Determining Strengths For Public Keys Used
For Exchanging Symmetric Keys

Status of this memo

This document is an Internet-Draft and is in full conformance with all
provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task
Force (IETF), its areas, and its working groups. Note that other
groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference
material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
http://www.ietf.org/ietf/1id-abstracts.txt The list of Internet-Draft
Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

Implementors of systems that use public key cryptography to exchange
symmetric keys need to make the public keys resistant to some
predetermined level of attack. That level of attack resistance is the
strength of the system, and the symmetric keys that are exchanged must
be at least as strong as the system strength requirements. The three
quantities, system strength, symmetric key strength, and public key
strength, must be consistently matched for any network protocol usage.

While it is fairly easy to express the system strength requirements in
terms of a symmetric key length and to choose a cipher that has a key
length equal to or exceeding that requirement, it is harder to choose a
public key that has a cryptographic strength meeting a symmetric key
strength requirement. This document explains how to determine the length
of an asymmetric key as a function of a symmetric key strength
requirement. Some rules of thumb for estimating equivalent resistance to
large-scale attacks on various algorithms are given. The document also
addresses how changing the sizes of the underlying large integers
(moduli, group sizes, exponents, and so on) changes the time to use the
algorithms for key exchange.

Table of Contents

[1](). **Model of Protecting Symmetric Keys with Public Keys**

Many books on cryptography and security explain the need to exchange symmetric keys in public as well as the many algorithms that are used for this purpose. However, few of these discussions explain how the strengths of the public keys and the symmetric keys are related.

To understand this, picture a house with a strong lock on the front door. Next to the front door is a small lockbox that contains the key to the front door. A would-be burglar who wants to break into the house through the front door has two options: attack the lock on the front door, or attack the lock on the lockbox in order to retrieve the key. Clearly, the burglar is better off attacking the weaker of the two locks. The homeowner in this situation must make sure that adding the second entry option (the lockbox containing the front door key) is at least as strong as the lock on the front door, in order not to make the burglar's job easier.

An implementor designing a system for exchanging symmetric keys using public key cryptography must make a similar decision. Assume that an attacker wants to learn the contents of a message that is encrypted with

a symmetric key, and that the symmetric key was exchanged between the sender and recipient using public key cryptography. The attacker has two options to recover the message: a brute-force attempt to determine the symmetric key by repeated guessing, or mathematical determination of the private key used as the key exchange key. A smart attacker will work on the easier of these two problems.

A simple-minded answer to the implementor's problem is to be sure that the key exchange system is always significantly stronger than the symmetric key; this can be done by choosing a very long public key. Such a design is usually not a good idea because the key exchanges become much more expensive in terms of processing time as the length of the public keys go up. Thus, the implementor is faced with the task of trying to match the difficulty of an attack on the symmetric key with the difficulty of an attack on the public key encryption. This analysis is not necessary if the key exchange can be performed with extreme security for almost no cost in terms of elapsed time or CPU effort; unfortunately, this is not the case for public key methods today.

A third consideration is the minimum security requirement of the user. Assume the user is encrypting with CAST-128 and requires a symmetric key with a resistance time against brute-force attack of 20 years. He might start off by choosing a key with 86 random bits, and then use a one-way function such as SHA-1 to "boost" that to a block of 160 bits, and then take 128 of those bits as the key for CAST-128. In such a case, the key exchange algorithm need only match the difficulty of 86 bits, not 128 bits.

The selection procedure is:

1. **Determine the attack resistance necessary to satisfy the security requirements of the application. Do this by estimating the minimum** number of computer operations that the attacker will be forced to do in order to compromise the security of the system and then take the logarithm base two of that number. Call that logarithm value "n".

   A 1996 report recommended 90 bits as a good all-around choice for system security. The 90 bit number should be increased by about 2/3 bit/year, or about 96 bits in 2005.

2. **Choose a symmetric cipher that has a key with at least n bits and at least that much cryptanalytic strength.**

3. **Choose a key exchange algorithm with a resistance to attack of at least n bits.**

A fourth consideration might be the public key authentication method used to establish the identity of a user. This might be an RSA digital signature or a DSA digital signature. If the modulus for the authentication method isn't large enough, then the entire basis for trusting the communication might fall apart. The following step is thus

added:

[4]. Choose an authentication algorithm with a resistance to attack of at least n bits. This ensures that a similar key exchanged cannot be forged between the two parties during the secrecy lifetime of the encrypted material. This may not be strictly necessary if the authentication keys are changed frequently and they have a well-understood usage lifetime, but in lieu of this, the n bit guidance is sound.

