Timing Attacks on RSA: Revealing Your Secrets through the Fourth Dimension

by Wing H. Wong

Introduction

If Alice wants to secure her home, she could buy high-quality locks and install several of them on her door. However, a clever burglar might simply unscrew the hinges, remove the door and walk away with all of Alice's valuables with minimal effort. This example of an indirect attack on household security is somewhat artificial but there exists a parallel in the world of encryption that is quite real. It is called the timing attack and it has been used to defeat some of the most popular encryption techniques.

Do you think your computer system is secure because you use strong cryptography? Do you know attackers may be able to attack your cryptography in a completely unexpected direction without directly breaking the cryptographic algorithm?

RSA [7] is a public key cryptographic algorithm that is widely used today to secure electronic data transfer. It is included as part of the Web browsers from Microsoft and Netscape and is used by the SSL (Secure Sockets Layer) which provides security and privacy over the internet. The RSA algorithm was invented by the team of Rivest, Shamir, and Adleman at MIT in 1978. Independently, Cliff Cocks discovered the same idea in the early 1970's [5]. A public key cryptosystem uses a one way function that is easy to compute in one direction and hard to compute in the reverse direction. For example, it is relatively easy to generate two prime numbers p, q and compute their product $N = pq$. But given N it is difficult to find its factors p and q. Encryption uses a public value, or key, which is distributed and known to anyone who wants to send a message. Decryption involves a related private key which is kept secret by the intended recipient and cannot be deduced from the public key. Public key cryptography works without requiring both parties involved to keep an agreed upon secret; the private key never needs to be sent to the sender.

RSA's public key includes a number N which is the product of two large prime numbers p, q. The strength of RSA comes from the fact that factoring large numbers is difficult. The best-known factoring methods are still very slow. For example, in a recent RSA challenge (August 1999), a 512-bit RSA challenge number was factored using 292 workstations and high-speed computers. The factoring took 35.7 CPU-years to accomplish which is equivalent to approximately 80,000 MIPS years. The feat required 3.7 months of calendar time [8]. Because so many people have been trying to find efficient ways to factor large numbers, so far without great success, we can probably assume that RSA is safe from a factoring attack for a typical key N of 1024 bits in length. RSA can be made more secure against factoring by increasing the key length to 2048 bits or more.

Despite this formidable mathematical strength, research has shown that it is feasible to recover RSA private keys without directly breaking RSA. This type of attack is known as a �timing attack� in which an attacker observes the running time of a cryptographic algorithm and thereby deduces the secret parameter involved in the operations. While it is generally agreed that RSA is secure from a direct attack, RSA�s vulnerability to timing attacks is not so well known and often overlooked. This paper explains the principles of timing attacks on RSA, summarizes the results of timing attacks implemented by various researchers, and discusses defenses against such attacks.

Timing Attacks on RSA

Kocher [4] was the first to discuss timing attacks. At the RSA Data Security and CRYPTO conferences in 1996, Kocher presented his preliminary result, warned vendors about his attack, and caught the attention of cryptographers including the inventors of the RSA cryptosystem. Timing attacks are a form of �side channel attack� where an attacker gains information from the implementation of a cryptosystem rather than from any inherent weakness in the mathematical properties of the system. Unintended channels of information arise due to the way an operation is performed or the media used. Side channel attacks exploit information about timing, power consumption, electromagnetic emanations or even sound to recover secret information about a cryptosystem [9].

Timing attacks exploit the timing variations in cryptographic operations. Because of performance optimizations, computations performed by a cryptographic algorithm often take different amounts of time depending on the input and the value of the secret parameter. If RSA private key operations can be timed reasonably accurately, in some cases statistical analysis can be applied to recover the secret key involved in the computations.

