ATTACKS ON RSA

KARAM GILL

Abstract. Over the many technological advancements, RSA is a key part in making sure that our messages are secure. In this paper we will go over the key attacks on RSA which have been going on for over 20 years such as common modulus attacks, blinding attacks, and meet in the middle attack. We will go over details about how each of these attacks work in depth and the mathematical proofs that allow these attacks to happen.

1. Introduction

The RSA cryptosystem is a key part of our daily lives and helps us by securing online communications and data by allowing secure key exchange.

The RSA cryptosystem was created by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT (Massachusetts Institute of Technology) in 1977. The name given was RSA as a result of the surnames of the founders (Rivest, Shamir, and Adleman). Without RSA there would be lack of security on websites and data breaches happening everywhere, possibly leaking important and confidential documents to the whole world. There have been many attacks on RSA in the past twenty years, and in this article we will go over some of the many attacks on RSA. Some examples of these attacks on RSA are the infamous 23andMe data breach, and the Dropbox data breach. In the October of 2023, the famous biotechnology company 23andMe suffered a data breach which resulted in malicious hackers gaining access to approximately 7 million peoples private information such as passwords. The 23andMe data breach most likely happened due to a breach in the RSA system. Another example of a data breach is the infamous Dropbox data breach. In this data breach hackers obtained Dropbox users passwords and leaked approximately 68 million credentials because of a breach in RSA. These are two of the many data breaches that have occurred because of insecure RSA and these data breaches are examples of consequences that could occur if the RSA isn't secure enough.

1.1. How RSA Works.

We will discuss how RSA works. We have the sender of the message, Alice, and the recipient Bob. We have a hacker named Charlie who is trying to intercept the message and find out private information. One of the key parts of the RSA system is N otherwise known as the RSA modulus. The RSA modulus, N, is generated by the product of two randomly chosen prime numbers p, q such that p and q approximately the same number of bits. Before 2002, N used to be around 512 bits (as a result, p and q were each 256 bits). In 2002, many realized that there was increasing need for higher security so N was changed to 1024 bits. Today, 2048 bit keys are common to ensure security and reduce vulnerability. The key parts of RSA are the public key which is (N, e) and is available for all to see, and you have the private key which is (N, d), only the recipient of the message can access the private key. These numbers e and d are inverses modulo $\phi(N)$ which is another way of saying $ed \equiv 1 \pmod{\phi(N)}$. Each message is converted to a number

Date: July 13, 2025.

M which is determined by the ASCII code. The first step is to encrypt M which is the same as finding the value $C \equiv M^e \pmod{n}$, this number C is called the cipher text. Now Bob has to decrypt the cipher text using his private key by computing $C^d \pmod{N}$. Note that the number that Bob receives is $M^{ed} = M^{k\phi(N)+1}$ for some positive integer k since $ed \equiv 1 \pmod{\phi(N)}$. Now $M^{\phi(N)} \equiv 1 \pmod{N}$ by Euler's Totient Theorem. So

$$M^{ed} \equiv M^{k\phi(N)+1} \equiv M^{k\phi(N)} \cdot M \equiv 1^k \cdot M \equiv M \pmod{N}$$

giving back the legitimate message to the receiver, Bob.

1.2. **Difficulty of Finding Private Exponent.** In this article we will talk about why is it hard to break RSA and what are some ways to break RSA. If Charlie, the hacker, finds the private key d, she can easily eavesdrop and find out the legitimate message. However there is immense difficulty in finding this private key d so another way that you can find the original message is by taking the eth root of C but as C and e are both large, it is near impossible to find the eth root even with advanced and fast computers. If you can find the factorization of N (namely if you can find p and q), you can find $\phi(N)$, and you know that $d \equiv e^{-1} \pmod{\phi)N}$ now using brute force methods or inputting the value of e and running a code to find d, after a while you will be able to find d. Note that finding the inverse of e is way easier than finding the eth root as it is way harder to compute large numbers raised to the power of another large number (as in the case of the e th root) compared to the simplicity of multiplying two numbers together to find an inverse. Think about it as multiplying two numbers compared to multiplying e numbers which takes approximately $\frac{e}{2}$ times longer which is massive considering that e is around 1024 bits.

1.3. **Modern Day Prevention of Attacks.** In almost all cryptosystems RSA is not used by itself. RSA by itself is considered insecure due to the amount of attacks that could be mounted on RSA. In fact, some attacks such as the meet in the middle attack could be mounted on modern day RSA and if there wasn't this extra padding then it would be easy to find and leak private keys and confidential information. A common form of padding that is used is called OAEP (Optimal Asymmetric Encryption Padding). OAEP adds randomness and complexity to messages before the encrypting process. OAEP uses hash functions and a random number, x in order to convert the plaintext into a more secure, padded message. To sum it up, OAEP provides a stronger and more secure way to encrypt data into RSA, especially against chosen-ciphertext attacks.

We will go over many attacks on RSA. These attacks include the blinding attack, common modulus attack, Wiener attack, Hastad's Broadcast attack, Coppersmith's theorem, Partial Key Exposure Attack, Franklin-Reiter Attack, Coppersmith Short Pad attack, Meet in the Middle Attack, Timing Attacks, Random Faults, Bleichenbacher Attack, Power Analysis, Cold Boot Attack, and Rabin-Miller Primality Testing. Furthermore, near the end of each attack we will provide an example with numbers for better understanding of how the attack works. Often we will use way smaller numbers than actual RSA works just for simplicity. Some conditions may not be satisfied as we might need larger numbers, but even with larger numbers the process will be the same.

1.4. **Overview of Attacks.** The blinding attack is when a malicious eavesdropper Charlie instead of asking for a signature on a message M, he asks for a signature on the message M' where M and M' are related and using this attack he can find the original message. The Common Modulus Attack occurs when two people Alice and Bob have the same RSA

modulus, and using this attack you can find the factorization of N and the private key d. The Wiener Attack occurs whenever the private exponent

$$d \leq \frac{1}{3} \cdot N^{\frac{1}{4}}.$$

Hastad's Broadcast Attack occurs when somebody foolishly sends the same message M to a large number of groups (in fact the number of groups i is greater than e). The Coppersmith's theorem targets a low public exponent and in this article we will not prove the whole Coppersmith theorem, however we will prove a few key claims that lead up to the proof of the theorem. The Partial Key Exposure Attack occurs when you are given the 16 most significant (left-most) or least significant digits (right-most) of the private key d (when there are 64 digits in d) The Franklin-Reiter Attack targets when two messages sent M_1, M_2 are related (namely by a function modulo N). The Coppersmith Short Pad Attack is an attack that demonstrates why you should not use a simple form of padding and proves given the public key and necessary encryptions, you can find the message M. The Meet in the Middle Attack is an attack that exposes the vulnerability of 2DES, an old cryptosystem (currently we use 3DES). After that we will talk a classic timing attack by Kocher that shows that by carefully measuring the time a smartcard takes for the decryption process, the private key d can be found. Many RSA forms compute the encryption both modulo p and q then use Chinese Remainder Theorem to find the value of $M^d \pmod N$ instead of computing modulo N. However, if one of the ciphertexts is signed incorrectly, devastating results could occur which we will show in the random faults section. After that attack we will discuss the Bleichenbacher attack which exploits a vulnerability in the RSA PKCS encryption. Then we will talk about power analysis which shows that we can find important information by checking and graphing the power levels that a processor has over a certain time period. The last attack we will talk about is the cold boot attack, we will briefly talk about this attack that shows how we can store data for hours after deletion. The last section will be about Rabin-Miller primality testing which shows how we can quickly and efficiently test if a number is prime or composite. In each attack we will talk about the process behind each attack and connect it back to the big picture, the world we currently live in. In some sections we will talk about how the attack is prevented in the modern day world.