[1.1] The key exchange algorithms

The Diffie-Hellman method uses a group, a generator, and exponents. In today's Internet standards, the group operation is based on modular multiplication. Here, the group is defined by the multiplicative group of an integer, typically a prime $p = 2q + 1$, where q is a prime, and the arithmetic is done modulo p; the generator (which is often simply 2) is denoted by g.

In Diffie-Hellman, Alice and Bob first agree (in public or in private) on the values for g and p. Alice chooses a secret large random integer (a), and Bob chooses a secret random large integer (b). Alice sends Bob A, which is $g^a \bmod p$; Bob sends Alice B, which is $g^b \bmod p$. Next, Alice computes $B^a \bmod p$, and Bob computes $A^b \bmod p$. These two numbers are equal, and the participants use a simple function of this number as the symmetric key k.

Note that Diffie-Hellman key exchange can be done over different kinds of group representations. For instance, elliptic curves defined over finite fields are a particularly efficient way to compute the key exchange [SCH95].

For RSA key exchange, assume that Bob has a public key (m) which is equal to p*q, where p and q are two secret prime numbers, and an encryption exponent e, and a decryption exponent d. For the key exchange, Alice sends Bob $E = k^e \bmod m$, where k is the secret symmetric key being exchanged. Bob recovers k by computing $E^d \bmod m$, and the two parties use k as their symmetric key. While Bob's encryption exponent e can be quite small (e.g., 17 bits), his decryption exponent d will have as many bits in it as m does.

[2]. Determining the Effort to Factor

The RSA public key encryption method is immune to brute force guessing attacks because the modulus (and thus, the secret exponent d) will have at least 512 bits, and that is too many possibilities to guess. The Diffie-Hellman exchange is also secure against guessing because the exponents will have at least twice as many bits as the symmetric keys that will be derived from them. However, both methods are susceptible to mathematical attacks that determine the structure of the public keys.

Factoring an RSA modulus will result in complete compromise of the security of the private key. Solving the discrete logarithm problem for a Diffie-Hellman modular exponentiation system will similarly destroy the security of all key exchanges using the particular modulus. This document assumes that the difficulty of solving the discrete logarithm problem is equivalent to the difficulty of factoring numbers that are the same size as the modulus. In fact, it is slightly harder because it requires more operations; based on empirical evidence so far, the ratio of difficulty is at least 20, possibly as high as 64. Solving either problem requires a great deal of memory for the last stage of the algorithm, the matrix reduction step. Whether or not this memory requirement will continue to be the limiting factor in solving larger integer problems remains to be seen. At the current time it is not, and there is active research into parallel matrix algorithms that might mitigate the memory requirements for this problem.

The number field sieve (NFS) [GOR93] [LEN93] is the best method today for solving the discrete logarithm problem. The formula for estimating the number of simple arithmetic operations needed to factor an integer, n, using the NFS method is:

$$L(n) = k * e^{((1.92 + o(1)) * \sqrt[3]{\ln(n) * (\ln(\ln(n)))^2})}$$

Many people prefer to discuss the number of MIPS years (MYs) that are needed for large operations such as the number field sieve. For such an estimation, an operation in the L(n) formula is one computer instruction. Empirical evidence indicates that 4 or 5 instructions might be a closer match, but this is a minor factor and this document sticks with one operation/one instruction for this discussion.

2.1 Choosing parameters for the equation

The expression above has two parameters that can be estimated by empirical means: k and o(1). For the range of numbers we are interested in, there is little distinction between them.

One could assume that k is 1 and o(1) is 0. This is reasonably valid if the expression is only used for estimating relative effort (instead of actual effort) and one assumes that the o(1) term is very small over the range of the numbers that are to be factored.

Or, one could assume that o(1) is small and roughly constant and thus its value can be folded into k; then estimate k from reported amounts of effort spent factoring large integers in tests.

This document uses the second approach in order to get an estimate of the significance of the factor. It appears to be minor, based on the following calculations.

Sample values from recent work with the number field sieve, collected by RSA Labs, include:

Test name	Number of decimal digits	Number of bits	MYs of effort
RSA130	130	430	500
RSA140	140	460	2000
RSA155	155	512	8000

There are no precise measurements of the amount of time used for these factorizations. In most factorization tests, hundreds or thousands of computers are used over a period of several months, but the number of their cycles were used for the factoring project, the precise distribution of processor types, speeds, and so on are not usually reported. However, in all cases, the amount of effort used was far less than the L(n) formula would predict if k was 1 and o(1) was 0.

2.2 Choosing k from empirical reports

By solving for k from the empirical reports, it appears that k is approximately 0.02. This means that the "effective key strength" of the RSA algorithm is about 5 or 6 bits less than is implied by the naive application of equation L(n) (that is, setting k to 1 and o(1) to 0). These estimates of k are fairly stable over the numbers reported in the table. The estimate is limited to a single significant digit of k because it expresses real uncertainties; however, the effect of additional digits would have make only tiny changes to the recommended key sizes.

