Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems

Paul C. Kocher

Cryptography Consultant
P.O. Box 8243, Stanford, CA 94309, USA.
E-mail: pck@cryptography.com.

Abstract. By carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. Against a vulnerable system, the attack is computationally inexpensive and often requires only known ciphertext. Actual systems are potentially at risk, including cryptographic tokens, network-based cryptosystems, and other applications where attackers can make reasonably accurate timing measurements. Techniques for preventing the attack for RSA and Diffie-Hellman are presented. Some cryptosystems will need to be revised to protect against the attack, and new protocols and algorithms may need to incorporate measures to prevent timing attacks.

Keywords: timing attack, cryptanalysis, RSA, Diffie-Hellman, DSS.

1 Introduction

Cryptosystems often take slightly different amounts of time to process different inputs. Reasons include performance optimizations to bypass unnecessary operations, branching and conditional statements, RAM cache hits, processor instructions (such as multiplication and division) that run in non-fixed time, and a wide variety of other causes. Performance characteristics typically depend on both the encryption key and the input data (e.g., plaintext or ciphertext). While it is known that timing channels can leak data or keys across a controlled perimeter, intuition might suggest that unintentional timing characteristics would only reveal a small amount of information from a cryptosystem (such as the Hamming weight of the key). However, attacks are presented which can exploit timing measurements from vulnerable systems to find the entire secret key.

2 Cryptanalysis of a Simple Modular Exponentiator

Diffie-Hellman[2] and RSA[8] private-key operations consist of computing $R = y^x \bmod n$, where n is public and y can be found by an eavesdropper. The attacker's goal is to find x, the secret key. For the attack, the victim must compute $y^x \bmod n$ for several values of y, where y, n, and the computation time are known to the attacker. (If a new secret exponent x is chosen for each operation,

N. Koblitz (Ed.): Advances in Cryptology - CRYPTO '96, LNCS 1109, pp. 104-113, 1996.

the attack does not work.) The necessary information and timing measurements might be obtained by passively eavesdropping on an interactive protocol, since an attacker could record the messages received by the target and measure the amount of time taken to respond to each y. The attack assumes that the attacker knows the design of the target system, although in practice this could probably be inferred from timing information.

The attack can be tailored to work with virtually any implementation that does not run in fixed time, but is first outlined using the simple modular exponentiation algorithm below which computes $R = y^x \bmod n$, where x is w bits long:

```
Let s_0 = 1.
For k = 0 upto w - 1:
  If (bit k of x) is 1 then
    Let R_k = (s_k · y) mod n.
  Else
    Let R_k = s_k.
  Let s_{k+1} = R_k^2 mod n.
EndFor.
Return (R_{w-1}).
```

The attack allows someone who knows exponent bits 0..(b-1) to find bit b. To obtain the entire exponent, start with b equal to 0 and repeat the attack until the entire exponent is known.

Because the first b exponent bits are known, the attacker can compute the first b iterations of the `For` loop to find the value of s_b. The next iteration requires the first unknown exponent bit. If this bit is set, $R_b = (s_b \cdot y) \bmod n$ will be computed. If it is zero, the operation will be skipped.

The attack will be described first in an extreme hypothetical case. Suppose the target system uses a modular multiplication function that is normally extremely fast but occasionally takes much more time than an entire normal modular exponentiation. For a few s_b and y values the calculation of $R_b = (s_b \cdot y) \bmod n$ will be extremely slow, and by using knowledge about the target system's design the attacker can determine which these are. If the total modular exponentiation time is ever fast when $R_b = (s_b \cdot y) \bmod n$ is slow, exponent bit b must be zero. Conversely, if slow $R_b = (s_b \cdot y) \bmod n$ operations always result in slow total modular exponentiation times, the exponent bit is probably set. Once exponent bit b is known, the attacker can verify that the overall operation time is slow whenever $s_{b+1} = R_b^2 \bmod n$ is expected to be slow. The same set of timing measurements can then be reused to find the following exponent bits.

3 Error Correction

If exponent bit b is guessed incorrectly, the values computed for $R_{k \geq b}$ will be incorrect and, so far as the attack is concerned, essentially random. The time

required for multiplies following the error will not be reflected in the overall exponentiation time. The attack thus has an *error-detection* property; after an incorrect exponent bit guess, no more meaningful correlations are observed.

