# Cryptanalysis of Diffie-Hellman, RSA, DSS, and Other Systems Using Timing Attacks

Paul C. Kocher[*]

EXTENDED ABSTRACT (7 December 1995)

**Since many existing security systems can be broken with timing attacks, I am releasing this preliminary abstract to alert vendors and users. Research in this area is still in progress.**

**Abstract.** Cryptosystems often take slightly different amounts of time to process different messages. With network-based cryptosystems, cryptographic tokens, and many other applications, attackers can measure the amount of time used to complete cryptographic operations. This abstract shows that timing channels can, and often do, leak key material. The attacks are particularly alarming because they often require only known *ciphertext*, work even if timing measurements are somewhat inaccurate, are computationally easy, and are difficult to detect. This preliminary draft outlines attacks that can find secret exponents in Diffie-Hellman key exchange, factor RSA keys, and find DSS secret parameters. Other symmetric and asymmetric cryptographic functions are also at risk. A complete description of the attack will be presented in a full paper, to be released later. I conclude by noting that closing timing channels is often more difficult than might be expected.

## 1 Introduction

Cryptosystem implementations often take 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). Although intuition might suggest that only a small amount of information, such as the Hamming weight of the key, would be revealed, timing measurements can often be analyzed to find the entire secret key.

[*] Independent Cryptography Consultant, P.O. Box 8243, Stanford, CA 94309, USA. E-mail: `pck@cryptography.com`, FAX: 1-(415)-321-1483.

## 2 Timing cryptanalysis of fixed-exponent Diffie-Hellman

If a party uses the same secret exponent ($x$) for multiple Diffie-Hellman[2] key exchanges, the exponent can often be found from timing measurements. (The attack does not work if a new exponent is generated for every exchange.) The attacker first observes $k$ operations, measuring the time ($t$) taken by the victim to compute each $z = (y^x \bmod n)$. The attacker is also assumed to know the $y$ values, the public parameter $n$, and the general design of the target system.

The attack is explained using the simple modular exponentiation algorithm below, but can be tailored easily to work with virtually any implementation that does not run in fixed time.

```
Algorithm to compute $R = y^x \bmod n$:
  Let $R_0 = 1$.
  Let $y_0 = y$.
  For $i = 0$ upto $(\text{bits\_in\_}x - 1)$:
    If (bit $i$ of $x$) is 1 then
      Let $R_{i+1} = (R_i \cdot y_i) \bmod n$.
    Else
      Let $R_{i+1} = R_i$.
    Let $y_{i+1} = y_i^2 \bmod n$.
  End.
```

The attack will allow 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.)

The attack is simplest to understand in an extreme case. Suppose the target system uses a modular multiplication function that is extremely fast in the vast majority of cases but occasionally takes much more time than an entire normal modular exponentiation.

Because the first $b$ exponent bits are known, the first $b$ iterations of the `For` loop can be completed by the attacker. However, the operation of the subsequent step depends on an unknown exponent bit. If the bit is set, $R_{b+1} = (R_b \cdot y) \bmod n$ will be computed. If it is zero, the operation will be skipped.

As indicated earlier, for a few $R_b$ and $y$ values the calculation of $R_{b+1}$ will be extremely slow, and the attacker knows which these are. If slow $(R_b \cdot y) \bmod n$ operations are always associated with slow overall times, exponent bit $b$ is probably set. If there is no relationship between time required to compute $R_{b+1}$ and the total processing time, the exponent bit is zero.

In practice, modular exponentiation implementations do not usually have such extreme timing characteristics, but do have enough variation for the attack to work. For each $y$, the attacker can estimate $d$, a measure of the time it would take for the $(R_b \cdot y) \bmod n$ calculation, if done. The attacker also knows the total time $t$. Because the first $b$ exponent bits are known, the attacker knows $R_b$ and can measure $c$, the amount of time required for the first $b$ iterations of the exponentiation loop.

