, will be found on computer networks. Consider the spectrum:

- Electronic Funds Transfers of millions or even billions of dollars, whose short term security is essential but whose exposure is brief;
- A company's strategic corporate plans, whose confidentiality must be preserved for a small number of years;
- A proprietary product (Coke formula, new drug design) that needs to be protected over its useful life, often decades; and
- Information private to an individual (medical condition, employment evaluation) that may need protection for the lifetime of the individual.

A.1.2 Encryption to protect confidentiality

Encryption Can Provide Strong Confidentiality Protection

Encryption is accomplished by scrambling data using mathematical procedures that make it extremely difficult and time consuming for anyone other than authorized recipients --- those with the correct

decryption _keys_ --- to recover the _plain text_. Proper encryption guarantees that the information will be safe even if it falls into hostile hands.

Encryption --- and decryption --- can be performed by either computer software or hardware. Common approaches include writing the algorithm on a disk for execution by a computer central processor; placing it in ROM or PROM for execution by a microprocessor; and isolating storage and execution in a computer accessory device (smart card or PCMCIA card).

The degree of protection obtained depends on several factors. These include: the quality of the cryptosystem; the way it is implemented in software or hardware (especially its reliability and the manner in which the keys are chosen); and the total number of possible keys that can be used to encrypt the information. A cryptographic algorithm is considered strong if:

1. There is no shortcut that allows the opponent to recover the plain text without using brute force to test keys until the correct one is found; and

2. The number of possible keys is sufficiently large to make such an attack infeasible.

The principle here is similar to that of a combination lock on a safe. If the lock is well designed so that a burglar cannot hear or feel its inner workings, a person who does not know the combination can open it only by dialing one set of numbers after another until it yields.

The sizes of encryption keys are measured in bits and the difficulty of trying all possible keys grows exponentially with the number of bits used. Adding one bit to the key doubles the number of possible keys; adding ten increases it by a factor of more than a thousand.

There is no definitive way to look at a cipher and determine whether a shortcut exists. Nonetheless, several encryption algorithms --- most notably the U.S Data Encryption Standard (DES) --- have been extensively studied in the public literature and are widely believed to be of very high quality. An essential element in cryptographic algorithm design is thus the length of the key, whose

size places an upper bound on the system's strength.

Throughout this paper, we will assume that there are no shortcuts and treat the length of the key as representative of the cryptosystem's _workfactor_ --- the minimum amount of effort required to break the system. It is important to bear in mind, however, that cryptographers regard this as a rash assumption and many would recommend keys two or more times as long as needed to resist brute-force attacks. Prudent cryptographic designs not only employ longer keys than might appear to be needed, but devote more computation to encrypting and decrypting. A good example of this is the popular approach of using _triple-DES_: encrypting the output of DES twice more, using a total of three distinct keys.

Encryption systems fall into two broad classes. Conventional or symmetric cryptosystems --- those in which an entity with the ability to encrypt also has the ability to decrypt and vice versa --- are the systems under consideration in this paper. The more recent public key or asymmetric cryptosystems have the property that the ability to encrypt does not imply the ability to decrypt. In contemporary cryptography, public-key systems are indispensable for managing the keys of conventional cryptosystems. All known public key cryptosystems, however, are subject to shortcut attacks and must therefore use keys ten or more times the lengths of those discussed here to achieve the an equivalent level of security.

Although computers permit electronic information to be encrypted using very large keys, advances in computing power keep pushing up the size of keys that can be considered large and thus keep making it easier for individuals and organizations to attack encrypted information without the expenditure of unreasonable resources.

A.1.3 There are a variety of attackers

There Are Threats from a Variety of Potential Attackers.