The error detection property can be used for error correction. For example, the attacker can maintain a list of the most likely exponent intermediates along with a value corresponding to the probability each is correct. The attack is continued for only the most likely candidate. If the currently-favored value is incorrect, it will tend to fall in ranking, while correct values will tend to rise. Error correction techniques increase the memory and processing requirements for the attack, but can greatly reduce the number of samples required.

4 The General Attack

The attack can be treated as a signal detection problem. The "signal" consists of the timing variation due to the target exponent bit, and "noise" results from measurement inaccuracies and timing variations due to unknown exponent bits. The properties of the signal and noise determine the number of timing measurements required to for the attack.

Given j messages $y_0, y_1, ..., y_{j-1}$ with corresponding timing measurements $T_0, T_1, ..., T_{j-1}$, the probability that a guess x_b for the first b exponent bits is correct is proportional to

$$P(x_b) \propto \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))$$

where $t(y_i, x_b)$ is the amount of time required for the first b iterations of the $y_i^x \bmod n$ computation using exponent bits x_b, and F is the expected probability distribution function of $T - t(y, x_b)$ over all y values and correct x_b. Because F is defined as the probability distribution of $T_i - t(y_i, x_b)$ if x_b is correct, it is the best function for predicting $T_i - t(y_i, x_b)$. Note that the timing measurements and intermediate s values can be used improve the estimate of F.

Given a correct guess for x_{b-1}, there are two possible values for x_b. The probability that x_b is correct and x_b' is incorrect can be found as

$$\frac{\prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))}{\prod_{i=0}^{j-1} F(T_i - t(y_i, x_b)) + \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b'))}.$$

In practice, this formula is not very useful because finding F would require extraordinary effort.

5 Simplifying the Attack

Fortunately it is generally not necessary to compute F. Each timing observation consists of $T = e + \sum_{i=0}^{w-1} t_i$, where t_i is the time required for the multiplication and squaring steps for bit i and e includes measurement error, loop overhead,

etc. Given guess x_b, the attacker can find $\sum_{i=0}^{b-1} t_i$ for each sample y. If x_b is correct, subtracting from T yields $e + \sum_{i=0}^{w-1} t_i - \sum_{i=0}^{b-1} t_i = e + \sum_{i=b}^{w-1} t_i$. Since the modular multiplication times are effectively independent from each other and from the measurement error, the variance of $e + \sum_{i=b}^{w-1} t_i$ over all observed samples is expected to be $\mathrm{Var}(e) + (w-b)\mathrm{Var}(t)$. However if only the first $c < b$ bits of the exponent guess are correct, the expected variance will be $\mathrm{Var}(e) + (w-b+2c)\mathrm{Var}(t)$. Correctly-emulated iterations decrease the expected variance by $\mathrm{Var}(t)$, while iterations following an incorrect exponent bit each increase the variance by $\mathrm{Var}(t)$. Computing the variances is easy and provides a good way to identify correct exponent bit guesses.

It is now possible to estimate the number of samples required for the attack. Suppose an attacker has j accurate timing measurements and has two guesses for the first b bits of a w-bit exponent, one correct and the other incorrect with the first error at bit c. For each guess the timing measurements can be adjusted by $\sum_{i=0}^{b-1} t_i$. The correct guess will be identified successfully if its adjusted values have the smaller variance.

It is possible to approximate t_i using independent standard normal variables. If $\mathrm{Var}(e)$ is negligible, the expected probability of a correct guess is

$$P\left(\sum_{i=0}^{j-1}\left(\sqrt{w-b}X_i + \sqrt{2(b-c)}Y_i\right)^2 > \sum_{i=0}^{j-1}\left(\sqrt{w-b}X_i\right)^2\right)$$
$$= P\left(2\sqrt{2(b-c)(w-b)}\sum_{i=0}^{j-1} X_iY_i + 2(b-c)\sum_{i=0}^{j-1} Y_i^2 > 0\right)$$

where X and Y are normal random variables with $\mu = 0$ and $\sigma = 1$. Because j is relatively large, $\sum_{i=0}^{j-1} Y_i^2 \approx j$ and $\sum_{i=0}^{j-1} X_iY_i$ is approximately normal with $\mu = 0$ and $\sigma = \sqrt{j}$, yielding