1.5. **Chinese Supercomputer Breaks RSA.** On June 24th, 2025 a Chinese quantum computer broke RSA encryption threatening the online security of the whole world. The software company D-Wave systems used a quantum annealing processor (quantum annealing is the computational process of using quantum mechanics to optimize complex problems) to factor a 22 bit RSA key. This may not seem like much as 22 bits is only a few million if you think about it, however, quantum annealers couldn't break keys of size greater than 19 bits so this was a small but impactful breakthrough. This discovery has a big impact on us as it was one of the first times quantum computers could possibly become a threat. This threat also displays threat against the SPN (Substitution-Permutation Network) system which was considered secure before this research and discovery. This discovery could be the end of RSA later as if quantum annealers keep advancing at this rate soon they might be able to factor keys similar to the value of real RSA keys (around 2048 bits).

2. Blinding Attack and Common Modulus Attack

2.1. **Blinding Attack.** We again have the malicious eavesdropper Charlie, and Alice who is just minding her business. Charlie asks Alice for a signature of Charlie's message M but as Alice isn't stupid she declines. So Charlie asks for a signature of a completely innocent looking message M' where $M' = M \cdot r^e \pmod N$ where r is a randomly generated positive integer. Bob thinks that providing his signature to this message could cause no harm so he gives his signature S' to Charlie not knowing that he just made a grave mistake. Now note that $S' = (M')^d \pmod N$, now Charlie can find the original signature S as $S = \frac{S'}{r} \pmod N$ since $M' \equiv M \cdot r^e$ so

$$M'^d \equiv r^{ed} \cdot M^d \equiv r \cdot M^d \pmod N$$

Now note that $S \equiv M^d$ so $S' \equiv r \cdot M^d$ now if we divide by r we get $S \equiv M^d$ as desired. As we can see,

$$S^e \equiv \left(\frac{S'}{r}\right)^e \equiv \frac{S'^e}{r^e} \equiv \frac{(M'^d)^e}{r^e} \equiv \frac{M'}{r^e} \equiv M \pmod N$$

as desired and we have proven the process behind the blinding attack. Since we have proven the process, we will provide an example with numbers. Let $N, e, d = (1147, 13, 11)$. Again note that in this example $ed \neq 1 \pmod{\phi(N)}$ but this doesn't impact the example. Then we let $M = 1200$. We know that Charlie doesn't know d. Let $M' = 1200 \cdot 3^{13}$. Thus $S' \equiv (1200 \cdot 3^{13})^d \pmod{1147}$. Thus Charlie can find the original signature S, which is S'/r both of which we know. The result of the blinding attack can be devastating if it is executed successfully and can result in your private key d being leaked to the public. This is yet another reason why you don't give people you don't know your signature which is a common mistake made by people. However, this attack isn't that serious as most webs apply a one way hash which instead of computing the signature first computes a hash then signs the hash.

2.2. **Common Modulus Attack.** First we will prove a claim that will help us with the proof of the common modulus attack.

Theorem 1: Given the public key (N, e) and the private key d, Charlie can factor $N = pq$ efficiently; conversely, if Charlie can factor N then he can compute the private key d.

Proof: The converse is that given the factorization of N, the private key d can be found efficiently. We will also demonstrate a proof to this converse on top of the proof of the claim. First we will prove the converse.If you know the factorization of N, you know $\phi(N)$ thus you can find d since $d \equiv e^{-1} \pmod{\phi(N)}$. In order to prove the given claim we can let $k = ed - 1$, we know that $ed \equiv 1 \pmod{\phi(N)}$ thus $k \equiv 0 \pmod{\phi(n)}$. Now we can let $k = 2^x \cdot r$ where x, r are positive integers with r being odd. Note that k is a multiple of two as $\phi(N) = (p-1)(q-1)$ which is even as both terms are even. We can see that for every $m \in \mathbb{Z}_N^*$ we have $m^k \equiv 1$, by the Chinese Remainder Theorem we have exactly 4 square roots of 1 $\pmod{pq}$ namely $1, -1$ and $\pm l$ for some l. Note that this l must satisfy $l \equiv 1 \pmod p, -1 \pmod q$ or vice versa, otherwise the square wouldn't be 1 $\pmod{pq}$. In order to find the factorization of N, we take $\gcd(l-1, N)$, we know that $l - 1$ must be either a multiple of p or q so by taking the GCD, we can find either one of p and q. Finding the remaining prime is easy as you can just use division. We will see how this claim is helpful in the following common modulus attack.

For every persons RSA modulus you need to generate two primes each around the same size to multiply to N. One might suggest to fix the same N for everybody so you don't have

to go through the extra struggle of generating two new primes every time. However, this doesn't work well as we will demonstrate. You might think generating the same modulus is safe as an eavesdropper Charlie won't know the private key of Alice. But using Charlie's private and public key as we just proved in the claim, Charlie can factor the common modulus N. Using the converse of the claim we just proved, Charlie can find Alice's private key. Let's say that $N = 13 \cdot 17 = 221$. Charlie doesn't know that $N = 13 \cdot 17$. Let's say Charlie has private key 23 and public key 29 so using the claim Charlie can find that $N = 13 \cdot 17$. From here using the claim Charlie can uncover the private key of Alice. Now that we have gone over some basic attacks, we will get into the more advanced attacks.

3. Wiener Attack and Meet in the Middle Attack

3.1. **Wiener Attack.** In this section we will talk about the M.Wiener attack named after cryptologist Micheal J. Wiener which demonstrates why we shouldn't have a low private exponent.

M. Wiener: Let $n = pq$ with $q < p < 2q$. Let $d \leq \frac{1}{3} \cdot N^{\frac{1}{4}}$. Given (N, e) with $ed \equiv 1 \pmod{\phi(N)}$, Charlie can efficiently recover d.

