Attacks on RSA

John McKeown, Grant Page, Ben Schoenfeld

January 24, 2017

Abstract

"Secrets are hard to keep" [Mck16].

RSA is a profound cryptosystem based on sound mathematics. It has been shown that breaking properly implemented RSA is just as hard as factoring (Which is currently seen as a hard problem for suitably large numbers and classical computers). Despite this theoretical security, problems still arise in the implementation. The remainder of this paper aims to illustrate some of the decisions that must be made correctly in implementation in order to ensure the security of RSA.

1 The RSA Algorithm

We begin by outlining how RSA works and the notation we will be using. RSA requires a public and private key sine it is an asymmetric encryption algorithm. Let $N = pq$ be the product of two large prime numbers. Let e, d be two integers such that $ed \equiv 1\ (mod\ \phi(N))$. Since p, q are primes $\phi(N) = (p-1)(q-1)$. N is called the modulus, e is called the encryption exponent, and d is called the decryption exponent. We will denote the public key as the pair (N, e) which is used to encrypt message. The private key is the pair (N, d) which is used to decrypt message. The private key should only be known by the recipient of the message, this way only the correct recipient can decrypt the message.

The message is an integer $M \in \mathbb{Z}_{\mathbb{N}}^*$. To encrypt M one computes $C = M^e\ (mod\ N)$. To decrypt the ciphertext, the proper reciever computes $M = C^d\ (mod\ N)$.
RSA relies on the fact that factoring N is very difficult for sufficiently large N.

2 Factoring Attacks

Factoring attacks on RSA are referred to as brute force attacks. These attacks rely on factoring the modulus N into it's disctinct prime factors. If N can be factored then calculating $\phi(N) = (p-1)(q-1)$ is very easy which allows one to find the decryption exponent d by solving $ed \equiv 1\ (mod\ \phi(N))$. Brute force attacks do not currently pose a serious threat if RSA is used properly since no classical algorithm (an algorithm that can be implemented on a classical computer) has been published that can factor an integer in polynomial time [B+99]. This means as the size of N increases, the time it takes for N to be factored increases exponentially [B+99]. If a brute force attack can be done in polynomial time in all cases, then RSA will be considered broken and will no longer be a viable encryption algorithm. Although factoring attacks on RSA are not very likely to succeed at this point in time, knowing the efficiency of the best factoring algorithms provide benchmarks for certain requirements N should meet in order for RSA to be secure.

2.1 Noteable Factoring Algorithms

There are two noteable factoring algorithms used in practice today. One of them is the Quadratic Number Field Sieve (QNFS) and the other is the General Number Field

Sieve (GNFS). The QNFS is considered the fastest algorithm for factoring numbers with approximately 50-100 decimal digits. Once the numbers start to have more than 100 decimal digits, the GNFS tends to be faster than the QNFS [Bri98].

The QNFS is one of the best classical factoring algorithms we have to date. The QNFS is an optimized version of Fermat's factoring method. Fermet's method factors an integer n by writing n as a difference of squares. That is $n = x^2 - y^2$ where $x, y \in \mathbb{Z}$. Then $n = (x + y)(x - y)$. If $(x + y)$ or $(x - y)$ are not prime numbers, then Fermat's method can be repeated with those values.

The following is an example of Fermat's method being used to factor $n = 1003$. We start by choosing an initial x value. For Fermat's method, $x_0 = \lceil\sqrt{n}\rceil$. For our example, $x_0 = \lceil\sqrt{1003}\rceil = 32$. Now we construct a table consisting of x and $x^2 - n$ starting at 32 and incrementing x until $x^2 - n$ is a perfect square.

x	$x^2 - n$
32	21
33	86
34	153
35	222
36	293
37	366
38	441

Since $441 = 21^2$ we can write $1003 = 38^2 - 21^2 = (38 + 21)(38 - 21) = 17 * 59$. That is, the prime factorization of 1003 is $17 * 59$.

One way the QNFS optimizes Fermat's method is by choosing x values in a more strategic manner. Fermat's method starts at an initial value and increments by one until $x^2 - n$ is a perfect square. This can take a very long time. Instead, the QNFS only tries x values that are considered smooth. A smooth number is one that only has "small" prime factors [Bri98].