$$P\left(2\sqrt{2(b-c)(w-b)}(\sqrt{j}Z) + 2(b-c)j > 0\right) = P\left(Z > -\frac{\sqrt{j(b-c)}}{\sqrt{2(w-b)}}\right)$$

where Z is a standard normal random variable. Finally, integrating to find the probability of a correct guess yields $\Phi\left(\sqrt{\frac{j(b-c)}{2(w-b)}}\right)$, where $\Phi(x)$ is the area under the standard normal curve from $-\infty$ to x. The required number of samples (j) is thus proportional to the exponent size (w). The number of measurements might be reduced if attackers choose inputs known to have extreme timing characteristics at exponent locations of interest.

6 Experimental Results

Figure 1 shows the distribution of 10^6 modular multiplication times observed using the RSAREF toolkit[10] on a 120-MHz Pentium^{TM} computer running MSDOS^{TM}. The distribution was prepared by timing one million ($a \cdot b \bmod n$) calculations using a and b values from actual modular exponentiation operations

with random inputs. The 512-bit sample prime #1 from the RSAREF Diffie-Hellman demonstration program was used for n. A few wildly aberrant samples (which took over $1300\mu s$) were discarded. The Figure 1 distribution has mean $\mu = 1167.8\mu s$ and standard deviation $\sigma = 12.01\mu s$. The measurement error is small; the tests were run twice and the average measurement difference was found to be under $1\mu s$. RSAREF uses the same function for squaring and multiplication, so squaring and multiplication times have identical distributions.

RSAREF precomputes y^2 and y^3 mod n and processes two exponent bits at a time. In total, a 512-bit modular exponentiation with a random 256-bit exponent requires 128 iterations of the modular exponentiation loop and a total of about 352 modular multiplication and squaring operations. Each iteration of the modular exponentiation loop does two squaring operations and, if either exponent bit is nonzero, one multiply. The attack can be adjusted to append pairs of exponent bits and to evaluate four candidate values at each exponent position instead of two.

Since modular multiplications consume most of the total modular exponentiation time, it is expected that the distribution of modular exponentiation times will be approximately normal with $\mu \geq (1167.8)(352) = 411,065.6\mu s$ and $\sigma \geq 12.01\sqrt{352} = 225.3\mu s$. Figure 2 shows measurements from 5000 actual modular exponentiation operations using the same computer and modulus, which yielded $\mu = 419,901\mu s$ and $\sigma = 235\mu s$.

FIGURE 1: RSAREF Modular Multiplication Times

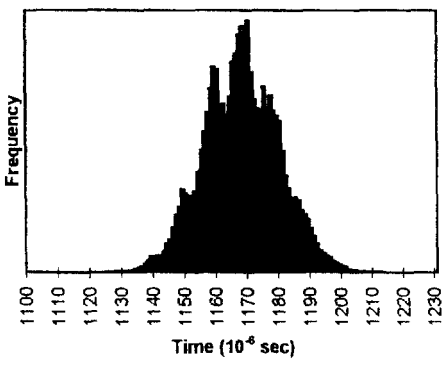
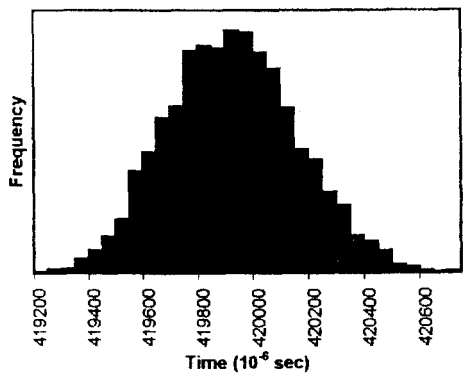


FIGURE 2: RSAREF Modular Exponentiation Times



With 250 timing measurements, the probability that subtracting the time for a correct modexp loop iteration from each sample will reduce the total variance more than subtracting an incorrect iteration is estimated to be $\Phi\left(\sqrt{\frac{j(b-c)}{2(w-b)}}\right)$, where $j = 250$, $b = 1$, $c = 0$, and $w = 127$. (There are 128 iterations of the RSAREF modexp loop for a 256-bit exponent, but the first iteration is ignored.) Correct guesses are thus expected with probability $\Phi\left(\sqrt{\frac{250(1-0)}{2(126)}}\right) \approx 0.84$. The