Given these values for one timing measurement, the probability that exponent bit $b$ is set can then be found as follows:

$$P(\text{bit=1} \mid y, t, c, d) \approx \frac{\phi(\frac{(t-c-d)-\mu(t-c-d)}{\sigma(t-c-d)})}{\phi(\frac{(t-c)-\mu(t-c)}{\sigma(t-c)}) + \phi(\frac{(t-c-d)-\mu(t-c-d)}{\sigma(t-c-d)})}$$

where $\phi(x) = \frac{e^{-x^2/2}}{\sqrt{2\pi}}$ is the standard normal function, $\mu(X)$ is the mean of $X$, and $\sigma(X)$ is the standard deviation of $X$.

The overall probability is:

$$P(\text{bit=1}) \approx \frac{\prod_{i=0}^{k-1} P(\text{bit=1} \mid y_i, t_i, c_i, d_i)}{\prod_{i=0}^{k-1} P(\text{bit=1} \mid y_i, t_i, c_i, d_i) + \prod_{i=0}^{k-1}(1 - P(\text{bit=1} \mid y_i, t_i, c_i, d_i))}.$$

For the attack to work, the actual probabilities do not need to be very large, since incorrect exponent bit guesses will destroy all future correlations. After a mistake, no new significant correlations will be detected, so one can identify that there has been an error, back up, and correct it.

Errors can thus be detected and corrected, since after a mistake no new significant correlations will be detected.

A preliminary implementation of the attack using the RSAREF toolkit[8] has been written. RSAREF scans across the exponent from MSB to LSB and does two exponent bits at a time, so corresponding adjustments to the attack were made. Using a 120-MHz Pentium$^{TM}$ computer running MSDOS$^{TM}$, a 512-bit modulus, and a 256-bit secret exponent, processing times ranged from 392411 $\mu$s to 393612 $\mu$s and closely approximated a normal distribution with a mean of 393017 $\mu$s and a standard deviation of 188 $\mu$s.

Timing measurements were taken for 2000 operations, each using a different starting value. (Each of these values would normally be a $g^x$ value computed by the other party.) Measurements of $c$ and $d$ were also made using RSAREF. Even though only 2000 samples were used, strong bit correlations (often 95 percent or better), were found. As noted before, the attack is self-correcting, so much weaker correlations would do. (The full version of this paper will show how the self-correction property works and will show how the probability equations can be adjusted for different implementations.)

The attack is computationally quite inexpensive. In particular, the attacker only has to do a small multiple of the number of modular multiplications done by server, not counting operations wasted due to incorrect exponent bit guesses.

The experiment above used random inputs, as might be gathered by a passive eavesdropper. As will be explained in the full paper, the required number of ciphertexts can often be reduced if attackers choose inputs known to have extreme timing characteristics under the target implementation.

## 3 Factoring RSA private keys using a timing attack

A private-key RSA operation consists of computing $m = (c^d \bmod pq)$, where $pq$ is publicly known but $d$ and the factorization of $pq$ are secret.

RSA signatures not performed using the Chinese Remainder Theorem can be attacked using identical techniques as were described for Diffie-Hellman in Section 2. If the Chinese Remainder Theorem (CRT) is used, a slightly different attack is required. With CRT, the first operations are to compute ($c \bmod p$) and ($c \bmod q$). As a rule, modular reduction operations do not run in constant time. The conceptually simplest attack is to simply choose values of $c$ 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 $c$ is less than $p$, computing $c \bmod p$ has no effect, while if $c$ is slightly larger than $c$, it is necessary to subtract $p$ from $c$ once.

RSAREF's modular reduction function with a 512-bit modulus on the same 120-MHz Pentium$^{TM}$ computer takes an average of approximately 17 $\mu$s less time if $c$ is slightly smaller than $p$, as opposed to slightly larger than $p$. Timing measurements of many ciphertexts can be combined to detect whether the chosen ciphertexts are larger or smaller than $p$.