Proof: We will use continued fractions to prove this theorem. Let k be the integer such that $ed - a\phi(N) = 1$ (there exists this k as e and d are inverses $\mod \phi(N)$). Note that

$$\left|\frac{e}{\phi(N)} - \frac{a}{d}\right| = \frac{1}{d\phi(N)}$$

as a result of dividing by $d\phi(N)$. Thus we can conclude that $\frac{e}{\phi(N)}$ and $\frac{a}{d}$ are similar (since $\frac{1}{d\phi(N)}$ is extremely small). Note that

$$\phi(N) = (p-1)(q-1) = pq - p - q + 1 = N - (p + q - 1).$$

Now we can see that $p + q - 1 < 3q < 3\sqrt{N}$ (where the last step follows since $q < p$), so we find out that $|N - \phi(N)| < 3\sqrt{N}$. Now we can replace the $\phi(N)$ with N to get

$$\left|\frac{e}{N} - \frac{a}{d}\right| = \left|\frac{ed - aN}{dN}\right| = \left|\frac{ed - a\phi(N) - aN + a\phi(N)}{dN}\right| = \left|\frac{1 - a(N - \phi(N))}{dN}\right|.$$

Note that

$$\left|\frac{1 - a(N - \phi(N))}{dN}\right| \leq \left|\frac{a \cdot 3\sqrt{N}}{dN}\right| = \left|\frac{3a}{d\sqrt{N}}\right|.$$

Now we know that $a\phi(N) < ed$, also $e < \phi(N)$ thus $a < d < \frac{1}{3}N^{\frac{1}{4}}$. Using this result,

$$\frac{3a}{d\sqrt{N}} < \frac{N^{\frac{1}{4}}}{d\sqrt{N}} = \frac{1}{dN^{\frac{1}{4}}} < \frac{1}{2d^2},$$

as desired. We can see that we could have made a stricter bound of $\frac{1}{3d^2}$ but we choose $\frac{1}{2d^2}$ since this is a classical example of an approximation relation. Using this relation we can test all values of $\frac{e}{N}$ that approximate $\frac{a}{d}$ (which is approximately $\log_2(N)$) to find the correct value of the private key d. For this example we will let $N = 1147 = 37 \cdot 31$ where $q = 31, p = 37$. We want $d \leq \frac{1}{3} \cdot 5.81$ so $d = 1$ (note that in this problem that is the only option for d as we can see this is very small compared to the large $N = 1147$. We won't go through the method again as it would be redundant but given e, by Weiner attack, Charlie can recover d.

Another way that you could deal with this problem is by setting a really big public exponent e, which would help reduce the risk if you still have a low d, however if e is really large, the encryption time would be too long.

3.2. **Meet in the Middle Attack.** In short form there is a DES (Data Encryption Standard) you can encrypt this DES as many times as necessary, as of now, Triple DES is used. There is a huge flaw in Double DES as it is susceptible to the meet in the middle attack. Firstly, I will provide some historical context on DES. The Electronic Frontier Foundation (EFF) claimed to have broken DES in 1998. So 2DES and 3DES were suggested to replace DES. Currently, 3DES is being used but is still slow.

In double DES, the encryption is done twice. In DES, the plaintext length is 64 bits and the key length is 56 bits. A common mistake that many make is that in double DES there are $2^{56+56} = 2^{112}$ keys that you need to try to ensure success. However, there is a simple way around trying this many that not many know about. We denote the ciphertext as C and the plaintext at P. Then we have the two keys k_1, k_2. The way that 2DES works is that you go from the plaintext, P then use k_1 and k_2 in that order to get C. This process can be reversed going from the ciphertext C, using k_2 and k_1 in that order, and getting the result P. In short form, $C = E_{k_2}(E_{k_1}(P))$ and $P = D_{k_1}(D_{k_2}(C))$.

The meet in the middle attack is a known-plaintext attack which means that you have some knowledge of the value of P. Once you find both keys, you can find all the information including the private key, ciphertext, and much more. Between E_{k_1} and E_{k_2} we have some number x. Also between D_{k_2} and D_{k_1}, we have this same number x. It is certainly possible to use a brute force method by going through all the possible 56 digit keys to find all possibilities of this number x from the side of E_{k_1} to the side of D_{k_1}. Once we brute force through all these possibilities we compare to see which number is common between the two lists to find the number in the middle. From here we can find both the private and public key. The name meet in the middle comes from the fact that we are meeting in the middle of E_{k_1} and E_{k_2} (same for the private exponents) to find the number in the middle which ultimately helps us find out the private keys. Triple DES is still susceptible to the meet in the middle attack however it is really hard to use the meet in the middle attack in Triple DES as the run time would be longer than most supercomputers can handle as there are 2^{56} the amount of possible combinations. We will not provide an example for this attack as it is difficult to define so many variables and undergo intensive calculations (like trying 2^{2n} combinations for some n).

4. Hastad's Broadcast Attack and Coppersmith

4.1. **Hastad Broadcast Attack.** We will go over an attack discovered by computer scientist Johan Hastad in 1988. First we will go over a simplified version of this attack. Let's say we have i groups, $G_1, G_2, \ldots, G_i$. Each party G_k has it's own public key (N_k, e_k). Alice wants to send an encrypted message, M to each of the groups. So Alice uses all the public keys to send ciphertext P_k to G_k. Assume that the value of M is less than the values of all the distinct RSA modulus of the i parties. To simplify this we assume that $e < i$ and that each e_i is equal to some constant e. The claim is that the malicious hacker Charlie can find efficiently recover M given that $i \geq 3$. We have the equations

$$C_1 \equiv M^e \pmod{N_1}, C_2 \equiv M^e \pmod{N_2}, \ldots C_i \equiv M^e \pmod{N_i}.$$

Using Chinese Remainder Theorem to all C_k we get that there exists a number A such that

$$A \equiv M^e \pmod{N_1 \cdot N_2 \ldots \cdot N_i}.$$

We know that $M^e < N_1 \cdot N_2 \ldots \cdot N_i$ since $e < i$ and $M < N_k$ for all k. Thus $M^e = A$, and we can recover this M by simply just taking the eth root of A. Now the attack that Hastad found out was a lot stronger than this attack and requires much more knowledge. Hastad's attack doesn't work in modern day RSA due to the padding and the fact that the public exponent is high. One of the main things that Hastad's relies on is a low public exponent. As we saw, if $e > i$, then we can't directly conclude that $M^e = A$ as M^e is not necessarily less than $N_1 \cdot N_2 \ldots \cdot N_i$. In modern day RSA, e is usually around 64000 so in order for Hastad's broadcast to work you would need to be sending the same message to over sixty-four thousand parties which would almost never happen. It is time to go over an example. Let's say there are 4 parties with public keys $(187, 3), (221, 3), (323, 3), (667, 3)$. Again, we have the equations

$$C_1 \equiv M^3 \pmod{187}, C_2 \equiv M^3 \pmod{221}, C_3 \equiv M^3 \pmod{323}, C_4 \equiv M^3 \pmod{667}.$$

Now we know that there exists $A \equiv M^3 \pmod{187 \cdot 221 \cdot 323 \cdot 667}$, we can find A through the Chinese Remainder Theorem so we just take $\sqrt[3]{A}$ to find M.

4.2. **Coppersmith.** We will go over one of the most important theorems that enables many low public exponent attacks just like the Hastad's Broadcast attack which we just went over.

(Coppersmith) Let $N \in \mathbb{Z}_{>0}$ be a positive integer and let $f(x) \in \mathbb{Z}[x]$ be a monic polynomial with degree d. Let $X = N^{\frac{1}{d}-\epsilon}$ for some $\epsilon \geq 0$. Given the pair (N, f) Charlie can find all $|x_0| < X$ such that $f(x_0) \equiv 0 \pmod{n}$.