5000 samples from Figure 2 were divided into 20 groups of 250 samples each, and variances from subtracting the time for incorrect and correct modexp loop iterations were compared at each of the 127 exponent bit pairs. Of the 2450 trials, 2168 produced a larger variance after subtracting an incorrect modexp loop time than after subtracting the time for a correct modexp loop, yielding a probability of 0.885. The first exponent bits are most difficult, since b becomes larger as more exponent bits become known and the probabilities should improve. (The test above did not take advantage of this property.) It is important to note that accurate timing measurements were used; measurement errors which are large relative to the total modular exponentiation time standard deviation will increase the number of samples needed.

The attack is computationally quite easy. With RSAREF, the attacker has to evaluate four choices per pair of bits. Thus the attacker only has to do four times the number of operations done by the victim, not counting effort wasted by incorrect guesses.

7 Montgomery Multiplication and the CRT

Modular reduction steps usually cause most of the timing variation in a modular multiplication operation. Montgomery multiplication[6] eliminates the mod n reduction steps and, as a result, tends to reduce the size of the timing characteristics. However, some variation usually remains. If the remaining "signal" is not dwarfed by measurement errors, the variance in t_b and the variance of $\sum_{i=b+1}^{w-1} t_i$ would be reduced proportionally and the attack would still work. However if the measurement error e is large, the required number of samples will increase in proportion to $\frac{1}{\mathrm{Var}(t_i)}$.

The Chinese Remainder Theorem (CRT) is also often used to optimize RSA private key operations. With CRT, (y mod p) and (y mod q) are computed first, where y is the message. These initial modular reduction steps can be vulnerable to timing attacks. The simplest such attack is to choose values of y that are close to p or q, then use timing measurements to determine whether the guessed value is larger or smaller than the actual value of p (or q). If y is less than p, computing y mod p has no effect, while if y is larger than p, it is necessary to subtract p from y at least once. Also, if the message is very slightly larger than p, y mod p will have leading zero digits, which may reduce the amount of time required for the first multiplication step. The specific timing characteristics depend on the implementation. RSAREF's modular reduction function with a 512-bit modulus the Pentium computer with y chosen randomly between 0 and $2p$ takes an average of 42.1μs if $y < p$, as opposed to 73.9μs if $y > p$. Timing measurements from many y could be combined to successively approximate p.

In some cases it may be possible to improve the Chinese Remainder Theorem RSA attack to use known (not chosen) ciphertexts, reducing the number of messages required and making it possible to attack RSA digital signatures. Modular reduction is done by subtracting multiples of the modulus, and exploitable timing variations can be caused by variations in the number of compare-and-subtract

steps. For example, RSAREF's division loop integer-divides the uppermost two digits of y by one more than the upper digit of p, multiplies p by the quotient, shifts left the appropriate number of digits, then subtracts the result from y. If the result is larger than p (shifted left), a extra subtraction is performed. The decision whether to perform an extra subtraction step in the first loop of the division algorithm usually depends only on y (which is known) and the upper two digits of p. A timing attack could be used to determine the upper digits of p. For example, an exhaustive search over all possible values for the upper two digits of p (or more efficient techniques) could identify value for which the observed times correlate most closely with the expected number of subtraction operations. As with the Diffie-Hellman/non-CRT attack, once one digit of p has been found, the timing measurements could be reused to find subsequent digits.

It is not yet known whether timing attacks can be adapted to directly attack the mod p and mod q modular exponentiations performed with the Chinese Remainder Theorem.

8 Timing Cryptanalysis of DSS

The Digital Signature Standard[5] computes $s = (k^{-1}(H(m) + x \cdot r)) \bmod q$, where r and q are known to attackers, k^{-1} is usually precomputed, $H(m)$ is the hash of the message, and x is the private key. In practice, $(H(m) + x \cdot r) \bmod q$ would normally be computed first, then is multiplied by $k^{-1} \pmod q$.