Before discussing timing attacks on RSA, we must first consider the mathematics of the cryptosystem. RSA is a public key cryptosystem that uses a public exponent e for encryption and a private exponent d for decryption. It uses a modulus N which is a product of two large prime numbers p and q , i.e., $N = pq$. The exponents e and d must be chosen to satisfy the condition $ed = 1 \bmod (p - 1)(q - 1)$. Then the RSA key pair consists of the public key (N, e) and the private key d. For example, if we select two prime numbers $p = 11$ and $q = 3$, then $N = 11 * 3 = 33$. Now compute $(p - 1)(q - 1) = 10 * 2 = 20$ and choose a value e relatively prime to 20, say 3. Then d has to be chosen such that $ed = 1 \bmod 20$. One possible value for d is 7 since $3 * 7 = 21 = 1 \bmod 20$. So we get the public key (N= 33, e= 3) and the corresponding private key $d = 7$. We discard the original factors p, q. Factoring breaks RSA because if an attacker can factor N into p and q, she can use the public value e to easily find the private value d.

To encrypt a plaintext message M, we compute $C = M^e \bmod N$, where C is the encoded message, or ciphertext. To decrypt the ciphertext C, we compute $M = C^d \bmod N$, which yields the original message M. Continue with our previous example where the public key (N= 33, e= 3) and the private key $d = 7$. Suppose we want to send a message $M = 19$. Encryption generates an encoded message $C = M^e \bmod N = 19^3 \bmod 33 = 28$. The sender would send the encrypted message $C = 28$. To decrypt message C, the receiver uses the private key d and computes $C^d \bmod N = 28^7 \bmod 33 = 19$, which must be the original message M. The decryption works thanks to a result from number theory known as Euler�s Theorem [2]. Another use of RSA is to generate digital signatures which serve to verify the source of the message. Signing uses the private key d and is the same mathematical operation as decrypting. The receiver uses the public key e and performs an encryption operation to verify the signature.

The operation in RSA that involves the private key is thus the modular exponentiation $M = C^d \bmod N$, where N is the RSA modulus, C is the text to decrypt or sign, and d is the private key. The attacker�s goal is to find d. For a timing attack, the attacker needs to have the target system compute $C^d \bmod N$ for several carefully selected values of C. By precisely measuring the amount of time required and analyzing the timing variations, the attacker can recover the private key d one bit at a time until the entire exponent is known. As Kocher [4] explained, the attack is in essence a signal detection problem. The �signal� consists of the timing variation caused by the target exponent bit, while the �noise� consists of the inaccuracies in the timing measurements and the random fluctuations in timing, particularly over a network. The number of timing samples required is thus determined by the properties of the signal and noise. In any case, the required sample size is proportional to the number of bits in the exponent d [4]. As there are only a limited number of bits in the private exponent d, the attack is computationally practical.

```
x = C
for j = 1 to n
    x = mod($x^2$, N)
```

```
        if $d_j$ == 1 then
            $x$ = mod($xC$, $N$)
        end if
    next $j$
    return $x$
```

Figure 1: Square and multiply algorithm.

The modular exponentiation in RSA consists of a large number raised to a large exponent which is a time consuming operation. A simple and efficient algorithm for computing C^d mod N is the square and multiply algorithm as shown in Figure 1, where $d = d_0d_1$�d_n in binary, with $d_0 = 1$. For example, an exponent value of 20 is 10100 in binary. Without leading zeros, d_0 is always 1. Notice that 20 can be built up one bit at a time from the left to right as (0, 1, 10, 101, 1010, 10100) = (0, 1, 2, 5, 10, 20). In other words, the exponent 20 can be constructed by a series of steps, where in each step, we double the number by shifting one bit to the left, and add 1 if the next binary bit is 1. To build up 20, the steps are

$1 = 0 * 2 + 1$
$2 = 1 * 2$
$5 = 2 * 2 + 1$
$10 = 5 * 2$
$20 = 10 * 2$.

Now suppose we want to compute 5^{20} mod 33, an efficient way would involve squaring which in effect doubles the exponent, and multiplying which is equivalent to adding 1 to the exponent. This is how the square and multiply algorithm works.