Threats to confidentiality of information come from a number of directions and their forms depend on the resources of the attackers. `Hackers,' who might be anything from high school students to commercial programmers, may have access to mainframe computers or networks of workstations. The same people can readily buy inexpensive, off-the-shelf, boards, containing _Field Programmable Gate Array (FPGA)_ chips that function as `programmable hardware' and vastly increase the effectiveness of a cryptanalytic effort. A startup company or even a well-heeled individual could afford large numbers of these chips. A major corporation or organized crime operation with `serious money' to spend could acquire custom computer chips specially designed for decryption. An intelligence agency,

engaged in espionage for national economic advantage, could build a machine employing millions of such chips.

A.1.4 Strong encryption is not expensive

Current Technology Permits Very Strong Encryption for Effectively the Same Cost As Weaker Encryption.

It is a property of computer encryption that modest increases in computational cost can produce vast increases in security. Encrypting information very securely (e.g., with 128-bit keys) typically requires little more computing than encrypting it weakly (e.g., with 40-bit keys). In many applications, the cryptography itself accounts for only a small fraction of the computing costs, compared to such processes as voice or image compression required to prepare material for encryption.

One consequence of this uniformity of costs is that there is rarely any need to tailor the strength of cryptography to the sensitivity of the information being protected. Even if most of the information in a system has neither privacy implications nor monetary value, there is no practical or economic reason to design computer hardware or software to provide differing levels of encryption for different messages. It is simplest, most prudent, and thus fundamentally most economical, to employ a uniformly high level of encryption: the strongest encryption required for any information that might be stored or transmitted by a secure system.

A.2. Brute-Force is becoming easier

Readily Available Technology Makes Brute-Force Decryption Attacks Faster and Cheaper.

The kind of hardware used to mount a brute-force attack against an encryption algorithm depends on the scale of the cryptanalytic operation and the total funds available to the attacking enterprise. In the analysis that follows, we consider three general classes of technology that are likely to be employed by attackers with differing

resources available to them. Not surprisingly, the cryptanalytic technologies that require larger up-front investments yield the lowest cost per recovered key, amortized over the life of the hardware.

It is the nature of brute-force attacks that they can be parallelized indefinitely. It is possible to use as many machines as are available, assigning each to work on a separate part of the

problem. Thus regardless of the technology employed, the search time can be reduced by adding more equipment; twice as much hardware can be expected to find the right key in half the time. The total investment will have doubled, but if the hardware is kept constantly busy finding keys, the cost per key recovered is unchanged.

At the low end of the technology spectrum is the use of conventional personal computers or workstations programmed to test keys. Many people, by virtue of already owning or having access to the machines, are in a position use such resources at little or no cost. However, general purpose computers --- laden with such ancillary equipment as video controllers, keyboards, interfaces, memory, and disk storage --- make expensive search engines. They are therefore likely to be employed only by casual attackers who are unable or unwilling to invest in more specialized equipment.

A more efficient technological approach is to take advantage of commercially available Field Programmable Gate Arrays. FPGAs function as programmable hardware and allow faster implementations of such tasks as encryption and decryption than conventional processors. FPGAs are a commonly used tool for simple computations that need to be done very quickly, particularly simulating integrated circuits during development.

FPGA technology is fast and cheap. The cost of an AT&T ORCA chip that can test 30 million DES keys per second is $200. This is 1,000 times faster than a PC at about one-tenth the cost! FPGAs are widely available and, mounted on cards, can be installed in standard PCs just like sound cards, modems, or extra memory.

FPGA technology may be optimal when the same tool must be used for attacking a variety of different cryptosystems. Often, as with DES, a cryptosystem is sufficiently widely used to justify the construction of more specialized facilities. In these circumstances,

the most cost-effective technology, but the one requiring the largest initial investment, is the use of _Application-Specific Integrated Circuits (ASICs)_. A $10 chip can test 200 million keys per second. This is seven times faster than an FPGA chip at one-twentieth the cost.

Because ASICs require a far greater engineering investment than FPGAs and must be fabricated in quantity before they are economical, this approach is only available to serious, well-funded operations such as dedicated commercial (or criminal) enterprises and government intelligence agencies.