If the modular reduction function runs in non-fixed time, the overall signature time should be correlated with the time for the $(x \cdot r \bmod q)$ computation. The attacker can calculate and compensate for the time required to compute $H(m)$. Since $H(m)$ is of approximately the same size as q, its addition has little effect on the reduction time. The most significant bits of $x \cdot r$ are typically the first used in the modular reduction. These depend on r, which is known, and the most significant bits of the secret value x. There would thus be a correlation between values of the upper bits of x and the total time for the modular reduction. By looking for the strongest probabilities over the samples, the attacker would try to identify the upper bits of x. As more upper bits of x become known, more of $x \cdot r$ becomes known, allowing the attacker to proceed through more iterations of the modular reduction loop to attack new bits of x. If k^{-1} is precomputated, DSS signatures require just two modular multiplication operations, potentially making the amount of additional timing noise which must be filtered out relatively small.

9 Masking Timing Characteristics

The most obvious way to prevent timing attacks is to make all operations take exactly the same amount of time. Unfortunately this is often difficult. Making software run in fixed time, especially in a platform-independent manner, is hard because compiler optimizations, RAM cache hits, instruction timings, and other

factors can introduce unexpected timing variations. If a timer is used to delay returning results until a pre-specified time, factors such as the system responsiveness or power consumption may still change detectably when the operation finishes. Some operating systems also reveal processes' CPU usage. Fixed time implementations are also likely to be slow; many performance optimizations cannot be used since all operations must take as long as the slowest operation. (Note: Always performing the optional $R_i = (s_i \cdot y) \bmod n$ step does not make an implementation run in constant time, since timing characteristics from the squaring operation and subsequent loop iterations can be exploited.)

Another approach is to make timing measurements so inaccurate that the attack becomes unfeasible. Random delays added to the processing time do increase the number of ciphertexts required, but attackers can compensate by collecting more measurements. The number of samples required increases roughly as the square of the timing noise. For example, if a modular exponentiator whose timing characteristics have a standard deviation of 10 ms can be broken successfully with 1000 timing measurements, adding a random normally distributed delay with 1 second standard deviation will make the attack require approximately $(\frac{1000ms}{10ms})^2 (1000) = 10^7$ samples. (Note: The *mean* delay would have to be several seconds to get a standard deviation of 1 second.) While 10^7 samples is probably more than most attackers can gather, a security factor of 10^7 is *not* usually considered adequate.

10 Preventing the Attack

Fortunately there is a better solution. Techniques used for blinding signatures[1] can be adapted to prevent attackers from knowing the input to the modular exponentiation function. Before computing the modular exponentiation operation, choose a random pair (v_i, v_f) such that $v_f^{-1} = v_i^x \bmod n$. For Diffie-Hellman, it is simplest to choose a random v_i then compute $v_f = (v_i^{-1})^x \bmod n$. For RSA it is faster to choose a random v_f relatively prime to n then compute $v_i = (v_f^{-1})^e \bmod n$, where e is the public exponent. Before the modular exponentiation operation, the input message should be multiplied by $v_i \pmod n$, and afterward the result is corrected by multiplying with $v_f \pmod n$. The system should reject messages equal to 0 $(\bmod n)$.

Computing inverses mod n is slow, so it is often not practical to generate a new random (v_i, v_f) pair for each new exponentiation. The $v_f = (v_i^{-1})^x \bmod n$ calculation itself might even be subject to timing attacks. However (v_i, v_f) pairs should not be reused, since they themselves might be compromised by timing attacks, leaving the secret exponent vulnerable. An efficient solution to this problem is update v_i and v_f before each modular exponentiation step by computing $v_i' = v_i^2$ and $v_f' = v_f^2$. The total performance cost is small (2 modular squarings, which can be precomputed, plus 2 modular multiplications). More sophisticated update operations using exponents other than 2, multiplication with other (v_i, v_f) pairs, etc. can also be used, but do not appear to offer any advantages.

If (v_i, v_f) is secret, attackers have no useful knowledge about the input to the modular exponentiator. Consequently the most an attacker can learn is the general timing distribution for exponentiation operations. In practice, distributions are close to normal and the 2^w exponents cannot possibly be distinguished. However, a maliciously-designed modular exponentiator could theoretically have a distribution with sharp spikes corresponding to exponent bits, so blinding does not *provably* prevent timing attacks.