We will not fully prove Coppersmith's as the proof requires an advanced method using the LLL (Lenstra–Lenstra–Lovász) which is a lattice basis reduction algorithm. Instead of proving the theorem we will prove a few key claims leading up to the proof of the theorem. First, we will provide an application of Coppersmith onto RSA as this theorem doesn't seem to directly relate to RSA. We will use a simplified version of RSA with smaller numbers so computation isn't difficult. We will take $N = 2026, e = 3, c = 1441$ where c is the ciphertext. We want to recover the message m. We know that $m^e \equiv c \pmod{N}$ so $m^3 \equiv 1441 \pmod{2026}$. Let $f(x) = x^3 - c = x^3 - 1441$. By Coppersmith, as long as $|x| < N^{\frac{1}{3}}$, we can find the x such that $f(x) \equiv x^3 - c \equiv 0 \pmod{N}$. This is the same as $x^3 \equiv c \pmod{N}$. We can see that this is the same equation as we had with m and by Coppersmith we can find all $x < \sqrt[3]{2026}$. This is how Coppersmith can apply in RSA.

Now it is time to prove one of the key claims that help with the proof of the theorem.

Theorem 2: Let $h(x) \in \mathbb{Z}[x]$ be a polynomial of degree d and let X be a positive integer. Suppose $||h(xX)|| < \frac{N}{\sqrt{d}}$. If $|x_0| < X$ satisfies $h(x_0) \equiv 0 \pmod{N}$, then $h(x_0) = 0$ holds over the integers.

Proof: For an $f(x)$ we can express $f(x) = \sum a_i x^i \in \mathbb{Z}$. Let $||f|| = \sum |a_i^2|$. If we can prove that $h(x_0) < N$, then using the condition that $h(x_0) \equiv 0 \pmod{N}$, we know that $h(x_0) = 0$ so we will try to prove $h(x_0) < N$.We can express $|h(x_0)|$ as $|\sum a_i x_0^i|$. So we have

$$|h(x_0)| = \left|\sum a_i x_0^i\right| = \left|\sum a_i \cdot X^i \cdot \left(\frac{x_0}{X}\right)^i\right|.$$

Let's say this summation has x terms. The absolute value of this sum is maximized when all of these x terms are the same parity. So,

$$\left|\sum a_i \cdot X^i \cdot \left(\frac{x_0}{X}\right)^i\right| \leq \sum \left|a_i \cdot X^i \cdot \left(\frac{x_0}{X}\right)^i\right| \leq \sum |a_i \cdot X^i|.$$

Using Cauchy Schwarz inequality, $\sum |a_i \cdot X^i| \leq \sqrt{d}||h(xX)|| < N$. Putting everything together,

$$|h(x_0)| = \left|\sum a_i x_0^i\right| = \left|\sum a_i \cdot X^i \cdot \left(\frac{x_0}{X}\right)^i\right| \leq \sum \left|a_i \cdot X^i \cdot \left(\frac{x_0}{X}\right)^i\right| \leq \sum |a_i \cdot X^i| \leq \sqrt{d}||h(xX)|| < N,$$

as desired.

This is the key fact/theorem that is needed before using the LLL method. After the application of the LLL method, there is one more key claim that is needed for the proof. We will state the key claim but not prove it. This claim can be left as an exercise for the reader if they wish.

(LLL) Let L be a lattice spanned by $< u_1, u_2, \ldots, u_2 >$. When $< u_1, u_2, \ldots, u_w >$ are given as an input, then the LLL algorithm outputs a point $v \in L$ satisfying $||V|| \leq 2^{\frac{w}{4}} \det(L)^{\frac{1}{w}}$.

Coppersmith's theorem will help in the proof of many later topics and attacks we will go over and might be one of the most important theorems that deals with a low public exponent.

5. Partial Key Exposure and Franklin-Reiter Attacks

5.1. **Partial Key Exposure Attack.** Partial key exposure happens when the malicious hacker Charlie can only find out a little of the private key d. For example, Charlie can somehow find out only $\frac{1}{4}$ or 16 of the bits in the private key d. Is it possible to recover the other 48 bits? Boneh, Durfee, and Frankel discovered and proved that as long as $e < \sqrt{N}$, it is possible to discover the rest of the private key given a small portion of its bits. We will prove that we can recover the rest of the digits.

Claim: Let $N = pq$ be a n bit RSA modulus. Then given the $\frac{n}{4}$ least significant bits of d or the $\frac{n}{4}$ most significant bits of d, one can efficiently factor N.

Proof: First we will define what we mean by least significant or most significant bits. By most significant bits we mean the bits that carry the most value which are the leftmost digits. By least significant we mean the rightmost digits. First we let k be the nonnegative integer such that $ed - k \cdot \phi(N) = 1$. We can express $\phi(N)$ as $(p-1)(q-1) = pq - p - q + 1$. So

$$ed - k(pq - p - q + 1) = 1.$$

Note that $0 < k < e$ since $d < \phi(N)$. Since $q = \frac{N}{p}$ we have

$$p(ed) - kp(pq - p - q + 1) \equiv p \pmod{2^{\frac{n}{4}}}.$$

Note that $-kp(-q) = kpq = Kn$ so

$$p(ed) - kp(pq - p - q + 1) \equiv p(ed) - kp(N - p + 1) + Kn \equiv p \pmod{2^{\frac{n}{4}}}.$$

Note that Charlie knows the $\frac{n}{4}$ least significant digits of d thus Charlie knows $d \pmod{2^{\frac{n}{4}}}$ so he knows

$$ed \pmod{2^{\frac{n}{4}}}.$$

Thus the only two variables left in the equation are p and k. Now running through all the e possible values of k, we can run through e quadratics in terms of p to find all possible values of

$p \pmod{2^{\frac{n}{4}}}$. In fact, the number of attempts that is required in order to find the factorization is $e\log_2(e)$. Now in this example we let $N = 13 \cdot 17 = 221$. We let $d = 11111\ldots1111$ where there are 16 ones on each side and thirty-two zeros in the middle. Charlie only knows the ones in the number and has no clue that there are 32 zeros in the middle. Then by the Partial Key Exposure Attack, they can recover the other 48 bits as long as $e < \sqrt{N} \approx 14.866$ so we let $e = 14$. Also we know that the number of attempts required is $14\log_2(14) \approx 53$.

5.2. **Franklin-Reiter Attack.** Now we will talk about the Franklin-Reiter attack. Franklin and Reiter found a useful attack which takes place when Alice delivers two related messages using the same modulus to Bob. If Alice sends distinct messages M_1, M_2 over to Bob, these messages M_1 and M_2 satisfy $M_1 \equiv f(M_2) \pmod{N}$ for a polynomial $f \in \mathbb{Z}$ where this polynomial is available to the public. Alice encrypts messages M_1 and M_2 to get C_1, C_2 and sends C_1 and C_2 over to Bob. Franklin and Reiter state that if the malicious hacker Charlie gains access to C_1, C_2 then Charlie can easily recover the messages M_1, M_2. We will prove a simplified version of this theorem by assuming the polynomial is linear and that $e = 3$.