A.3. 40-Bit Key Lengths Offer Virtually No Protection

Current U.S. Government policy generally limits exportable mass market software that incorporates encryption for confidentiality to using the RC2 or RC4 algorithms with 40-bit keys. A 40-bit key length means that there are 2^40 possible keys. On average, half of these (2^39) must be tried to find the correct one. Export of other algorithms and key lengths must be approved on a case by case basis. For example, DES with a 56-bit key has been approved for certain applications such as financial transactions.

The recent successful brute-force attack by two French graduate students on Netscape's 40-bit RC4 algorithm demonstrates the dangers of such short keys. These students at the Ecole Polytechnique in Paris used `idle time' on the school's computers, incurring no cost to themselves or their school. Even with these limited resources, they were able to recover the 40-bit key in a few days.

There is no need to have the resources of an institution of higher education at hand, however. Anyone with a modicum of computer expertise and a few hundred dollars would be able to attack 40-bit encryption much faster. An FPGA chip --- costing approximately $400 mounted on a card --- would on average recover a 40-bit key in five hours. Assuming the FPGA lasts three years and is used continuously to find keys, the average cost per key is eight cents.

A more determined commercial predator, prepared to spend $10,000 for a set-up with 25 ORCA chips, can find 40-bit keys in an average of 12 minutes, at the same average eight cent cost. Spending more money to buy more chips reduces the time accordingly: $300,000 results in a solution in an average of 24 seconds; $10,000,000 results in an average solution in 0.7 seconds.

As already noted, a corporation with substantial resources can design and commission custom chips that are much faster. By doing this, a company spending $300,000 could find the right 40-bit key in an average of 0.18 seconds at 1/10th of a cent per solution; a larger company or government agency willing to spend $10,000,000 could find the right key on average in 0.005 seconds (again at 1/10th of a cent per solution). (Note that the cost per solution remains constant because we have conservatively assumed constant costs for chip acquisition --- in fact increasing the quantities purchased of a custom chip reduces the average chip cost as the initial design and set-up costs are spread over a greater number of chips.)

These results are summarized in Table I (below).

A.4. Even DES with 56-Bit Keys Is Increasingly Inadequate

A.4.1 DES is no panacea today

The Data Encryption Standard (DES) was developed in the 1970s by IBM and NSA and adopted by the U.S. Government as a Federal Information Processing Standard for data encryption. It was intended to provide strong encryption for the government's sensitive but unclassified information. It was recognized by many, even at the time DES was adopted, that technological developments would make DES's 56-bit key exceedingly vulnerable to attack before the end of the century.

Today, DES may be the most widely employed encryption algorithm and continues to be a commonly cited benchmark. Yet DES-like encryption strength is no panacea. Calculations show that DES is

inadequate against a corporate or government attacker committing serious resources. The bottom line is that DES is cheaper and easier to break than many believe.

As explained above, 40-bit encryption provides inadequate protection against even the most casual of intruders, content to scavenge time on idle machines or to spend a few hundred dollars. Against such opponents, using DES with a 56-bit key will provide a substantial measure of security. At present, it would take a year and a half for someone using $10,000 worth of FPGA technology to search out a DES key. In ten years time an investment of this size would allow one to find a DES key in less than a week.

The real threat to commercial transactions and to privacy on the Internet is from individuals and organizations willing to invest substantial time and money. As more and more business and personal information becomes electronic, the potential rewards to a dedicated commercial predator also increase significantly and may justify the commitment of adequate resources.

A serious effort --- on the order of $300,000 --- by a legitimate or illegitimate business could find a DES key in an average of 19 days using off-the-shelf technology and in only 3 hours using a custom developed chip. In the latter case, it would cost $38 to find each key (again assuming a 3 year life to the chip and continual use). A business or government willing to spend $10,000,000 on custom chips, could recover DES keys in an average of 6 minutes, for the same $38 per key.

