Abstract

The RSA encryption algorithm has become one of the most important public-key encryption schemes in today's computing world. A vital performance boosting operation is the implementation of the Chinese Remainder Theorem to increase the speed of decryption. However, there are rumors and findings that suggest that this execution method, while providing significant performance benefits, is highly susceptible to side channel attacks that may reveal the private keys to an attacker. This would render the encryption useless, and allow the attacker access to important data. We will be researching these attacks on the Chinese Remainder Theorem and whether or not there exists a method of protecting against these attacks.

Montgomery Reduction

In arithmetic computation, Montgomery reduction is an algorithm that allows modular arithmetic to be performed efficiently when the modulus is large (typically several hundred bits). The Montgomery reduction algorithm was introduced by Peter Montgomery in 1985. A single application of the Montgomery reduction is faster than a naïve modular multiplication. A Montgomery step follows the form:

\[ C = a \times b \pmod{n}. \]

In the Montgomery reduction, there are more conversions needed to do a single step, which makes a single step actually slower than a naïve step. However, modular exponentiation can be implemented with Montgomery reduction and the only conversions are needed at the start and end of the sequence. The greater speed during the process far outweighs the time used converting at the beginning and end of the sequence.

Many cryptosystems such as RSA and DSA are based on arithmetic operations modulo a large number. The classical way of executing these computations are very expensive computationally, and much worse when the algorithm requires modular exponentiation. Using a Montgomery representation of the numbers allows calculating \( a^b \pmod{n} \) for any \( b \) with only two modular reductions. The Montgomery reduction implementation greatly increases the computations used within the cryptosystem.

Chinese Remainder Theorem

The original form of the theorem is a statement about simultaneous congruences in modular arithmetic. The use of the Chinese Remainder Theorem in RSA comes from the need within RSA to make many calculations modulo \( n \), where \( n \) is a product of
two large prime numbers $p$ and $q$. 1024, 2048 or 4096 bit integers are commonly used, making calculations very time consuming. By utilizing the Chinese Remainder Theorem, however, these calculations can be done in a much quicker time than if it was normal modular arithmetic. However, the tradeoff for more speed encrypting something is that by implementing the Chinese Remainder Theorem into RSA, RSA becomes more vulnerable to attacks such as side-channel attacks, fault attacks, etc. The different operations and different amount of operations of the Chinese Remainder Theorem within RSA provide enough information to onlookers to attempt to crack RSA through the CRT.

**Timing Attacks on RSA Using CRT and Montgomery Reduction**

The Chinese Remainder Theorem, because it performs many actions which are directly related to the prime factors of the key, can be taken advantage of to reveal the keys. When an input is given to the decryption that is close to one of the prime factors, there will be fewer modular reductions performed by the CRT process. When the input is less than one of the prime factors, no modular reductions will be performed for this. As a result, an input that is less than the prime factors will take the least time to process; one can slowly raise the inputs until increases in time are seen. Thus, through a large number of guesses, a statistical analysis can be performed and potentially reveal one of the prime factors, thus breaking the encryption.

Montgomery Reduction, while providing an excellent way to speed up modular multiplication, also can be easily analyzed to reveal a prime factor. It has been found that Montgomery Reduction may perform an extra reduction at the end of multiplication. As a result, one can time calculations and determine when this reduction is, or is not, being performed. It has been seen that the probability of an extra reduction is directly related to how close to one of the prime factors the input is. When the input approaches a prime factor, the probability of an extra reduction rises. When the input is equal to one of the prime factors or a multiple of it, the frequency or extra reductions being performed drops significantly. The probability of an extra reduction has been found to be $\Pr[\text{Extra Reduction}] = (g \mod q)/2R$, where $R$ is the Montgomery Reduction modulo value, $g$ is the input, and $q$ is the prime factor being revealed. With a large enough number of guesses, we can determine the prime factors of the key and defeat the encryption.

David Brumley and Dan Boneh used these in addition to other features of the OpenSSL RSA implementation to successfully guess the key of a web server in 2 hours with around 350,000 guesses. This shows that timing attacks can be highly successful against implementations of RSA that do not have sufficient defenses against side channel attacks.
How to Prevent Timing Attacks

There are multiple ways to fight against timing attacks against RSA. These are different forms of blinding. Blinding refers to the acting in a way so as to make timing information unreadable or unusable. Some methods of blinding include elimination of conditional executions, the performance of data-independent calculations, modification of modular behaviors, and the equalization of operation times.

One method of blinding is to remove the executions of statements conditionally. Many calculations, such as modular multiplications or exponentiations, take varying amounts of time to execute and are executed only under specific conditions. If we execute both multiplications and exponentiations at every step, and only make use of the results that are wanted by the current step of our operation, we can effectively make it impossible for the attacker to differentiate between either of these steps. This makes timing results from these heavily attacked steps useless. This may add to the complexity of the implementation. However, it won’t be severely detrimental to performance.

Another method of blinding is the performance of data-independent calculations at each step of modular exponentiation. In doing this, we execute an additional calculation which would be essentially random and alter the execution times in a way so as to make them unusable. At each step of the CRT calculations, we can calculate $x^r \mod h$ where $r$ and $h$ are random numbers, and $x$ is the message being used in the CRT process. This will slow computation times slightly, and makes timing impossible.

Many timing attacks focus on the differentiation between the times taken to calculate modular exponentiations and modular multiplications. One way to greatly hinder timing is to modify your algorithms of multiplication and exponentiation, in addition to hardware. The purpose of such modifications would be to ensure that both calculations take the same amount of time to execute, thus making it impossible for an attacker to determine which action is being performed at a given time.

Finally, the most effective, yet least practical method of blinding is to ensure that all calculations take the same amount of time. This would require that all calculations take an amount of time equal to the execution time of the slowest calculation that would be performed. This defeats the purpose of speedups provided by CRT and Montgomery Reduction, however makes timing impossible.
References

1. Remote Timing Attacks are Practical by David Brumley and Dan Boneh
2. Introduction to Side Channel Attacks
3. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems by Paul C. Kocher
4. Side-Channel Attacks by Neal Koblitz and Alfred J. Menezes