Claim (Franklin-Reiter): Let $e = 3$ and (N, e) be an RSA public key.Let messages M_1, M_2 be two distinct messages satisfying $M_1 \equiv f(M_2) \pmod{N}$ for a linear polynomial $f = ax+b$ with $b \neq 0$. Given N, e, C_1, C_2, f a hacker Charlie can recover the messages M_1, M_2.

Proof: We know that $C_1 \equiv M_1^3 \pmod{N}$. We also know that M_2 and M_1 are roots of the functions

$$g(x) = f(x)^e - C_1 = f(x)^3 - C_1$$

and

$$h(x) = x^3 - C_2$$

respectively. Note that M_2 is a root of both these polynomials as $M_2^3 - C_2 = 0$ and $M_1^3 - C_1 = 0$ by definition. So $x - M_2$ is a factor of both $h(x)$ and $g(x)$. Now Marvin can find the greatest common divisor of the two functions by using the Euclidean algorithm (since he knows the two functions). If the greatest common divisor is linear then he knows that $x - M_2$ must be a divisor of the polynomial so he can find out M_2 thus he can recover M_1. So all we need to prove is that the greatest common divisor is linear. To prove this we will prove that there can only be one integer root modulo either p or q. We will first prove that cubes of integer roots are all the same modulo p and q. Note that both p and q must be $2 \pmod 3$ as $\phi(N) = (p-1)(q-1)$ can't be a multiple of $e = 3$ so $p-1, q-1 \equiv 1, 2 \pmod 3$ but p and q are primes so

$$p - 1, q - 1 \equiv 1 \pmod 3 \implies p, q \equiv 2 \pmod 3.$$

We let g be a primitive root of unity $\pmod p$, so the list

$$1, g, g^2, g^3, \ldots, g^{p-2}$$

covers the whole residue class modulo p. If we cube this list note that we still have the whole residue class modulo p as $(g^r)^3 \equiv g^r \pmod 3$ by Fermat's Little Theorem. Now note that $C_1 \equiv M_2^3$ so the other roots must satisfy $s^3 \equiv C_2$ thus the cubes must be equal. Let's say the cubes of g^k, g^m are equal so

$$g^{3k} \equiv g^{3m} \implies 3k \equiv 3m \pmod{p-1} \implies k \equiv m \mod p-1 \implies g^k \equiv g^m,$$

which means that these two roots are equal. If two roots are equal then the third root must be equal since by Vieta's, the product of the roots are

$$M_2^2 \cdot a = C_2 = M_2^3 \implies a = M_2,$$

for some third root a. If two of the roots are a, and one root is M_2 we have

$$M_2 \cdot a^2 = C_2 \implies a = M_2.$$

Now if all three roots are the same, by Vieta's the sum of the roots is 0 so $3M_2 = 0 \implies M_2 = 0$ which is clearly a contradiction. Thus there can be no roots with the same cubes $\pmod p$ so M_2 is the only integer root $\pmod p$. We can do the same thing with $\pmod q$ which implies that M_2 is the only integer root of $h(x)$. We can do the exact same thing for $g(x)$ to prove that the only integer root is M_2. So we can express $g(x) = (x - M_1)g_1(x)$ and $h(x) = (x - M_2)g_2(x)$ where $g_1(x), g_2(x)$ are both irreducible quadratics. If these quadratics are the same then $g(x) = h(x)$ which is clearly not true, so $g_1(x) \neq g_2(x)$ thus $\gcd(g(x), h(x)) = x - M_2$ which is a linear polynomial as desired. We will not provide an example for the Franklin-Reiter attack as the process is as written in the proof.

6. Random Faults and Timing Attacks I

Now we will move on into the side of RSA that tackles whenever there is a fault in the system itself, not only a breaking of the RSA.

6.1. **Random Faults.** First we will go over Random Faults. Many forms of RSA decryption and encryption often use Chinese Remainder Theorem to make decryption time faster. The computation that they speed up is $M^d \pmod N$. Obviously this computation is possible but will take a decent amount of time so we take M^d both $\pmod p$, $\pmod q$ and use Chinese Remainder Theorem to compute $M^d \pmod N$. The first step in this process is to compute the two ciphertexts, C_p, C_q where

$$C_p \equiv M^{d_p} \pmod p, C_q \equiv M^{d_q} \pmod q.$$

And,

$$d_p \equiv d \pmod{p-1}, d_q \equiv d \pmod{q-1}.$$

Define T_1 and T_2 to satisfy the equations

$$1 \pmod p, 0 \pmod q \text{ and } 0 \pmod p, 1 \pmod q$$

respectively. Now you can find the real ciphertext

$$C \equiv T_1C_p + T_2C_q \pmod N.$$

First, will prove that $C \equiv T_1C_p + T_2C_q \pmod N$. Note that

$$T_1C_p + T_2C_q \equiv T_1C_p \equiv C_p \equiv M^{d_p} \equiv C \pmod N.$$

Similarly,

$$T_1C_p + T_2C_q \equiv T_2C_q \equiv C_q \equiv C \pmod N.$$

Thus

$$C \equiv T_1C_p + T_2C_q \pmod p, C \equiv T_1C_p + T_2C_q \pmod q \implies C \equiv T_1C_p + T_2C_q \pmod N$$

as desired. The last step where we applied Chinese Remainder Theorem doesn't take up much time compared to the first applications of the Chinese Remainder Theorem. Now we will prove a simple claim considering the approximate run time of calculations when we take both $\pmod p$ and $\pmod q$ or when we just take $\pmod N$.

Theorem: If taking $\pmod p$ and $\pmod q$ separately then finding the ciphertext C takes time x, then evaluating $C \equiv M^d \pmod N$ takes approximately time $4x$.

Proof: First we will note that by definition of N, we know that p, q are both approximately half the size of N. Thus multiplication modulo p will take $\left(\frac{1}{2}\right)^2$ as much time as multiplication (mod N). Also, d_p is half the length of d so we talk $\frac{1}{2}$ the time from that so we take $\frac{1}{8}$th of the time. But this is only from p so we have to multiply by 2 (since we have both p and q) showing us that taking $\pmod p$ and $\pmod q$ separately takes $\frac{1}{4}$ of the time as simply evaluating $M^d \pmod N$. We will now go over a version of this attack discovered by Arjen Klaas Lenstra, a Dutch mathematician, cryptographer, and a computational number theorist. This attack happens when there is an error in the process of signing. Let's say C_p is signed correctly but C_q is signed incorrectly so we have an incorrect C_q'. Now we have a false signature

$$C' = T_1 C_p + T_2 C_q'.$$

Given Charlie knows M, Charlie knows that C' is false since $C'^e \neq M \pmod N$. But $C'^e \equiv M \pmod p$ since

$$C' \equiv T_1 C_p + T_2 C_q' \equiv C_p \pmod p.$$

However, $C'^e \neq M \pmod q$. Thus

$$C'^e - M \equiv 0 \pmod p$$

so you can find $\gcd(C'^e - M, N) = p$ hence the factorization of N is revealed. Using Theorem 1, we can find the private key d. Most of the time this attack doesn't work as Charlie wouldn't have access to the message M. If Alice or Bob has random padding incorporated then this attack wouldn't work so we are assuming that there is no padding involved prior to the attack. When using this attack Alice should check that everything is correct before releasing the message as random faults are common and can be hazardous if not checked.