$5^1 = (5^0)^2 * 5^1 = 5 \bmod 33$
$5^2 = (5^1)^2 = 5^2 = 25 \bmod 33$
$5^5 = (5^2)^2 * 5^1 = 25^2 * 5 = 3125 = 23 \bmod 33$
$5^{10} = (5^5)^2 = 23^2 = 529 = 1 \bmod 33$
$5^{20} = (5^{10})^2 = 1^2 = 1 \bmod 33$

Typical RSA implementations use the Montgomery algorithm to perform the multiplication and the square operations. With the Montgomery algorithm, multiplications take a constant amount of time, independent of the size of the factors. But if the intermediate result of the multiplication exceeds the modulus N, an additional subtraction, called an extra reduction, is performed [3]. It is this extra reduction step that causes the timing difference for different inputs, and exposes information about the secret key.

Another typical way to optimize the RSA implementation is to use the Chinese Remainder Theorem (CRT) to perform the exponentiation. With CRT, the function $M = C^d$ mod N is computed by first evaluating $M_1 = C^d1$ mod N and $M_2 = C^d2$ mod N, where $d1$ and $d2$ are precomputed from d. M_1 and M_2 are then combined to yield M [1]. RSA with CRT makes the original attack by Kocher inoperative. Nevertheless, a timing attack can expose one of the factors of N, as illustrated by Brumley and Boneh [1].

A Simple Illustration

Here we give a simple illustration to show how a timing attack works. It is a generalization of the basic idea presented by Dhem et al. in [3].

Let C be a message to decrypt, d be the private key to be recovered, denoted by d_0d_1�d_n where $d_0 = 1$. The computation to be performed is C^d mod N. Suppose the square and multiply algorithm (Figure 1) is used to compute this modular exponentiation, and the mod(x, N) operation is implemented as in Figure 2.

```
mod(x, N)
if x >= N
   x = x % N
end if
return x
```

Figure 2: Pseudocode for the mod function

Consider the square and multiply algorithm in Figure 1. If $d_j = 0$ then $x = \mathrm{mod}(x^2, N)$, but if $d_j = 1$ then two operations, $x = \mathrm{mod}(x^2, N)$ and $x = \mathrm{mod}(xC, N)$, occur. In other words, the computation time should be longer when $d_j = 1$. Also notice in Figure 2 that the modular reduction, �%�, is only executed if the intermediate result of the multiplication is greater than the modulus N. Our attack exploits these two facts and uses two sets of messages, one for which the computation of $\mathrm{mod}(xC, N)$ would require a reduction and another for which it would not, to recover any bit d_j.

Assume we can get the system to decrypt the messages of our choice, which is often possible in real systems. We start attacking d_1 by choosing two messages Y and Z, where $Y^3 < N$ and $Z^2 < N < Z^3$.

Compare the operations of the square and multiply algorithm (Figure 1) on the two messages Y and Z when $j = 1$. If d_1 is really 1, the operation $x = \mathrm{mod}(x$ �$x^2, N)$ will be performed. Since $Y^3 < N$, the �%� operation does not occur for message Y. But since $Z^2 < N < Z^3$, the �%� operation occurs for message Z. On the other hand, if the actual value of d_1 is 0, the operation $x = \mathrm{mod}(x$ �$x^2, N)$ will not be performed and the computations on Y and Z should take about the same time. In other words, the run time of the algorithm will be longer for Z only if $d_1 = 1$. We can recover the bit d_1 using this fact.

But how much longer is a significant difference in run time? We�ll need to rely on statistics here. Instead of using a single message Y, we choose a set of messages Y_i, for $i = 0, 1,$ �, m-1, with $Y_i^3 < N$ and let y_i be the time required to compute $Y_i^d \bmod N$. Similarly, we choose a set of messages Z_i, for $i = 0, 1,$ �, m-1, with $Z_i^2 < N < Z_i^3$ and let z_i be the time required to compute $Z_i^d \bmod N$. If d_1 is actually 1, all the z_i's should be greater than the y_i's.