At the very high end, an organization --- presumably a government intelligence agency --- willing to spend $300,000,000 could recover DES keys in 12 seconds each! The investment required is large but

not unheard of in the intelligence community. It is less than the cost of the Glomar Explorer, built to salvage a single Russian submarine, and far less than the cost of many spy satellites. Such an expense might be hard to justify in attacking a single target, but seems entirely appropriate against a cryptographic algorithm, like DES, enjoying extensive popularity around the world.

There is ample evidence of the danger presented by government intelligence agencies seeking to obtain information not only for military purposes but for commercial advantage. Congressional

hearings in 1993 highlighted instances in which the French and Japanese governments spied on behalf of their countries' own businesses. Thus, having to protect commercial information against such threats is not a hypothetical proposition.

A.4.2 There are smarter avenues of attack than brute force

It is easier to walk around a tree than climb up and down it. There is no need to break the window of a house to get in if the front door is unlocked.

Calculations regarding the strength of encryption against brute-force attack are _worst case_ scenarios. They assume that the ciphers are in a sense perfect and that attempts to find shortcuts have failed. One important point is that the crudest approach --- searching through the keys --- is entirely feasible against many widely used systems. Another is that the keylengths we discuss are always minimal. As discussed earlier, prudent designs might use keys twice or three times as long to provide a margin of safety.

A.4.3 Other algorithms are similar

The Analysis for Other Algorithms Is Roughly Comparable.

The above analysis has focused on the time and money required to find a key to decrypt information using the RC4 algorithm with a 40-bit key or the DES algorithm with its 56-bit key, but the results are not peculiar to these ciphers. Although each algorithm has its own particular characteristics, the effort required to find the keys of other ciphers is comparable. There may be some differences as the result of implementation procedures, but these do not materially affect the brute-force breakability of algorithms with roughly comparable key lengths.

Specifically, it has been suggested at times that differences in set-up procedures, such as the long key-setup process in RC4, result

in some algorithms having effectively longer keys than others. For the purpose of our analysis, such factors appear to vary the

effective key length by no more than about eight bits.

A.5. Appropriate Key Lengths for the Future --- A Proposal

Table I summarizes the costs of carrying out brute-force attacks against symmetric cryptosystems with 40-bit and 56-bit keys using networks of general purpose computers, Field Programmable Gate Arrays, and special-purpose chips.

It shows that 56 bits provides a level of protection --- about a year and a half --- that would be adequate for many commercial purposes against an opponent prepared to invest $10,000. Against an opponent prepared to invest $300,000, the period of protection has dropped to the barest minimum of 19 days. Above this, the protection quickly declines to negligible. A very large, but easily imaginable, investment by an intelligence agency would clearly allow it to recover keys in real time.

What workfactor would be required for security today? For an opponent whose budget lay in the $10 to 300 million range, the time required to search out keys in a 75-bit keyspace would be between 6 years and 70 days. Although the latter figure may seem comparable to the `barest minimum' 19 days mentioned earlier, it represents --- under our amortization assumptions --- a cost of $19 million and a recovery rate of only five keys a year. The victims of such an attack would have to be fat targets indeed.

Because many kinds of information must be kept confidential for long periods of time, assessment cannot be limited to the protection required today. Equally important, cryptosystems --- especially if they are standards --- often remain in use for years or even decades. DES, for example, has been in use for more than 20 years and will probably continue to be employed for several more. In particular, the lifetime of a cryptosystem is likely to exceed the lifetime of any individual product embodying it.

A rough estimate of the minimum strength required as a function of time can be obtained by applying an empirical rule, popularly called `Moore's Law,' which holds that the computing power available for a given cost doubles every 18 months. Taking into account both the lifetime of cryptographic equipment and the lifetime of the secrets it protects, we believe it is prudent to require that encrypted data should still be secure in 20 years. Moore's Law thus predicts that the keys should be approximately 14 bits longer than required to protect against an attack today.