To finish off this subsection we will provide two examples. One of them will be of the general procedure if everything goes correctly. The other will be if C_q is incorrect.

Let $N = 13 \cdot 17 = 221$ (so let $p = 13, q = 17$). Next, let $M = 9, d = 5$. For simplicity we let $d_p = d_q = d = 5$. We can see that the equations

$$d_p \equiv d \pmod{p-1}, d_q \equiv q \pmod{q-1}$$

are obviously satisfied. We know that the real ciphertext is

$$M^d \equiv 9^5 \equiv 42 \pmod{221}.$$

We see that $C_p \equiv 9^5 \equiv 3 \pmod{13}$. Also $C_q \equiv 9^5 \equiv 8 \pmod{17}$. Let $T_1 = 170$ and $T_2 = 52$. As we can see,

$$T_1 \equiv 1 \pmod{13}, 0 \pmod{17}$$

and

$$T_2 \equiv 0 \pmod{13}, 1 \pmod{17}.$$

Note that

$$T_1 C_p + T_2 C_q \equiv 170 \cdot 3 + 52 \cdot 8 \equiv 926 \equiv 42 \equiv C \pmod N$$

as desired. Now if $C_q' \equiv 9^4 \equiv 16 \pmod{17}$, we can see that C_q' is obviously wrong. So

$$T_1 C_p + T_2 C_q' \equiv 170 \cdot 3 + 52 \cdot 16 \equiv 510 \pmod{13} \equiv 3 \pmod{13}.$$

Now note that $ed \equiv 1 \pmod{\phi(N)}$ thus $ed \equiv 1 \pmod{192}$. From this we can see that $e = 77$ works so we let $e = 77$. Thus

$$C'^e \equiv 3^{77} \equiv (3^3)^{25} \cdot 3^2 \equiv 9 \equiv M \pmod{13}$$

as desired. Thus

$$C'^e - M \equiv 0 \pmod{13} \implies \gcd(221, 3^{77} - 9) = 13.$$

We will also check that $C'^e \neq M \pmod{17}$. Note that

$$3^{77} \equiv (3^{16})^4 \cdot 3^{13} \equiv 3^{13} \equiv (3^6)^2 \cdot 3 \equiv 3 \neq M \pmod{17}$$

as we expected (we used Fermat's Little Theorem to deduce that $3^{16} \equiv 1 \pmod{17}$ and you can manually compute that $3^6 \equiv 1 \pmod{17}$).

6.2. **Timing Attacks.** In this section we will talk about certain timing attacks. Firstly, we will go over the basic algorithm of an important timing attack. .You have a smartcard, which safely secures your private key and confidential information, so you would think that Charlie, the hacker cannot find out your private key. Well that is wrong, Paul Kocher discovered that given the amount of time it takes for the decryption process, Charlie can find the private key d. The process is as follows, let $d = d_k d_{k-1} d_{k-2} \dots d_0$ be the binary representation of d (where $d_0, d_1, \dots d_k$ donate digits either 0 or 1). We are going to use the repeated square algorithm which basically computes the ciphertext, C using at most $2n$ modular multiplications. This algorithm has a few steps to it so we will list them. Let $x = M$, and $C = 1$ (note that this C isn't the actual value of the ciphertext). Do the following steps for all integers i ranging from 0 to k. If $d_i = 1$, then change C to $C \cdot x \pmod{N}$, after that change x to $x^2 \pmod{N}$. After applying these steps for all d_i we get the desired plaintext C as we get the correct value of M^d. The way that this attack works is that Charlie requests the smartcard to sign a bunch of random messages $M_1, M_2, \dots M_n$ and then he measures the time T_i for each n messages that it takes to sign each of the messages. To start off the proof we will note that as d is odd, we know that $d_0 = 1$. Now we will continue with the procedure that we previously listed. When we start out, $C = M$ and $x \equiv M^2 \pmod{N}$. If $d_1 = 1$ then $C \equiv M \cdot M^2 \equiv M^3 \pmod{N}$, if $d_1 = 0$ then this product is not computed. We compute the amount of time t_i it takes the smartcard to compute

$$M_i \cdot M_i^2 \pmod{N}.$$

Note that Charlie measures all the t_i's before mounting this attack. Kocher found out that $d_1 = 1$ implies that T_i, t_i are related. For example, let's say for some i, we discover that T_i is lower than usual then t_i would also be lower than usual. This is not a strict correlation (there is not necessarily a polynomial that maps T_i to t_i for all i). However, if $d_1 = 0$, there is no correlation between T_i and t_i. Given these pieces of information, Charlie can compare the two times to see if there is a correlation, so Charlie can find the value of d_1. Using this method, Charlie can recover d_2 next, then d_3, and so on so Charlie can uncover the whole private key, d. There are two ways to prevent this timing attack. The first of which is used more commonly is to add appropriate buffer time. So all decryption take the same amount of time thus you cant uncover d by finding whether t_i and T_i are correlated as they are always the same. The other way is to use blinding. We went over blinding in an earlier section and we can use this blinding to help prevent timing attacks. Again we let

$$M' \equiv M \cdot r^e \pmod{N}$$

for some random positive integer r. Then we apply d onto M' to get $M'^d \pmod{N}$. The reason this works is that we make a new message, M' that Charlie is completely unaware of and we apply d to this message that Charlie doesn't know the value of so the timing attack cannot be mounted. To finish off this section we will give an example with $d = 5 = 101_2$.

So $d_0 = 1$ thus $C \equiv C \cdot M \equiv M \pmod{N}$ and $x \equiv M^2 \pmod{N}$. Now $d_1 = 0$ so $C \equiv M \pmod{N}, x \equiv M^4 \pmod{N}$. Now $d_2 = 1$ thus $C \equiv M \cdot x^4$ and $x \equiv x^8 \pmod{N}$. Thus,

$$C \equiv M \cdot M^4 \equiv M^5 \equiv M^d \equiv C \pmod{N}.$$

As we can see from this example, the process of the timing attack works on the example of $d = 5$.

7. Bleichenbacher and Coppersmith Short Pad Attack

These attacks will focus on padding of the RSA cryptosystem and some misuses of padding.