The factorers of RSA130 used about 1700 MYs, but they felt that this was unrealistically high for prediction purposes; by using more memory on their machines, they could have easily reduced the time to 500 MYs. Thus, the value used in preparing the table above was 500. This story does, however, underscore the difficulty in getting an accurate measure of effort. This document takes the reported effort for factoring RSA155 as being the most accurate measure.

As a result of examining the empirical data, it appears that the L(n) formula can be used with the o(1) term set to 0 and with k set to 0.02 when talking about factoring numbers in the range of 100 to 200 decimal digits. The equation becomes:

$$L(n) = 0.02 * e^{(1.92 * \sqrt[3]{\ln(n) * (\ln(\ln(n)))^2})}$$

To convert L(n) from simple math instructions to MYs, divide by $3*10^{13}$. The equation for the number of MYs needed to factor an integer n then reduces to:

$$MYs = 6 * 10^{(-16)} * e^{(1.92 * \sqrt[3]{\ln(n) * (\ln(\ln(n)))^2})}$$

With what confidence can this formula be used for predicting the difficulty of factoring slightly larger numbers? The answer is that it should be a close upper bound, but each factorization effort is usually

marked by some improvement in the algorithms or their implementations that makes the running time somewhat shorter than the formula would indicate.

2.3 Pollard's rho method

In Diffie-Hellman exchanges, there is a second attack, Pollard's rho method [POL78]. The algorithm relies on finding collisions between values computed in a large number space; its success rate is proportional to the square root of the size of the space. Because of Pollard's rho method, the search space in a DH key exchange for the key (the exponent in a g^a term), must be twice as large as the symmetric key. Therefore, to securely derive a key of K bits, an implementation must use an exponent with at least 2*K bits.

When the Diffie-Hellman key exchange is done using an elliptic curve method, the NFS methods are of no avail. However, the collision method is still effective, and the need for an exponent (called a multiplier in EC's) with 2*K bits remains. The modulus used for the computation can also be 2*K bits, and this will be substantially smaller than the modulus needed for modular exponentiation methods as the desired security level increases past 64 bits of brute-force attack resistance.

One might ask, how can you compare the number of computer instructions really needed for a discrete logarithm attack to the number needed to search the keyspace of a cipher? In comparing the efforts, one should consider what a "basic operation" is. For brute force search of the keyspace of a symmetric encryption algorithm like DES, the basic operation is the time to do a key setup and the time to do one encryption. For discrete logs, the basic operation is a modular squaring. The log of the ratio of these two operations can be used as a "normalizing factor" between the two kinds of computations. However, even for very large moduli (16K bits), this factor amounts to only a few bits of extra effort.

2.4 Limits of large memory and many machines

Robert Silverman has examined the question of when it will be practical to factor RSA moduli larger than 512 bits. His analysis is based not only on the theoretical number of operations, but it also includes expectations about the availability of actual machines for performing the work (this document is based only on theoretical number of operations). He examines the question of whether or not we can expect there be enough machines, memory, and communication to factor a very large number.

The best factoring methods need a lot of random access memory for collecting data relations (sieving) and a critical final step that does a row reduction on a large matrix. The memory requirements are related to the size of the number being factored (or subjected to discrete logarithm solution). Silverman [SILIEEE99] [SIL00] has argued that there

is a practical limit to the number of machines and the amount of RAM that can be brought to bear on a single problem in the foreseeable future. He sees two problems in attacking a 1024-bit RSA modulus: the machines doing the sieving will need 64-bit address spaces and the matrix row reduction machine will need several terabytes of memory. Silverman notes that very few 64-bit machines that have the 170 gigabytes of memory needed for sieving have been sold. Nearly a billion such machines are necessary for the sieving in a reasonable amount of time (a year or two).

Silverman's conclusion, based on the history of factoring efforts and Moore's Law, is that 1024-bit RSA moduli will not be factored until about 2037. This implies a much longer lifetime to RSA keys than the theoretical analysis indicates. He argues that predictions about how many machines and memory modules will be available can be with great confidence, based on Moore's Law extrapolations and the recent history of factoring efforts.

One should give the practical considerations a great deal of weight, but in a risk analysis, the physical world is less predictable than trend graphs would indicate. In considering how much trust to put into the inability of the computer industry to satisfy the voracious needs of factorers, one must have some insight into economic considerations that are more complicated than the mathematics of factoring. The demand for computer memory is hard to predict because it is based on applications: a "killer app" might come along any day and send the memory industry into a frenzy of sales. The number of processors available on desktops may be limited by the number of desks, but very capable embedded systems account for more processor sales than desktops. As embedded systems absorb networking functions, it is not unimaginable that millions of 64-bit processors with at least gigabytes of memory will pervade our environment.