Just like the QNFS aims to optimize Fermat's method. The GNFS aims to optimize the QNFS in order to factor integers larger than 100 decimal digits. There are a lot of different implementations of the GNFS and the details are fairly advanced and beyond the scope of this paper. What is important is the efficiency of the GNFS. The largest RSA modulus that was successfully facotered using the GNFS was 768-bits, or 232 decimal digits [KAF+10]. According to Kleinjung et al. this took the equivalent of almost 2000 years of computing on a signle-core 2.2 GHz computer.

2.2 Mitigation

Althogh being able to factor in polynomial time would be a great achievement in Mathematics and Computer Science, not being able to allows for relatively simple solutions to mitigate factoring attacks. All we have to do is increase the size of the modulus N. The current standard for the size of N is 2048-bits, well beyond the current factoring record of 768-bits. To put the size of a 2048-bit integer in perspective, we can calculate how many decimal digits a 2048-bit number is by solving the following equation

$$\begin{aligned} 2^{2048} &= 10^N \\ \log_{10}(2^{2048}) &= \log_{10}(10^N) \\ 2048 \log_{10} 2 &= N \\ N &\approx 616.5. \end{aligned}$$

Since we can not have half a decimal digit, a 2048-bit integer corresponds to a 617 decimal digit integer. This is the size the modulus must be if you want an SSL (Secure Socket Layer) Certificate from a third party like digicert. According to digicert's website, an integer this large would take a little over 6.4 quadrillion years to factor based on the reserach done by Kleinjung et al.[Dig].

2.3 Problems in the Future

All the factoring algorithms mentioned above are classcial factoring algorithms. As mentioned earlier, there has been no classical factoring algorithm published that can factor in polynomial time. However, there is a quantum algorithm, namely Shor's algorithm, that can factor an integer in polynomial time. Implementation of this algorithm requires a quantum computer, which currently have many limitations due to cost and the current technology we have. So far the largest integer factored using Shor's algorithm is 21. Nonetheless, quantum computing will eventually break RSA. This has led to the creation of a large field called Post-Quantum Cryptography [Sev16].

3 Common Modulus

The common modulus attack uncovers the plaintext message without factoring N or finding the secret decryption exponent d. Imagine a scenario where Charlie would like to send the message M to Alice and Bob individually. To do this Alice gives Charlie her public key (N, e_1) and Bob gives Charlie his public key (N, e_2) where e_1 and e_2 are relatively prime. The problem arises because Alice and Bob are both using the same modulus N. Charlie sends $C_1 = M^{e_1}$ $(mod\ N)$ to Alice and $C_2 = M^{e_2}$ $(mod\ N)$ to Bob. Now suppose an eavesdropper Eve intercepts C_1 and C_2. Since $gcd(e_1, e_2) = 1$ Eve can use the Euclidean Algorithm to get integers x and y such that $1 = e_1x + e_2y$. Exactly one of x or y will be negative. Without loss of genearlity assume x is negative. Eve can now calculate

$$\begin{aligned} (C_1^{-1})^{-x}(C_2)^y &= C_1^x C_2^y \\ &= (M^{e_1})^x (M^{e_2})^y \\ &= M^{e_1x+e_2y} \\ &= M^1 \\ &= M\ (mod\ N) \end{aligned}$$

Thus, Eve has uncovered the plaintext message without factoring N or uncovering the decryption exponent d [Bau13].

3.1 Mitigation

In order to avoid this kind of attack, two individual users in the same party of communication should not use the same modulus [Bau13].

4 Man-In-The-Middle Attacks

Consider the following situation, You are a resident of the United States and you have a pen-pal in India. You want to tell him/her something in private and you are concerned that someone who isn't your pen-pal will see it. Since you are both mathematicians, you decide to use RSA to send encrypted messages to each other. This seems to be the ideal solution for preserving the secrecy of the message until we acknowledge the possibility of a malicious mail-courier who knows how RSA encryption and decryption works. This is a example of a Man-In-The-Middle Attack [B+99].