Bearing in mind that the additional computational costs of stronger encryption are modest, we strongly recommend a minimum key-length of 90 bits for symmetric cryptosystems.

It is instructive to compare this recommendation with both Federal Information Processing Standard 46, The Data Encryption Standard (DES), and Federal Information Processing Standard 185, The Escrowed Encryption Standard (EES). DES was proposed 21 years ago and used a 56-bit key. Applying Moore's Law and adding 14 bits, we see that the strength of DES when it was proposed in 1975 was comparable to that of a 70-bit system today. Furthermore, it was estimated at the time that DES was not strong enough and that keys could be recovered at a rate of one per day for an investment of about twenty-million dollars. Our 75-bit estimate today corresponds to 61 bits in 1975, enough to have moved the cost of key recovery just out of reach. The Escrowed Encryption Standard, while unacceptable to many potential users for other reasons, embodies a notion of appropriate key length that is similar to our own. It uses 80-bit keys, a number that lies between our figures of 75 and 90 bits.

Table I

Type of Attacker	Budget	Tool	Time and cost per key recovered 40bits	56bits	Length Needed for protection in Late 1995
Pedestrian Hacker					
	tiny	scavenged computer time	1 week	infeasible	45
	$400	FPGA	5 hours ($0.08)	38 years ($5,000)	50
Small Business					
	$10,000	FPGA	12 minutes ($0.08)	556 days ($5,000)	55
Corporate Department					
	$300K	FPGA or ASIC	24 seconds ($0.08) .18 seconds	19 days ($5,000) 3 hours	60

		($0.001)	($38)	
Big Company				

$10M	FPGA or ASIC	.7 seconds ($0.08) .005 seconds ($0.001)	13 hours ($5,000) 6 minutes ($38)	70
Intellegence Agency				
$300M	ASIC	.0002 seconds ($0.001)	12 seconds ($38)	75

A.6 About the Authors

Matt Blaze is a senior research scientist at AT&T Research in the area of computer security and cryptography. Recently Blaze demonstrated weaknesses in the U.S. government's `Clipper Chip' key escrow encryption system. His current interests include large-scale trust management and the applications of smartcards.

Whitfield Diffie is a distinguished Engineer at Sun Microsystems specializing in security. In 1976 Diffie and Martin Hellman created public key cryptography, which solved the problem of sending coded information between individuals with no prior relationship and is the basis for widespread encryption in the digital information age.

Ronald L. Rivest is a professor of computer science at the Massachusetts Institute of Technology, and is Associate Director of MIT's Laboratory for Computer Science. Rivest, together with Leonard Adleman and Adi Shamir, invented the RSA public-key cryptosystem that is used widely throughout industry. Ron Rivest is one of the founders of RSA Data Security Inc. and is the creator of variable key length symmetric key ciphers (e.g., RC4).

Bruce Schneier is president of Counterpane Systems, a consulting firm specializing in cryptography and computer security. Schneier writes and speaks frequently on computer security and privacy and is the author of a leading cryptography textbook, Applied Cryptography,

and is the creator of the symmetric key cipher Blowfish.

Tsutomu Shimomura is a computational physicist employed by the San Diego Supercomputer Center who is an expert in designing software security tools. Last year, Shimomura was responsible for tracking down the computer outlaw Kevin Mitnick, who electronically stole and altered valuable electronic information around the country.

Eric Thompson heads AccessData Corporation's cryptanalytic team and is a frequent lecturer on applied crytography. AccessData specializes in data recovery and decrypting information utilizing brute force as well as `smarter' attacks. Regular clients include

the FBI and other law enforcement agencies as well as corporations.

Michael Wiener is a cryptographic advisor at Bell-Northern Research where he focuses on cryptanalysis, security architectures, and public-key infrastructures. His influential 1993 paper, Efficient DES Key Search, describes in detail how to construct a machine to brute force crack DES coded information (and provides cost estimates as well).