The bottom line on this is that the key length recommendations predicted by theory may be overly conservative, but they are what we have used for this document. This question of machine availability is one that should be reconsidered in light of current technology on a regular basis.

2.5 Special purpose machines

In August of 2003, a design for a special-purpose "sieving machine" (TWIRL) surfaced [Shamir2003], and it substantially changed the cost estimates for factoring numbers up to 1024 bits in size. By applying many high-speed VLSI components in parallel, such a machine might be able to carry out the sieving of 512-bit numbers in 10 minutes at a cost of $10K for the hardware. A larger version could sieve a 1024-bit number in one year for a cost of $10M. The work cites some advances in approaches to the row reduction step in concluding that the security of 1024-bit RSA moduli is doubtful.

The estimates for the time and cost for factoring 512-bit and 1024-bit

numbers correspond to a speed-up factor of about 2 million over
what can be achieved with commodity processors today.

3. Compute Time for the Algorithms

This section describes how long it takes to use the algorithms to
perform key exchanges. Again, it is important to consider the increased
time it takes to exchange symmetric keys when increasing the length of
public keys. It is important to avoid choosing unfeasibly long public
keys.

3.1 Diffie-Hellman Key Exchange

A Diffie-Hellman key exchange is done with a finite cyclic group G with
a generator g and an exponent x. As noted in the Pollard's rho method
section, the exponent has twice as many bits as are needed for the final
key. Let the size of the group G be p, let the number of bits in the
base 2 representation of p be j, and let the number of bits in the
exponent be K.

In doing the operations that result in a shared key, a generator is
raised to a power. The most efficient way to do this involves squaring a
number K times and multiplying it several times along the way. Each of
the numbers has j/w computer words in it, where w is the number of bits
in a computer word (today that will be 32 or 64 bits). A naive
assumption is that you will need to do j squarings and j/2 multiplies;
fortunately, an efficient implementation will need fewer (NB: for the
remainder of this section, n represents j/w).

A squaring operation does not need to use quite as many operations as a
multiplication; a reasonable estimate is that squaring takes .6 the
number of machine instructions of a multiply. If one prepares a table
ahead of time with several values of small integer powers of the
generator g, then only about one fifth as many multiplies are needed as
the naive formula suggests. Therefore, one needs to do the work of
approximately .8*K multiplies of n-by-n word numbers. Further, each
multiply and squaring must be followed by a modular reduction, and a
good assumption is that it is as hard to do a modular reduction as it is
to do an n-by-n word multiply. Thus, it takes K reductions for the
squarings and .2*K reductions for the multiplies. Summing this, the
total effort for a Diffie-Hellman key exchange with K bit exponents and
a modulus of n words is approximately 2*K n-by-n-word multiplies.

For 32-bit processors, integers that use less than about 30 computer
words in their representation require at least n^2 instructions for an
n-by-n-word multiply. Larger numbers will use less time, using Karatsuba
multiplications, and they will scale as about $n^{(1.58)}$ for larger n, but
that is ignored for the current discussion. Note that 64- bit processors
push the "Karatsuba cross-over" number out to even more bits.

The basic result is: if you double the size of the Diffie-Hellman modular exponentiation group, you quadruple the number of operations needed for the computation.

3.1.1 Diffie-Hellman with elliptic curve groups

Note that the ratios for computation effort as a function of modulus size hold even if you are using an elliptic curve (EC) group for Diffie-Hellman. However, for equivalent security, one can use smaller numbers in the case of elliptic curves. Assume that someone has chosen an modular exponentiation group with an 2048 bit modulus as being an appropriate security measure for a Diffie-Hellman application and wants to determine what advantage there would be to using an EC group instead. The calculation is relatively straightforward, if you assume that on the average, it is about 20 times more effort to do a squaring or multiplication in an EC group than in a modular exponentiation group. A rough estimate is that an EC group with equivalent security has about **200 bits in its representation. Then, assuming that the time is** dominated by n-by-n- word operations, the relative time is computed as:

$$((2048/200)^2)/20 ~= 5$$

showing that an elliptic curve implementation should be five times as fast as a modular exponentiation implementation.

3.2 RSA encryption and decryption

Assume that an RSA public key uses a modulus with j bits; its factors are two numbers of about j/2 bits each. The expected computation time for encryption and decryption are different. As before, we denote the number of words in the machine representation of the modulus by the symbol n.