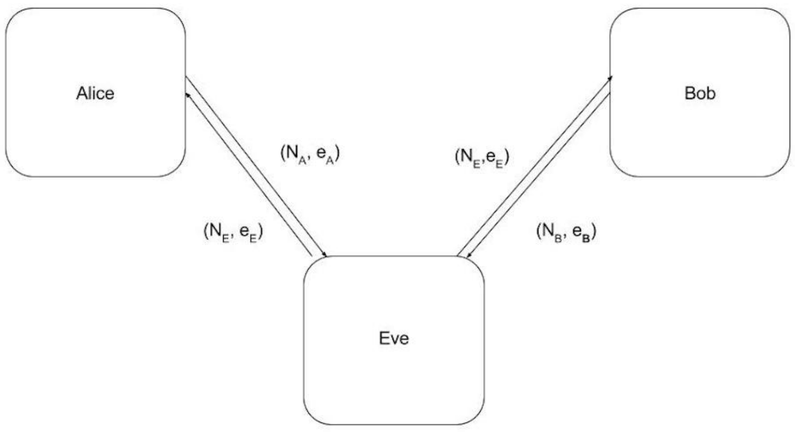

Figure 1: A model representing a Man In The Middle Attack. Eve intercepts Both Alice and Bob's public key and substitutes her public in. This way she is able to decrypt all intercepted messages using her private key.

This is a rather dangerous attack because it is difficult to detect. The Mail-Man can listen in on our conversations and collect information we disclose. He could also fabricate any information in order to take control of the communication channel. As we can clearly see we are at a huge disadvantage in exchanging our Public Keys. So the question remains: 'How do we prevent these attacks?'.

4.1 Mitigation

To prevent Man-In-The-Middle Attacks we are going to want to erase the possibility of a 'Man In The Middle'. By using a Trusted-Third-Party Source we can trust them as a service for exchanging keys. To continue our example we might want to use FedEx instead of the U.S. mail system since FedEx guarantees that employees are not allowed to view the package and they have a sender to receiver delivery system.

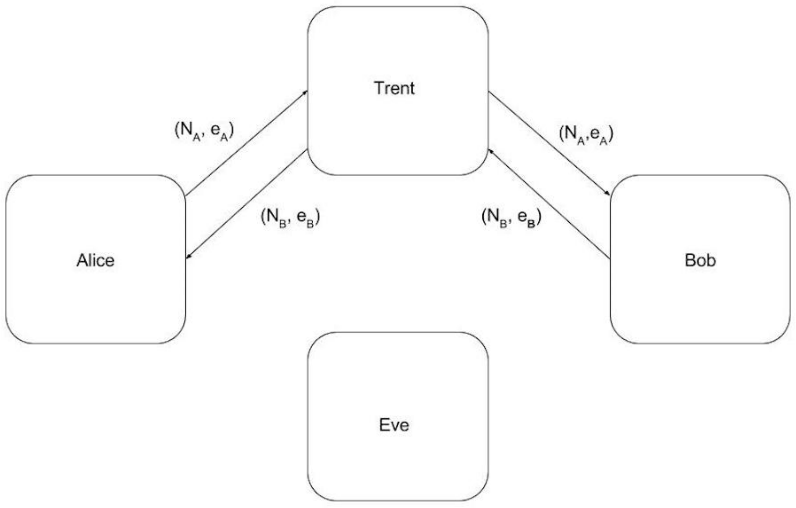

Figure 2: To avoid Man-In-The-Middle attacks we use a trusted third party. In this scenario Alice trusts Trent as a key distributor and Trent will ensure that the key will go to Alice and not Eve.

5 Searching the Message Space

A common mistake that users make when sending encrypted messages is sending short or low entropy messages [Bau13]. For example, Say I have an RSA channel that I use to send small keyword messages to people. If these words are less than 9 characters and we are using English as our plain text that gives us $26^9 = 5429503678976$ possible plaintext messages. If it takes 0.18 milliseconds to encrypt a message using RSA 1024, Then it should take 434360294318.08 milliseconds, or 5027, days to cycle through all possible messages on one computer. However, Say that we have 676 computers and we partition the work evenly so that separate machines won't encrypt the same plain-text message. If we do this it will only take us 7 days to cycle through all the keys.