7.1. **Bleichenbacher Attack.** The first attack we will go over is Bleichenbacher Attack. In order to understand this attack we will have to learn what PCKS 1 is and how it operates. PCKS 1 stands for Public Key Cryptography Standard 1 and is an old way of padding messages. Let N have n digits in base 2. We pad the message M such that the message M has n bits (assume that originally M had m bits with $m < n$). Note that when we are adding these extra $n - m$ bits, almost all of the bits are randomly generated. The way that we pad is that we have a leading byte that is zero (otherwise the message might be more than the RSA modulus N). Then we have 16 more bits often donated as 02. After this we have another byte that can be anything and called the padding identifier as this byte helps identify what operation is being used. Then we have a string of randomly generated bits (which is as long as necessary to fulfill the condition $n = m$). Second to last, we have the separator which is one bit and indicates the end of the padding and the start of the actual message. At the end of the padded message is just the message itself. When the message is decrypted, the initial block is checked then the block with all the randomly generated blocks is removed. If this first byte isn't present then an error message would be sent back. Daniel Bleichenbacher, a Swiss cryptographer, discovered that using this error message, Charlie the malicious hacker can successfully decrypt any ciphertexts that he chooses.

Now we will discuss the procedure behind the attack. Let's say Charlie wants to decrypt a ciphertext C. Similar to the blinding attack, Charlie picks a random positive integer r and computes

$$C' \equiv r \cdot C \pmod{N}.$$

Then Charlie sends this message to Alice and Alice puts the message into a decryption machine. The machine either doesn't respond or responds with an error giving Charlie information about the first 16 digits. Namely he knows whether these first 16 digits are the same as 02 of the original message. This means that Charlie has an oracle (this is an entity that can decrypt RSA ciphertexts) to check whether the first 16 digits of $C'r \pmod{N}$ are the same as 02. Bleichenbacher proved that this oracle is sufficient enough for Charlie to decrypt the whole message C.

7.2. **Coppersmith Short Pad Attack.** Now we will go over Coppersmith's Short Pad Attack. Coppersmith strengthened the Franklin-Reiter Attack and proved a key result on padding. The Franklin-Reiter doesn't sound very practical, after all why would Alice send two related messages to Bob, that doesn't sound like the brightest idea. So Coppersmith proved a stronger generalization of the Franklin-Reiter Attack. As we saw in the previous section, padding is quite complicated and has a lot of components to it. Why would we need all these components, couldn't we just stick a few random bits to the front or the back of the message? This may seem like it works but in reality it doesn't and this method isn't

too hard to break as we will demonstrate. Let's say Alice sends Bob a message and Charlie eavesdrops gaining access of the message. Bob doesn't reply to Alice's message so Alice thinks that Bob didn't receive the message so she sends the same message again. Charlie now has access to two ciphertexts C_1', C_2' of the message M that was randomly padded. We will show that from this, Charlie can find the plaintext $P \equiv M^e \pmod{N}$.

We will now go over the algorithm and proof of the Coppersmith Short Pad Attack.

Theorem (Coppersmith Short Pad): Let (N, e) be a RSA public key where N has length n bits. Set $m = \lfloor \frac{n}{e^2} \rfloor$. Let M be a positive integer message with length at most $n - m$ bits. Define $M_1 = 2^m \cdot M + r_1$ and $M_2 = 2^m \cdot M + r_2$ where r_1 and r_2 are distinct integers with $0 \leq r_1, r_2 < 2^m$. If Charlie is given the public key (N, e) and the encryptions C_1, C_2 of M_1, M_2 he can efficiently recover M. Note that Charlie is not given either r_1 or r_2.

Proof: We will start by defining two functions. Let

$$f(x) = x^e - C_1$$

and

$$h(x, y) = (x + y)^e - C_2.$$

Obviously M_1 is a root of $f(x)$, and when $y = r_2 - r_1$ we will show that M_1 is also a root of $h(x, y)$. Note that

$$h(x, r_2 - r_1) = (x + r_2 - r_1)^e - C_2 = (x + M_2 - M_1)^e - C_2 = M_2^e - C_2 = 0,$$

to get to the last step we plugged in $x = M_1$. Now we define a polynomial

$$h(y) = \text{res}_x(g_1, g_2)$$

which has degree at most e^2. We know that

$$r_2 - r_1 < 2^m < N^{\frac{1}{e^2}}.$$

Since $r_1, r_2 < 2^m$ and $m = \lfloor \frac{n}{e^2} \rfloor$. From this we know that by Coppersmith's Theorem one can efficiently recover $r_2 - r_1$. After this application of Coppersmith's we use Franklin-Reiter to recover M_1, M_2 thus we can recover M.

To finish off this section we will provide yet another example for your better understanding. Let $N = 31 \cdot 37 = 1147$ and $e = 3$. Now note that N has 11 bits so $m = 1$ thus M has a length of at most $11 - 1 = 10$ bits. Let

$$r_1, r_2 = 0, 1$$

respectively as $r_1, r_2 < 2^m = 2$. Thus

$$M_1 = 2M, M_2 = 2M + 1.$$

We see that

$$f(x) = x^3 - C_1, h(x, y) = (x + y)^3 - C_2.$$

When $y = r_2 - r_1 = 1$ we can see that $h(x, y) = (x + 1)^3 - C_2$ and using the same algorithm that we used for the proof we can prove that 1 is a root of the second polynomial also.

8. Power Analysis and Cold Boot Attacks

In this section we will talk about two attacks that are less on the mathematical side and require less math. We will be talking about two classical examples of implementation attacks.

8.1. **Power Analysis.** First we will talk about power analysis. Power analysis happens by measuring the power of a microprocessor. At first this may seem harmless as how would we find and leak important information from a microprocessor just by measuring the power. If you have used your device before you can find that if you are actively running multiple applications at the same time, your battery or power drains faster than if you are idle or not using many applications. The same thing happens with the microprocessor, when it is constantly running RSA encryptions then it drains power faster than if it is idle. Let's say the microprocessor is performing an encryption. Then we can measure the power constantly to gain some important information about the encryption process (note that we won't always find all the information such as the ciphertext, private key, message, etc). To measure power consumption you insert a resistor (usually 50 OHMS) then we measure current versus time and graph it which can result in some key discoveries. Power analysis can reveal the list of instructions executed inside the microprocessor and can break cryptographic systems that rely on the execution being dependent on the data that is being processed. We will demonstrate some examples of these systems that can be broken through power analysis. The DES key schedule is very easy to break as the 28 bits of the key schedule can each be determined individually by power analysis. When we use power analysis, the graphs if the bit is 0 or if the bit is 1 will be distinct so by comparing graphs we can recover all the 28 bits. When we are performing exponentiation (whether its decryption or encryption) we can see how many multiplications are being performed to find out if the bit is 1 or 0 (note that if it is 1 we have increased run time so that's how we can find the distinction). We can find the exponent of the expression (which could be the private key if we are computing the message) by comparing the power consumption characteristics. Power analysis is easy to prevent and now most applications of RSA have implemented systems that make power variation sufficiently small making it near impossible to do the same analysis as we can if the power varies. To sum it all up the power analysis is a simple consequence of battery/power consumption at different rates. Next, we will go over yet another implementation attack of RSA.

8.2. **Cold Boot Attack.** It is time to go over an interesting implementation attack otherwise known as the cold boot attack. Dynamic Random Access Memory (DRAM) is a type of computer memory. We will not go over this attack in detail as understanding the details of this attack takes a long time and takes us of our main topic of RSA. Instead, we will briefly overview the key points of this attack. When your computers memory is completely erased, you might think that it is nearly impossible to retrieve the memory almost instantly without specialized experts. However, this is not true, as the data could be accessed for a few hours if the chips are kept cold. The Cold Boot Attack is an attack that exploits the DRAM system to recover important cryptographic documents such as keys.