Compute the average timing $y = (y_0 + y_1 +$ � $+ y_{m-1}) / m$, and $z = (z_0 + z_1 +$ � $+ z_{m-1}) / m$. If $z > y + e$, where e can be determined empirically, we conclude that $d_1 = 1$; otherwise $d_1 = 0$. Once d_1 is known, we can attack d_2 in an analogous process by choosing Y and Z values that satisfy different criteria. The private key d can be recovered bit by bit in this manner.

The beauty of this attack lies in its simplicity. For the attack to work, we do not need to know the details of the implementation of the algorithm, and we do not need to know how to factor large numbers � all we need to know is that the exponentiation is done using square-and-multiply.

Implemented Timing Attacks

Dhem et al. mounted a similar attack against an earlier version of the CASCADE smart card [3]. A smart card is a kind of hardware security token. The fact that it is in the possession of the attacker makes it relatively easy to measure the running time of cryptographic operations and so smart cards are fairly vulnerable to timing attacks (as well as many other side channel attacks). Dhem�s implementation attacked the square rather than the multiply by observing the extra reduction at the square operation instead of the multiply operation. They reported breaking a 512-bit key in a few minutes with 350,000 timing measurements. A 128-bit key could be broken at a rate of 4 bits/sec using 10,000 samples. In cases where there are not enough samples to recover the complete key, the attack can still reveal a part of the key. Dhem et al. observed that with only half of the required number of samples, the attack was able to recover � of the key [3].

Systems using the Chinese Remainder Theorem (CRT) are not vulnerable to Kocher�s original attack or Dhem�s attack. However, Brumley and Boneh devised a timing attack against OpenSSL, an open source cryptographic library used in web servers and other SSL applications [6], and successfully extracted a factor of the RSA modulus N and therefore, the private key d [1]. Before this attack was publicized, it was generally believed that timing attacks would not work against general purpose servers because timing variations would mask the decryption times of the cryptographic algorithm. Nevertheless, Brumley and Boneh�s attack worked over a local network between machines separated by multiple routers. It was also effective between two processes running on the same machine and between two virtual machines on the same computer. They attacked three different OpenSSL-based RSA decryption applications: a simple RSA decryption oracle, Apache/mod_ssl, and Stunnel. They reported using about a million queries to remotely extract a 1024-bit key from an OpenSSL 0.9.7 server in about two hours [1]. On networks with low variance in latency such as a LAN or corporate/campus network, their attack is feasible [10]. In the case of a remote attack, the attacker must also account for variables other than network latency such as the load on the server. Thus, during periods of low activity, a server may be more vulnerable [10].

These attacks illustrate that cryptosystems that rely on modular exponentiation may be vulnerable to timing attacks. One such example is the Diffie-Hellman key exchange algorithm which establishes a shared secret to be used as a symmetric key. Brumley and Boneh�s attack on the Chinese remainder operation shows that RSA with CRT is also not secure against timing attacks. Any applications that perform RSA private key operations may be vulnerable. These applications include SSL/TLS-enabled network services, IPsec, Secure Shell (SSH1, ssh-agent), and TCPA/Palladium [10]. Their attack is also relevant to trusted computing efforts such as Microsoft's Next-Generation Secure Computing Base (NGSCB), which provides sealed storage for secret keys. By asking NGSCB to decrypt data in sealed storage, a user may learn the secret application key [1].

Possible Defenses

There are defenses against these timing attacks. The most widely accepted method is RSA blinding. With RSA blinding, randomness is introduced into the RSA computations to make timing information unusable. Before decrypting the ciphertext C, we first compute $X = r^e C \bmod N$, where r is a random value and e is the public exponent. We decrypt X as usual, i.e., compute $X^d \bmod N = r^{ed} C^d \bmod N = r\ C^d \bmod N$ again by Euler�s Theorem [2]. We then multiply the output by r^{-1} to obtain $C^d \bmod N$ which is the plaintext we want. Since a different r is used for each message, the original message is changed in a random way before the exponentiation operation. Thus, blinding prevents an attacker from entering a known input to the exponentiation function and using the resulting timing information to reveal the key. Blinding incurs a small performance penalty in the range of 2% to 10% [1].