Most implementations of RSA use a small exponent for encryption. An encryption may involve as few as 16 squarings and one multiplication, using n-by-n-word operations. Each operation must be followed by a modular reduction, and therefore the time complexity is about 16*(.6 + 1) + 1 + 1 ~= 28 n-by-n-word multiplies.

RSA decryption must use an exponent that has as many bits as the modulus, j. However, the Chinese Remainder Theorem applies, and all the computations can be done with a modulus of only n/2 words and an exponent of only j/2 bits. The computation must be done twice, once for each factor. The effort is equivalent to 2*(j/2) (n/2 by n/2)-word multiplies. Because multiplying numbers with n/2 words is only 1/4 as difficult as multiplying numbers with n words, the equivalent effort for RSA decryption is j/4 n-by-n-word multiplies.

If you double the size of the modulus for RSA, the n-by-n multiplies will take four times as long. Further, the decryption time doubles because the exponent is larger. The overall scaling cost is a factor of **4 for encryption, a factor of 8 for decryption.**

To make these numbers more real, here are a few examples of software implementations run on hardware that was current as of a few years before the publication of this document. The examples are included to show rough estimates of reasonable implementations; they are not benchmarks. As with all software, the performance will depend on the exact details of specialization of the code to the problem and the specific hardware.

The best time informally reported for a 1024-bit modular exponentiation (the decryption side of 2048-bit RSA), is 0.9 ms (about 450,000 CPU cycles) on a 500 MHz Itanium processor. This shows that newer processors are not losing ground on big number operations; the number of instructions is less than a 32-bit processor uses for a 256-bit modular exponentiation.

For less advanced processors timing, the following two tables (computed by Tero Monenen at SSH Communications) for modular exponentiation, such as would be done in a Diffie-Hellman key exchange.

Celeron 400 MHz; compiled with GNU C compiler, optimized, some platform specific coding optimizations:

group type	modulus size	exponent size	time
mod	768	~150	18 msec
mod	1024	~160	32 msec
mod	1536	~180	82 msec
ecn	155	~150	35 msec
ecn	185	~200	56 msec

The group type is from RFC2409 and is either modular exponentiation ("mod") or elliptic curve ("ecn"). All sizes here and in subsequent tables are in bits.

Alpha 500 MHz compiled with Digital's C compiler, optimized, no platform specific code:

group type	modulus size	exponent size	time
mod	768	~150	12 msec
mod	1024	~160	24 msec
mod	1536	~180	59 msec
ecn	155	~150	20 msec
ecn	185	~200	27 msec

The following two tables (computed by Eric Young) were originally for RSA signing operations, using the Chinese Remainder representation. For ease of understanding, the parameters are presented here to show the interior calculations, i.e., the size of the modulus and exponent used

by the software.

Dual Pentium II-350:

```
     equiv       equiv         equiv
    modulus    exponent        time
     size         size
      256         256          1.5 ms
      512         512          8.6 ms
     1024        1024         55.4 ms
     2048        2048         387   ms
```

Alpha 264 600mhz:

```
     equiv       equiv         equiv
    modulus    exponent        time
     size         size
      512         512          1.4 ms
```

Current chips that accelerate exponentiation can perform 1024-bit
exponentiations (1024 bit modulus, 1024 bit exponent) in about 3
milliseconds or less.

[4](#). **Equivalences of Key Sizes**

In order to determine how strong a public key is needed to protect a
particular symmetric key, you first need to determine how much effort is
needed to break the symmetric key. Many Internet security protocols
require the use of TripleDES for strong symmetric encryption, and it is
expected that the Advanced Encryption Standard (AES) will be adopted on
the Internet in the coming years. Therefore, these two algorithms are
discussed here. In this section, for illustrative purposes, we will
implicitly assume that the system security requirement is 112 bits; this
doesn't mean that 112 bits is recommended. In fact, 112 bits is arguably
too strong for any practical purpose. It is used for illustration simply
because that is the upper bound on the strength of TripleDES.

If one could simply determine the number of MYs it takes to break
TripleDES, the task of computing the public key size of equivalent
strength would be easy. Unfortunately, that isn't the case here because
there are many examples of DES-specific hardware that encrypt faster
than DES in software on a standard CPU. Instead, one must determine the
equivalent cost for a system to break TripleDES and a system to break
the public key protecting a TripleDES key.

In 1998, the Electronic Frontier Foundation (EFF) built a DES-cracking
machine [GIL98] for US$130,000 that could test about 1e11 DES keys per
second (additional money was spent on the machine's design). The
machine's builders fully admit that the machine is not well optimized,
and it is estimated that ten times the amount of money could probably
create a machine about 50 times as fast. Assuming more optimization by

guessing that a system to test TripleDES keys runs about as fast as a
system to test DES keys, so approximately US$1 million might test 5e12
TripleDES keys per second.