Even with blinding, the distribution will reveal the average time per operation, which can be used to infer the Hamming weight of the exponent. If anonymity is important or if further masking is required, a random multiple of $\varphi(n)$ can be added to the exponent before each modular exponentiation. If this is done, care must be taken to ensure that the addition process itself does not have timing characteristics which reveal $\varphi(n)$. This technique may be helpful in preventing attacks that gain information leaked during the modular exponentiation operation due to electromagnetic radiation, system performance fluctuations, changes in power consumption, etc. since the exponent bits change with each operation.

11 Further Work

Timing attacks can potentially be used against other cryptosystems, including symmetric functions. For example, in software the 28-bit C and D values in the DES[4] key schedule are often rotated using a conditional which tests whether a one-bit must be wrapped around. The additional time required to move nonzero bits could slightly degrade the cipher's throughput or key setup time. The cipher's performance can thus reveal the Hamming weight of the key, which provides an average of $\sum_{n=0}^{56} \frac{\binom{56}{n}}{2^{56}} \log_2 \left(\frac{2^{56}}{\binom{56}{n}} \right) \approx 3.95$ bits of key information. IDEA[3] uses an $f()$ function with a modulo $(2^{16} + 1)$ multiplication operation, which will usually run in non-constant time. RC5[7] is at risk on platforms where rotates run in non-constant time. RAM cache hits can produce timing characteristics in implementations of Blowfish[11], SEAL[9], DES, and other ciphers if tables in memory are not used identically in every encryption.

Additional research is needed to determine whether specific implementations are at risk and, if so, the degree of their vulnerability. So far, only a few specific systems have been studied in detail and the attacks against CRT/Montgomery RSA and DSS are currently theoretical.

Further refinements to the attack may also be possible. A direct attack against p and q in RSA with the Chinese Remainder Theorem would be particularly important.

12 Conclusions

In general, any channel which can carry information from a secure area to the outside should be studied as a potential risk. Implementation-specific timing

characteristics provide one such channel and can sometimes be used to compromise secret keys. Vulnerable algorithms, protocols, and systems need to be revised to incorporate measures to resist timing cryptanalysis and related attacks.

13 Acknowledgements

I am grateful to Matt Blaze, Joan Feigenbaum, Martin Hellman, Phil Karn, Ron Rivest, and Bruce Schneier for their encouragement, helpful comments, and suggestions for improving the manuscript.

References

1. D. Chaum, "Blind Signatures for Untraceable Payments," *Advances in Cryptology: Proceedings of Crypto 82,* Plenum Press, 1983, pp. 199-203.
2. W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, IT-22, n. 6, Nov 1976, pp. 644-654.
3. X. Lai, *On the Design and Security of Block Ciphers,* ETH Series in Information Processing, v. 1, Konstanz: Hartung-Gorre Verlag, 1992.
4. National Bureau of Standards, "Data Encryption Standard," Federal Information Processing Standards Publication 46, January 1977.
5. National Institute of Standards and Technology, "Digital Signature Standard," Federal Information Processing Standards Publication 186, May 1994.
6. P.L. Montgomery, "Modular Multiplication without Trial Division," *Mathematics of Computation,* v. 44, n. 170, 1985, pp. 519-521.
7. R.L. Rivest, "The RC5 Encryption Algorithm," *Fast Software Encryption: Second International Workshop, Leuven, Belgium, December 1994, Proceedings*, Springer-Verlag, 1994, pp. 86-96.
8. R.L. Rivest, A. Shamir, and L.M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, **21**, 1978, pp. 120-126.
9. P.R. Rogaway and D. Coppersmith, "A Software-Optimized Encryption Algorithm," *Fast Software Encryption: Cambridge Security Workshop, Cambridge, U.K., December 1993, Proceedings*, Springer-Verlag, 1993, pp. 56-63.
10. RSA Laboratories, "RSAREF: A Cryptographic Toolkit," Version 2.0, 1994, available via FTP from rsa.com.
11. B. Schneier, "Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish)," *Fast Software Encryption: Second International Workshop, Leuven, Belgium, December 1994, Proceedings*, Springer-Verlag, 1994, pp. 191-204.