More sophisticated variants of the attack can be even more effective. For example, if the approximate value of $p$ is first shifted several digits to the left, guesses of $p$ that are slightly too high will develop leading zero digits which will tend to increase the performance of the first modular multiply.

The Chinese Remainder Theorem RSA attack can also be adapted to use only known ciphertext, and thus can be used to attack RSA digital signatures. Modular reduction is done by subtracting multiples of the modulus. The number of subtraction steps is usually not constant. RSAREF's division loop, for example, integer-divides the uppermost two digits of $c$ 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 $c$. If the result is larger than $p$ (shifted left), another subtraction is performed. This additional subtraction affects the modular exponentiation time. The decision whether to perform an additional subtraction in the first loop of the division algorithm depends on $c$ and, in the vast majority of cases, only the upper two digits of $p$. The next division loop's decision depends on the upper three digits of $p$, and so forth. One can thus test candidates for digits of $p$ by measuring whether expected additional subtractions are reflected in the total modular exponentiation time. As with the Diffie-Hellman/non-CRT attack, once one digit of $p$ has been found, the same timing measurements can be reused to find subsequent digits. (Additional details will be presented in the full paper.)

## 4 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 be 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. (Since

$H(m)$ is of approximately the same size as $q$, its addition has little effect.) The most significant bits of $x \cdot r$ are 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 should thus be a correlation between values of the upper bits of $x$ and the total time for the modular reduction. A simple attack would be to use this correlation to test candidate values for 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 $x$. (The final version of this paper will describe DSS attacks in detail.)

Because DSS signatures require just two modular multiplication operations, variations in the computation time can be detected from relatively few timing measurements.

## 5 Other applications of timing attacks

Timing attacks can be effective against other cryptosystems as well. For example, the 28-bit C and D values in the DES[4] key schedule are sometimes rotated using a conditional to test whether a one-bit must be wrapped around. The additional time required to move nonzero bits will slightly degrade the cipher's overall performance, revealing the Hamming weight of the key. This 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 almost always run in non-constant time. RC5[6] is also at risk on platforms where rotates run in non-constant time. RAM cache hits can produce timing characteristics in implementations of Blowfish[9], SEAL[7], DES, and other ciphers if tables in memory are not used identically in every encryption.

## 6 Conclusions

Writing software that runs in fixed time can be hard, especially for platform-independent implementations, since many factors affect performance, including compiler optimizations, RAM cache hits, and instruction timings. Fixed-time code must always exhibit worst-case performance. A better alternative would be to use blinding techniques, such as those used for blind signatures[1], to prevent attackers from knowing actual inputs to the modular exponentiation function.

Random delays added to the processing time may increase the number of ciphertexts required, but do not completely solve the problem since attackers can compensate for the delay by collecting more measurements. (If enough random noise is added, the attack can become infeasible.) Computing optional $R_{i+1}$ calculations regardless of whether the exponent bit is set does not work and can actually make the attack *easier*; the computations still diverge but attackers no longer have to identify the *lack* of a correlation for adjacent zero exponent bits. If a timer is used to delay returning results until a pre-specified time, attackers may be able to monitor other aspects of the system performance to determine when the cryptographic computation completes.

## 7 Acknowledgements

I am grateful to Matt Blaze, Ron Rivest, and Bruce Schneier for a number of 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*, v. 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, NBS FIPS PUB 46-1, "Data Encryption Standard," U.S. Department of Commerce, Apr 1981.
5. National Institute of Standards and Technology, NIST FIPS PUB 186, "Digital Signature Standard," U.S. Department of Commerce, May 1994.
6. R.L. Rivest, "The RC5 Encryption Algorithm," *Fast Software Encryption: Second International Workshop, Leuven, Belgium, December 1994, proceedings*, Springer-Verlag, 1994, pp. 86-96.
7. 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.
8. RSA Laboratories, "RSAREF: A Cryptographic Toolkit," version 2.0, 1994, available via FTP from rsa.com.
9. 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.