In case your adversaries are much richer than EFF, you may want to
assume that they have US$1 trillion, enough to test 5e18 keys per
second. An exhaustive search of the effective TripleDES space of 2^112
keys with this quite expensive system would take about 1e15 seconds or
about 33 million years. (Note that such a system would also need 2^60
bytes of RAM [MH81], which is considered free in this calculation). This
seems a needlessly conservative value. However, if computer logic speeds
continue to increase in accordance with Moore's Law (doubling in speed
every 1.5 years), then one might expect that in about 50 years, the
computation could be completed in only one year. For the purposes of
illustration, this 50 year resistance against a trillionaire is assumed
to be the minimum security requirement for a set of applications.

If 112 bits of attack resistance is the system security requirement,
then the key exchange system for TripleDES should have equivalent
difficulty; that is to say, if the attacker has US$1 trillion, you want
him to spend all his money to buy hardware today and to know that he
will "crack" the key exchange in not less than 33 million years.
(Obviously, a rational attacker would wait for about 45 years before
actually spending the money, because he could then get much better
hardware, but all attackers benefit from this sort of wait equally.)

It is estimated that a typical PC CPU of just a few years ago can
generate over 500 MIPs and could be purchased for about US$100 in
quantity; thus you get more than 5 MIPs/US$. Again, this number doubles
about every 18 months. For one trillion US dollars, an attacker can get
5e12 MIP years of computer instructions on that recent-vintage hardware.
This figure is used in the following estimates of equivalent costs for
breaking key exchange systems.

4.1 Key equivalence against special purpose brute force hardware

If the trillionaire attacker is to use conventional CPU's to "crack" a
key exchange for a 112 bit key in the same time that the special purpose
machine is spending on brute force search for the symmetric key, the key
exchange system must use an appropriately large modulus. Assume that the
trillionaire performs 5e12 MIPs of instructions per year. Use the
following equation to estimate the modulus size to use with RSA
encryption or DH key exchange:

$$5*10^{33} = (6*10^{-16})*e^{(1.92*cubrt(ln(n)*(ln(ln(n)))^2))}$$

Solving this approximately for n yields:

$$n = 10^{(625)} = 2^{(2077)}$$

Thus, assuming similar logic speeds and the current efficiency of the

number field sieve, moduli with about 2100 bits will have about the same resistance against attack as an 112-bit TripleDES key. This indicates that RSA public key encryption should use a modulus with around 2100 bits; for a Diffie-Hellman key exchange, one could use a slightly smaller modulus, but it is not a significant difference.

Key equivalence against conventional CPU brute force attack

An alternative way of estimating this assumes that the attacker has a less challenging requirement: he must only "crack" the key exchange in less time than a brute force key search against the symmetric key would take with general purpose computers. This is an "apples-to-apples" comparison, because it assumes that the attacker needs only to have computation donated to his effort, not built from a personal or national fortune. The public key modulus will be larger than the one in 4.1, because the symmetric key is going to be viable for a longer period of time.

Assume that the number of CPU instructions to encrypt a block of material using TripleDES is 300. The estimated number of computer instructions to break 112 bit TripleDES key:

        300 * 2^112
        = 1.6 * 10^(36)
        = .02*e^(1.92*cubrt(ln(n)*(ln(ln(n)))^2))

Solving this approximately for n yields:

        n = 10^(734) = 2^(2439)

Thus, for general purpose CPU attacks, you can assume that moduli with about 2400 bits will have about the same strength against attack as an 112-bit TripleDES key. This indicates that RSA public key encryption should use a modulus with around 2400 bits; for a Diffie-Hellman key exchange, one could use a slightly smaller modulus, but it not a significant difference.

Note that some authors assume that the algorithms underlying the number field sieve will continue to get better over time. These authors recommend an even larger modulus, over 4000 bits, for protecting a 112-bit symmetric key for 50 years. This points out the difficulty of long-term cryptographic security: it is all but impossible to predict progress in mathematics and physics over such a long period of time.

A One Year Attack: 80 bits of strength

Assuming a trillionaire spends his money today to buy hardware, what size key exchange numbers could he "crack" in one year? He can perform 5*e12 MYs of instructions, or

        3*10^13 * 5*10^12 = .02*e^(1.92*cubrt(ln(n)*(ln(ln(n)))^2))

Solving for an approximation of n yields

    n = 10^(360) = 2^(1195)

This is about as many operations as it would take to crack an 80-bit symmetric key by brute force.