A.7 Acknowledgement

The [Appendix] authors would like to thank the Business Software Alliance, which provided support for a one-day meeting, held in Chicago on 20 November 1995.

Informative References

[AES] - "Specification of the Advanced Encryption Standard (AES)", United States of America, Department of Commerce, National Institute of Standards and Technology, Federal Information Processing Standard 197, November 2001.

[ASYMMETRIC] - "Secure Communications and Asymmetric Cryptosystems", edited by Gustavus J. Simmons, AAAS Selected Symposium 69, Westview Press, Inc.

[BBS] - "A Simple Unpredictable Pseudo-Random Number Generator", SIAM Journal on Computing, v. 15, n. 2, 1986, L. Blum, M. Blum, & M. Shub.

[BRILLINGER] - "Time Series: Data Analysis and Theory", Holden-Day, 1981, David Brillinger.

[CRC] - "C.R.C. Standard Mathematical Tables", Chemical Rubber Publishing Company.

[CRYPTO1] - "Cryptography: A Primer", A Wiley-Interscience Publication, John Wiley & Sons, 1981, Alan G. Konheim.

[CRYPTO2] - "Cryptography: A New Dimension in Computer Data Security", A Wiley-Interscience Publication, John Wiley & Sons, 1982, Carl H. Meyer & Stephen M. Matyas.

[CRYPTO3] - "Applied Cryptography: Protocols, Algorithsm, and Source Code in C", Second Edition, John Wiley & Sons, 1996, Bruce Schneier.

[DAVIS] - "Cryptographic Randomness from Air Turbulence in Disk Drives", Advances in Cryptology - Crypto '94, Springer-Verlag Lecture Notes in Computer Science #839, 1984, Don Davis, Ross Ihaka, and Philip Fenstermacher.

[DES] - "Data Encryption Standard", United States of America, Department of Commerce, National Institute of Standards and Technology, Federal Information Processing Standard (FIPS) 46-3, October 1999.
- "Data Encryption Algorithm", American National Standards Institute, ANSI X3.92-1981.
(See also FIPS 112, Password Usage, which includes FORTRAN code for performing DES.)

[DES MODES] - "DES Modes of Operation", United States of America, Department of Commerce, National Institute of Standards and Technology, Federal Information Processing Standard (FIPS) 81, December 1980.
- Data Encryption Algorithm - Modes of Operation, American National Standards Institute, ANSI X3.106-1983.

[D-H] - "New Directions in Cryptography", IEEE Transactions on Information Technology, November, 1976, Whitfield Diffie and Martin E. Hellman.

[DNSSEC] - RFC 2535, "Domain Name System Security Extensions", D. Eastlake, March 1999.

[DoD] - "Password Management Guideline", United States of America, Department of Defense, Computer Security Center, CSC-STD-002-85.
(See also FIPS 112, Password Usage, which incorporates CSC-STD-002-85 as one of its appendices.)

[DSS] - "Digital Signature Standard (DSS)", United States of America, Department of Commerce, National Institute of Standards and

Technoloy, Federal Information Processing Standard (FIPS) 186-2, January 2000.

[GIFFORD] - "Natural Random Number", MIT/LCS/TM-371, September 1988, David K. Gifford

[IPSEC] - RFC 2401, "Security Architecture for the Internet Protocol", S. Kent, R. Atkinson, November 1998

[KNUTH] - "The Art of Computer Programming", Volume 2: Seminumerical Algorithms, Chapter 3: Random Numbers. Addison Wesley Publishing Company, Second Edition 1982, Donald E. Knuth.