5.1 Mitigation

The Solution to this attack is simple, simply increase the size of the message space. That is, instead of sending a message with less than 9 characters send a message with 18 or more characters. In general, it will always be possible for attackers to search the message space for the plaintext. We just need a large enough message space so that traversing it will take a larger amount of time that the universe can give [Bau13].

6 Ron was wrong, Whit was right

In this attack, Eve collects k public keys (N_i,e_i) using the internet. She then can choose to calculate the $\gcd(N_i, N_j)$ for any distinct $i, j < k$. If $\gcd(N_i, N_j) = 1$, then Eve is out of luck, but in the rare occasion (0.2%) that $\gcd(N_i, N_j) \neq 1$, then $\gcd(N_i, N_j) = p$ is a prime factor and can be used to find the private keys of those two entities! This is done of course by finding the other factor of N, so that we have both factors p and q. From there you derive the private key by solving $ed \equiv 1 \ (mod\ (p-1)(q-1))$ This is a serious issue and yet is hard to fix since each entity on the internet effectively has two secret values (a "p" and a "q") and the more entities that exist, the higher the probability that any two random values will be equal. This is known as the Birthday Paradox and is also important to keep in mind with cryptographic hash functions [Bau13].

6.1 Mitigation

There are two simple things we can do to mitigate this attack without completely scrapping RSA. First off, we could increase the size of the range within which we choose p and q in order to make p and q harder to share when randomly chosen. In addition, we should make sure that p and q are chosen "truly" randomly from the set of prime numbers within this range [Bau13].

7 Timing Attack

7.1 Introduction to the Attack

The following two attacks are not attacks specifically on RSA, but on implementation specific details. Consider the following scenario:

Alice has computed everything necessary for RSA communication including N,e and d and she has published N and e. Let $d = d_k d_{k-1} \dots d_1 d_0$ be the binary expansion of d. Alice will respond to messages she receives as soon as she decrypts them. A malicious (and very clever) attacker, Eve, realizes that she can learn d based on how long it takes for Alice to decrypt messages that Eve sends her.

To see how this could be accomplished we must first look at how $M = C^d \pmod{N}$ is actually calculated. The *Repeated Squaring Algorithm* takes advantage of the following equality.

$$M = C^d = C^{\sum_{i=0}^{k} 2^i d_i} = \prod_{i=0}^{k} C^{2^i d_i} \tag{1}$$

This leads directly to the following algorithm that can compute a modular exponentiation with at most $2k$ modular multiplications.

Repeated Squaring Algorithm:
Initially set $z = C$ and $M = 1$. For $i = 0, 1, \dots, k-1, k$ do the following:
(1) If $d_i = 1$ then set $M = Mz \pmod{N}$.
(2) Set $z = z^2 \pmod{N}$
At then end of this, $M = C^d \pmod{N}$.

What allows Eve to figure out bits of d, is the fact that step (1) is only executed for for bits that are ones and it takes time proportionate to the iteration you're currently on! This is amazing and is very dangerous for certain applications of RSA, such as smartcards. This would be difficult, but still not impossible to pull off over a network with unpredictable latency. Another thing to think about is that this doesn't only apply to explicit responses sent by Alice, but also to any sort of side-channel that could be observed as well [B+99].

7.2 Mitigation

This problem could be mitigated by modifying the server so that response to messages happens after some constant period of time has elapsed [B+99].

References

[B$^+$99] Dan Boneh et al. Twenty years of attacks on the rsa cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

[Bau13] Craig P. Bauer. *Secret History: The Story of Cryptology*. Chapman & Hall/CRC, 2013.

[Bri98] Matthew E Briggs. *An introduction to the general number field sieve*. PhD thesis, Virginia Polytechnic Institute and State University, 1998.

[Dig] Digicert. Check our numbers. https://www.digicert.com/TimeTravel/math.htm. Accessed: 2016-11-29.

[KAF$^+$10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K Lenstra, Emmanuel Thomé, Joppe W Bos, Pierrick Gaudry, Alexander Kruppa, Peter L Montgomery, Dag Arne Osvik, et al. Factorization of a 768-bit rsa modulus. In *Annual Cryptology Conference*, pages 333–350. Springer, 2010.

[Mck16] Jack Mckeown. Computer Security, 2016.

[Sev16] Evan Severson. personal communication, 2016.