Thus, for protecting data that has a secrecy requirement of one year against an incredibly rich attacker, a key exchange modulus with about **1200** bits protecting an 80-bit symmetric key is safe even against a nation's resources.

4.4 Key equivalence for other ciphers

Extending this logic to the AES is straightforward. For purposes of estimation for key searching, one can think of the 128-bit AES as being at least 16 bits stronger than TripleDES but about three times as fast. The time and cost for a brute force attack is approximately 2^(16) more than for TripleDES, and thus, under the assumption that 128 bits of strength is the desired security goal, the recommended key exchange modulus size is about 700 bits longer.

If it is possible to design hardware for AES cracking that is considerably more efficient than hardware for DES cracking, then (again under the assumption that the key exchange strength must match the brute force effort) the moduli for protecting the key exchange can be made smaller. However, the existence of such designs is only a matter of speculation at this early moment in the AES lifetime.

The AES ciphers have key sizes of 128 bits up to 256 bits. Should a prudent minimum security requirement, and thus the key exchange moduli, have similar strengths? The answer to this depends on whether or not one expect Moore's Law to continue unabated. If it continues, one would expect 128 bit keys to be safe for about 60 years, and 256 bit keys would be safe for another 400 years beyond that, far beyond any imaginable security requirement. But such progress is difficult to predict, as it exceeds the physical capabilities of today's devices and would imply the existence of logic technologies that are unknown or infeasible today. Quantum computing is a candidate, but too little is known today to make confident predictions about its applicability to cryptography (which itself might change over the next 100 years!).

If Moore's Law does not continue to hold, if no new computational paradigms emerge, then keys of over 100 bits in length might well be safe "forever". Note, however that others have come up with estimates based on assumptions of new computational paradigms emerging. For example, Lenstra and Verheul's web-based paper "Selecting Cryptographic Key Sizes" chooses a more conservative analysis than the one in this document.

4.5 Hash functions for deriving symmetric keys from public key algorithms

The Diffie-Hellman algorithm results in a key that is hundreds or
thousands of bits long, but ciphers need far fewer bits than that. How
can one distill a long key down to a short one without losing strength?

Cryptographic one-way hash functions are the building blocks for this,
and so long as they use all of the Diffie-Hellman key to derive each
block of the symmetric key, they produce keys with sufficient strength.

The usual recommendation is to use a good one-way hash function applied
to he base material (the result of the key exchange) and to use a subset
of the hash function output for the key. However, if the desired key
length is greater than the output of the hash function, one might wonder
how to reconcile the two.

The step of deriving extra key bits must satisfy these requirements:

- The bits must not reveal any information about the key exchange secret

- The bits must not be correlated with each other

- The bits must depend on all the bits of the key exchange secret

Any good cryptographic hash function satisfies these three requirements.
Note that the number of bits of output of the hash function is not
specified. That is because even a hash function with a very short output
can be iterated to produce more uncorrelated bits with just a little bit
of care.

For example, SHA-1 has 160 bits of output. For deriving a key of attack
resistance of 160 bits or less, SHA(DHkey) produces a good symmetric
key.

Suppose one wants a key with attack resistance of 160 bits, but it is to
be used with a cipher that uses 192 bit keys. One can iterate SHA-1 as
follows:

        Bits 1-160   of the symmetric key = K1 = SHA(DHkey | 0x00)
                     (that is, concatenate a single octet of value 0x00 to
                     the right side of the DHkey, and then hash)
        Bits 161-192 of the symmetric key = K2 =
                     select_32_bits(SHA(K1 | 0x01))

But what if one wants 192 bits of strength for the cipher? Then the
appropriate calculation is

        Bits 1-160   of the symmetric key = SHA(0x00 | DHkey)
        Bits 161-192 of the symmetric key =
                     select_32_bits(SHA(0x01 | DHkey))

(Note that in the description above, instead of concatenating a full
octet, concatenating a single bit would also be sufficient.)

The important distinction is that in the second case, the DH key is used
for each part of the symmetric key. This assures that entropy of the DH
key is not lost by iteration of the hash function over the same bits.

>From an efficiency point of view, if the symmetric key must have a great
deal of entropy, it is probably best to use a cryptographic hash
function with a large output block (192 bits or more), rather than
iterating a smaller one.

Newer hash algorithms with longer output (such as SHA-256, SHA-384, and
SHA-512) can be used with the same level of security as the stretching
algorithm described above.

4.6 **Importance of randomness**

Some of the calculations described in this document require random
inputs; for example, the secret Diffie-Hellman exponents must be chosen
based on n truly random bits (where n is the system security
requirement). The number of truly random bits is extremely important to
determining the strength of the output of the calculations. Using truly
random numbers is often overlooked, and many security applications have
been significantly weakened by using insufficient random inputs. A much
more complete description of the importance of random numbers can be
found in [ECS].