[KRAWCZYK] - "How to Predict Congruential Generators", Journal of Algorithms, V. 13, N. 4, December 1992, H. Krawczyk

[MAIL PEM] - RFCs 1421 through 1424:
- RFC 1424, Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services, 02/10/1993, B. Kaliski
- RFC 1423, Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers, 02/10/1993, D. Balenson
- RFC 1422, Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management, 02/10/1993, S. Kent
- RFC 1421, Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures, 02/10/1993, J. Linn

[MAIL PGP] - RFC 2440, "OpenPGP Message Format", J. Callas, L. Donnerhacke, H. Finney, R. Thayer", November 1998

[MAIL S/MIME] - RFC 2633, "S/MIME Version 3 Message Specification", B. Ramsdell, Ed., June 1999.

[MD4] - "The MD4 Message-Digest Algorithm", RFC1320, April 1992, R. Rivest
[MD5] - "The MD5 Message-Digest Algorithm", RFC1321, April 1992, R. Rivest

[MOORE] - Moore's Law: the exponential increase the logic density of silicon circuts. Originally formulated by Gordon Moore in 1964 as a doubling every year starting in 1962, in the late 1970s the rate fell to a doubling every 18 months and has remained there through the date of this document. See "The New Hacker's Dictionary", Third Edition, MIT Press, ISBN 0-262-18178-9, Eric S. Raymondm 1996.

[ORMAN] - "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", draft-orman-public-key-lengths-*.txt, Hilarie Orman, Paul Hoffman, work in progress.

[RFC 1750] - "Randomness Requirements for Security", D. Eastlake, S. Crocker, J. Schiller, December 1994.

[RSA BULL1] - "Suggestions for Random Number Generation in Software", RSA Laboratories Bulletin #1, January 1996.

[RSA BULL13] - "A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths", RSA Laboratories Bulletin #13, Robert Silverman, April 2000 (revised November 2001).

[SBOX1] - "Practical s-box design", S. Mister, C. Adams, Selected Areas in Cryptography, 1996.
[SBOX2] - "Perfect Non-linear S-boxes", K. Nyberg, Advances in Cryptography - Eurocrypt '91 Proceedings, Springer-Verland, 1991.

[SHANNON] - "The Mathematical Theory of Communication", University of Illinois Press, 1963, Claude E. Shannon. (originally from: Bell System Technical Journal, July and October 1948)

[SHIFT1] - "Shift Register Sequences", Aegean Park Press, Revised Edition 1982, Solomon W. Golomb.

[SHIFT2] - "Cryptanalysis of Shift-Register Generated Stream Cypher Systems", Aegean Park Press, 1984, Wayne G. Barker.

[SHA-1] - "Secure Hash Standard (SHA-1)", United States of American, National Institute of Science and Technology, Federal Information Processing Standard (FIPS) 180-1, April 1993.
- RFC 3174, "US Secure Hash Algorithm 1 (SHA1)", D. Eastlake, P. Jones, September 2001.

[SHA-2] - "Secure Hash Standard", Draft (SHA-2156/384/512), Federal Information Processing Standard 180-2, not yet issued.

[SSH] - draft-ietf-secsh-*, work in progress.

[STERN] - "Secret Linear Congruential Generators are not Cryptograhically Secure", Proceedings of IEEE STOC, 1987, J. Stern.

[TLS] - RFC 2246, "The TLS Protocol Version 1.0", T. Dierks, C. Allen, January 1999.

[VON NEUMANN] - "Various techniques used in connection with random digits", von Neumann's Collected Works, Vol. 5, Pergamon Press, 1963, J. von Neumann.

Authors Addresses

Donald E. Eastlake 3rd
Motorola Laboratories
155 Beaver Street
Milford, MA 01757 USA

Telephone: +1 508-786-7554 (w)
+1 508-634-2066 (h)
EMail: Donald.Eastlake@motorola.com

Jeffrey I. Schiller
MIT, Room E40-311
77 Massachusetts Avenue
Cambridge, MA 02139-4307 USA

Telephone: +1 617-253-0161
E-mail: jis@mit.edu

Steve Crocker

EMail: steve@stevecrocker.com

File Name and Expiration

This is file draft-eastlake-randomness2-05.txt.

It expires June 2004.