9. Finding the Primes

In RSA as you have seen so far in the paper we must generate many large primes. For example, p and q are two important primes that must be generated to find $pq = N$. As you might know, the gap between primes increase drastically and as the numbers get large the probability that we pick a prime is pretty low. In fact if we are picking primes p and q that are around 1024 bits the probability that we actually get a prime is $\frac{1}{709}$. This is due to there being a formula that around a number N, the approximate probability that we get a prime

is $\frac{1}{\ln(N)}$. We will solve the equation

$$\left(\frac{708}{709}\right)^n = \frac{1}{2} \implies n \approx 491.$$

Note that this equation shows us that we need to test approximately 491 numbers. in order to get a 50 percent chance of finding a prime. This may not seem like much as for a computer 491 shouldn't be much, however we will demonstrate how the normal prime checking method isn't efficient enough.

The most commonly used method to check if a number is a prime is to run through all primes less than $\sqrt{N}$ to see if they divide N. Unfortunately, there are a lot of primes to test that are less than$\sqrt{p} \approx 2^{512}$. In fact there are approximately

$$\frac{N}{\ln(N)} = \frac{2^{512}}{\ln(2^{512})} \approx \frac{2^{512}}{355} > 2^{503}$$

primes that need to be tested for divisibility. As we can see manually running through these many primes would be too time consuming so how do we generate primes? One might say that you just use a supercomputer that runs for days, even weeks to find all primes then we just store this list in a computer. However, this is not feasible given our current storage capacity (this might be feasible in a few years, however key size probably will increase making this unfeasible again).

The way that RSA generates primes is through a technique called Miller-Rabin primality testing. The first step is to let $n-1 = 2^s \cdot m$ where s is a positive integer and m is odd (note that we are testing if n is a prime). We will take casework based on the value of s. Firstly, if $a^m \equiv \pm 1 \pmod{n}$ we declare that n is a prime. Why is this? Well,

$$a^{n-1} \equiv a^{m^{2^s}} \equiv 1 \pmod{n}.$$

Now note that we can't find another square root other than ± 1 as this is basically repeated squaring of something that's $\pm 1 \pmod{n}$ thus the square root is ± 1 so we know that n is a prime.

The next case is when s isn't 1. If s isn't 1 then we square a^m to achieve $a^{2m} \pmod{n}$. If $a^{2m} \equiv 1 \pmod{n}$ then n has to be composite as a^{2m} has a square root a^m that isn't ± 1. If $a^{2m} \equiv -1 \pmod{n}$ then n has to be prime. Why is this? Well we know that $a^{n-1} \equiv 1 \pmod{n}$ thus we can't find a square root of 1 other than ± 1.

Otherwise, unless $s = 2$, we obtain a^{4m} by squaring a^{2m}. Similar to last time, if $a^{4m} \equiv 1 \pmod{n}$ then we have discovered a square root other than ± 1 this root is namely a^{2m} so n is composite. We can tackle the other case when the expression is equal to -1 as we did previously. We see that now we have an algorithm that can conclude whether a number is a prime or not. However, we have to rely on the fact that some power of a is $\pm 1 \pmod{n}$. What if this power is never ± 1? If we haven't stopped after we reach value 2^s then we can conclude that n is composite. This is not a straightforward claim, and we will not prove it in this paper. This is left as an exercise for the reader.

The Miller-Rabin test can be confusing to understand so we will provide two examples to test if numbers are prime or composite. Let $a = 117$ and we are trying to see if 561 is prime. Note that

$$n - 1 = 561 - 1 = 560 = 2^4 \cdot 35$$

thus $s = 4, m = 35$. Thus we find the following values,

$$117^{35} \pmod{561} \equiv 417 \pmod{561}, 117^{70} \equiv 540 \pmod{561}, 117^{140} \equiv 441 \pmod{561}$$

and

$$117^{280} \equiv 375 \pmod{561}, 117^{560} \equiv 375 \pmod{561}.$$

We can see that none of these exponentiations result in $\pm 1 \pmod{561}$ as we stated previously if this happens then n is composite, thus we conclude 561 is composite. We can check that this is true as $561 = 3 \cdot 11 \cdot 17$. The last example we will tackle is when $a = 117$ again, and $n = 701$. In this case $701 - 1 = 700 = 2^2 \cdot 175$ thus $s = 2, m = 175$. Now it is time to do a similar computation as we did previously. The computation is as follows,

$$117^{175} \equiv 700 \equiv -1 \pmod{701}$$

thus as we saw previously this implies that 701 is prime which it indeed is. Once we find a prime through this message how do we indeed confirm it's a prime in an efficient method (just checking the veracity of the prime just in case an error is made in the process). We can do this through a formula called Fermat's Little Theorem. We will state and prove the formula in this section as well.

Theorem (Fermat's Little Theorem): If p is a prime number and a is a positive integer such that a is not divisible by p then $a^p \equiv a \pmod{p}$.

Proof: We will use proof by induction on a. If a is 1 then this statement is obviously true. For the inductive step we assume $a^p \equiv a \pmod{p}$ and we will prove that $(a+1)^p \equiv a + 1 \pmod{p}$. We will use the binomial theorem. Note that every term of $(a+1)^p$ is a multiple of p except the first term a^p and the last term 1. So

$$(a+1)^p \equiv a^p + 1 \equiv a + 1 \pmod{p},$$

as desired.

10. Conclusion

Now that you have thoroughly learned about almost all attacks on RSA we have to go back to the big question. Will RSA exist in the future or will we have to adapt a new form of online security?

To answer this we will look at the likely future of RSA in the short term, medium term, and long term. In the short term, RSA key lengths will most likely increase to 3072 bits then 4096 bits as computational advancements will probably occur making our 2048 bit key insecure. In the medium term (5 to 10 years) RSA will be getting closer to being broken as factorization algorithms of large numbers will advance (such as Shor's algorithm or quantum computing devices). Note that once we have the factorization of the RSA modulus using Theorem 1 we can find the private key d. In the long term (10+ years) many predict that RSA will be broken so we will have to adapt a new form of online security.

11. References

Boneh, Dan. *Twenty Years of Attacks on RSA*, https://crypto.stanford.edu/ dabo/papers/RSA-survey.pdf.

Staff, The420.in. "The End of RSA Algorithm? China's Quantum Breakthrough Sparks Global Panic." The420.in, June 25, 2025. https://the420.in/china-quantum-breaks-rsa-encryption/.

"China Breaks RSA Encryption with a Quantum Computer, Threatening Global Data Security." Earth.com. Accessed July 11, 2025. https://www.earth.com/news/china-breaks-rsa-encryption-with-a-quantum-computer-threatening-global-data-security/.

"Primality Tests." Number Theory - Primality Tests. Accessed July 11, 2025. https://crypto.stanford.edu