5. **Conclusion**

In this table it is assumed that attackers use general purpose
computers, that the hardware is purchased in the year 2000, and that
mathematical knowledge relevant to the problem remains the same as
today. This is an pure "apples-to-apples" comparison demonstrating how
the time for a key exchange scales with respect to the strength
requirement. The subgroup size for DSA is included, if that is being
used for supporting authentication as part of the protocol; the DSA
modulus must be as long as the DH modulus, but the size of the "q"
subgroup is also relevant.

System requirement for attack resistance (bits)	Symmetric key size (bits)	RSA or DH modulus size (bits)	DSA subgroup size (bits)
70	70	947	129
80	80	1228	148
90	90	1553	167
100	100	1926	186
150	150	4575	284

```
|    200      |    200   |     8719    |     383      |
|    250      |    250   |     14596   |     482      |
+------------+----------+-------------+-------------+
```

5.1 TWIRL Correction

If the TWIRL machine becomes a reality, and if there are advances in
parallelism for row reduction in factoring, then conservative
estimates would subtract about 11 bits from the system security column
of the table. Thus, in order to get 89 bits of security, one would
need an RSA modulus of about 1900 bits.

6. Security Considerations

The equations and values given in this document are meant to be as
accurate as possible, based on the state of the art in general purpose
computers at the time that this document is being written. No
predictions can be completely accurate, and the formulas given here are
not meant to be definitive statements of fact about cryptographic
strengths. For example, some of the empirical results used in
calibrating the formulas in this document are probably not completely
accurate, and this inaccuracy affects the estimates. It is the authors'
hope that the numbers presented here vary from real world experience as
little as possible.

7. References

7.1 Informational References

[DL] B. Dodson, A.K. Lenstra, NFS with four large primes: an explosive
experiment, Proceedings Crypto 95, Lecture Notes in Comput. Sci. 963,
(1995) 372-385.

[ECS] D. Eastlake, S. Crocker. J. Schiller, "Randomness Recommendations
for Security", RFC 1750.

[GIL98] Cracking DES: Secrets of Encryption Research, Wiretap Politics &
Chip Design , Electronic Frontier Foundation, John Gilmore (Ed.), 272
pages, May 1998, O'Reilly & Associates; ISBN: 1565925203

[GOR93] D. Gordon, "Discrete logarithms in GF(p) using the number field
sieve", SIAM Journal on Discrete Mathematics, 6 (1993), 124-138.

[LEN93] A. K. Lenstra, H. W. Lenstra, Jr. (eds), The development of the
number field sieve, Lecture Notes in Math, 1554, Springer Verlag,
Berlin, 1993.

[MH81] Merkle, R.C., and Hellman, M., "On the Security of Multiple
Encryption", Communications of the ACM, v. 24 n. 7, 1981, pp. 465-467.

[ODL95] RSA Labs Cryptobytes, Volume 1, No. 2 - Summer 1995; The Future

of Integer Factorization, A. M. Odlyzko

[ODL99] A. M. Odlyzko, Discrete logarithms: The past and the future, Designs, Codes, and Cryptography (1999).

[POL78] J. Pollard, "Monte Carlo methods for index computation mod p", Mathematics of Computation, 32 (1978), 918-924.

[RFC2409] D. Harkens and D. Carrel, "Internet Key Exchange (IKE)", RFC 2409.

[SCH95] R. Schroeppel, et al., Fast Key Exchange With Elliptic Curve Systems, In Don Coppersmith, editor, Advances in Cryptology -- CRYPTO '95, volume 963 of Lecture Notes in Computer Science, pages 43-56, 27-**31** August 1995. Springer-Verlag

[SHAMIR03] Shamir, Adi and Eran Tromer, "Factoring Large Numbers with the TWIRL Device", Advances in Cryptology - CRYPTO 2003, Springer, Lecture Notes in Computer Science 2729.

[SIL99] R. D. Silverman, RSA Laboratories Bulletin, Number 12 - May 3, 1999; An Analysis of Shamir's Factoring Device

[SIL00] R. D. Silverman, RSA Laboratories Bulletin, Number 13 - April 2000, A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths

[SILIEEE99] R. D. Silverman, "The Mythical MIPS Year", IEEE Computer, August 1999.

8. Authors' Addresses

Hilarie Orman
Purple Streak Development
500 S. Maple Dr.
Salem, UT 84653
hilarie@purplestreak.com and ho@alum.mit.edu

Paul Hoffman
VPN Consortium
127 Segre Place
Santa Cruz, CA 95060 USA
paul.hoffman@vpnc.org