Another possible defense is to make all private key operations not depend on the input. One possible way to accomplish this is to always carry out the extra reduction in the Montgomery algorithm even though the result may not be used. This modification should be relatively easy to implement and does not affect the performance by much [3]. Care will need to be taken to ensure that the extra reduction is not optimized away by the compiler.

One other alternative is to quantize all RSA computations, i.e. make them always a multiple of some predefined time quantum [1]. The major drawback of this approach is that all computations must then take the longest of all computation times, eliminating the possibility of performance optimizations.

Mozilla�s NSS crypto library uses blinding to defend against timing attacks. Most crypto acceleration cards also have implemented defenses against timing attacks [1]. Before the publication of Boneh and Brumley�s paper, OpenSSL 0.9.7 included RSA blinding as an option, though it was generally not turned on. In March 2003, the United States Computer Emergency Readiness Team (CERT) issued a vulnerability note (VU#997481) [10] discussing the attack by Brumley and Boneh, its impact, and its prevention. At the same time, the OpenSSL Project issued a security advisory and included a patch to switch blinding on by default.

Lessons Learned

Side channel attacks tell us that even if a cryptographic scheme is mathematically strong, it may not be secure in practice depending on the way it is implemented and on the design of the system as a whole. Cryptography should not be examined in isolation. The design of a secure system should encompass every aspect of the system, including the cryptographic and non-cryptographic aspects [9]. Attackers are like cheaters in a game; they don�t play by the presumed rules and they attack in ways that are sneaky and "unfair". Security engineers should constantly be on the lookout for unusual or unexpected attacks and be ready to re-secure a system when such an attack appears.

Conclusion

Cryptographic algorithms that rely on modular exponentiation such as RSA and Diffie-Hellman may be vulnerable to timing attacks. If the exponentiation operation that involves the secret key can be timed by an attacker with reasonable accuracy, the key can be recovered by using carefully selected input values, the number of which being proportional to the length of the key. Researchers have implemented timing attacks against an earlier version of the CASCADE smart card, and three different OpenSSL-based RSA decryption applications. They showed that it is feasible to recover the RSA private keys used in these systems. Defenses against such attacks are possible. Today, the most widely used method is RSA blinding which incurs a small performance penality of 2% to 10%. Timing attacks illustrate that attackers do not necessarily play by the presumed rules and they will always attack the weakest link in a system. Strong cryptography gives us security only if it is implemented and used in ways that complement its strength.

Acknowledgements

The author thanks Dr. Mark Stamp and Dr. Oliver Shih for their valuable comments and editorial suggestions.

Biography

Wing H. Wong is a graduate student at San Jos� State University. Her research interests include network security and bioinformatics. She enjoys playing tennis in her spare time.

References

1. D.Brumley and D. Boneh, "Remote timing attacks are practical", at crypto.stanford.edu/~dabo/papers/ssl-timing.pdf

2. D.M. Burton, *Elementary Number Theory* , second edition, Wm. C. Brown Publishers,1989

3. J.F. Dhem, F. Koeune, P.-A. Leroux, P. Mestr�, J.-J. Quisquater, and J.-L. Willems, "A practical implementation of the timing attack", at www.cs.jhu.edu/~fabian/courses/CS600.624/Timing-full.pdf

4. P.Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems", at www.cryptography.com/resources/whitepapers/TimingAttacks.pdf

5. S.Levy, "The open secret", *Wired* , issue 7.04, April 1999, at http://www.wired.com/wired/archive/7.04/crypto.html

6. OpenSSL Project. http://www.openssl.org

7. R. L. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems". *Comm. ACM* 21 (1978) no. 2, 120-126.

8. RSA Laboratories. http://www.rsasecurity.com/rsalabs/node.asp?id=2098

9. B.Schneier, "Risks of relying on cryptography", Inside Risks 112, *Communications of the ACM* , vol. 42, no. 10, October 1999, at www.schneier.com/essay-021.html

10. US-CERT Vulnerability Note VU#997481, at http://www.kb.cert.org/vuls/id/997481