

Mathematics of RSA Encryption

Wyatt Rutledge

Department of Mathematics

Senior Thesis

Kelly Cline

May 1, 2026

Abstract

We explained the underlying mathematics that allow RSA encryption to function.

Mathematical proofs are used to demonstrate the security of RSA encryption. We created a mathematical proof for a potential vulnerability of RSA encryption, and we addressed one way in which it can be prevented. This is all used to demonstrate why RSA encryption is still in use today.

Table of Contents

<i>1: Introduction</i>	<i>4</i>
<i>2: Generating Keys</i>	<i>5</i>
<i>3: Distributing Public Keys</i>	<i>7</i>
<i>4: Proofs for Encrypting and Decrypting Messages</i>	<i>7</i>
<i>4.1: First Encryption and Decryption Proof</i>	<i>7</i>
<i>4.2: Second Encryption and Decryption Proof</i>	<i>12</i>
<i>5: Encrypting and Decrypting Messages</i>	<i>13</i>
<i>6: Proof for Vulnerability Due to a Low e</i>	<i>15</i>
<i>7: Vulnerability Due to a Low e</i>	<i>15</i>
<i>8: Conclusion</i>	<i>17</i>
<i>9: References.....</i>	<i>18</i>

1: Introduction

RSA encryption is a cryptographic system developed in 1977 by Ronald Rivest, Adi Shamir, and Leonard Adleman whose initials are the RSA in RSA encryption (Rivest et al., 1983). Despite its age, RSA encryption is still widely used with small improvements being added to the encryption method over time. This is possible because, despite the methods of RSA encryption being well documented, the math behind the encryption still ensures security. The vulnerabilities of RSA encryption are largely due to issues with implementation and taking shortcuts to make it faster.

In this paper we will be going over the mathematics behind RSA encryption and how it creates the process which is at the core of RSA encryption. The process of RSA encryption has 4 fundamental steps: generating the private and public keys, distributing the public keys, using the public keys to encrypt messages, and using the private key to decrypt messages. We will go through this process step by step with explanations and proofs of the mathematics behind it. Then we will go over a potential vulnerability of RSA encryption, how it mathematically occurs and one way in which it can be addressed.

2: Generating Keys

We will begin with the generation of keys. First, we will look at the relation between a few important values used in key generation, typically labeled n , p , and q , where p and q are prime numbers.

$$n = pq \quad (1)$$

Here n is the modulus of the public key while p and q are values used for making the private key. In practice, it is ideal to have a very large value of n for two reasons. Firstly, the size of individual messages, as binary numbers, must be smaller than the value n , with the computer having to divide the message into smaller pieces if a larger message needs to be sent. Second, a larger value of n makes it more difficult to find p and q for someone trying to break the encryption. Furthermore, p and q are typically chosen at random from a large number of possible prime numbers. The exact way p and q are chosen is not important, it simply needs to be a way a potential attacker trying to break the encryption could not deduce and take advantage of. The security of RSA encryption comes in large part from the fact that only knowing the value of n it is extremely slow, to the point of being infeasible, to find the values of p and q . If the method of picking p and q leaves too few possibilities and an attacker finds out how they were chosen, it could seriously compromise the security of the encryption.

Next, we need to actually generate our private key value, d , which fulfills equation (2).

$$\gcd(d, (p - 1) * (q - 1)) = 1 \quad (2)$$

The exact method used to generate d has varied over time, with later additions to the system finding ways to consistently find the smallest possible value of d (Jonsson and

Kaliski, 2003). It is not strictly necessary for the encryption to work that any particular value of d is used, but smaller values of d potentially result in the encryption and decryption being slightly less computationally intense. Various methods of generating d are still in use along with variations that add several extra values to both keys to result in more efficient computing (Jonsson and Kaliski, 2003).

With the private key, d , we can use it to calculate the final piece of the public key. The public exponent, e , is the multiplicative inverse of $d \bmod [(p-1)*(q-1)]$, that is to say it fulfills equation (3).

$$(e * d) \bmod [(p - 1) * (q - 1)] = 1 \quad (3)$$

Because of the nature of the modulus function, there are multiple acceptable values for e . There are three things to take into account when picking a value for e . First, e having a shorter length allows for faster encryption. Second, a very small value for e can potentially cause or exacerbate some security vulnerabilities. Third, a lower Hamming weight, which is the number of stored values that are not zero, also contributes toward faster encryption. Therefore, a commonly chosen value of e when possible is 65537, as in binary 65537 is written 10000000000000001. Picking a common value of e is not a security issue as it is the public exponent, a part of the public key, meaning it is a public value any user or attacker would have easy access to.

At this point, the values of p and q can be discarded; all actual calculations in the encryption system only need to use the private key, d , the public modulus, n , and the public exponent, e .

3: Distributing Public Keys

Distributing the public keys for RSA encryption does not need to be done securely, which is why they are called public keys. For any two computers sending each other messages with RSA encryption, they must each first generate their own private key and public key. They then send each other their public keys but not their private keys. Whenever one computer is encrypting or decrypting messages, the only data it has access to is its own private key, its own public key, and the other computer's public key.

4: Proofs for Encrypting and Decrypting Messages

4.1: First Encryption and Decryption Proof

In order for encryption and decryption to work, a few equations need to be true. First, we will prove an equation used in the first part of decrypting signed messages. We need specific equations to be true for the decryption to work, so we will start with the equation we want to prove, then make some substitutions to make an equivalent equation that we can prove, and finally prove it.

$$M_s = C_s^{d_B} \text{mod}(n_B) \quad (4)$$

We will go over why we want to prove equation (4) for RSA encryption in the next section. For now, it is important to know that the values are defined in terms of the values of our keys, M for the message, C for the ciphertext, subscript s for a signed term, and subscripts A and B for the distinction between sender and receiver respectively. What signed means in this context is that a mathematical signature has been applied to it, in this case using equation (5). We will use substitution to change the terms into something that

we can work with. In order to do that, we will use equations (5) and (6), the definitions of terms in equation (4), to get equation (7).

$$M_s = M^{d_A} \text{mod}(n_A) \quad (5)$$

$$C_s = M_s^{e_B} \text{mod}(n_B) \quad (6)$$

$$M^{d_A} \text{mod}(n_A) = ([M^{d_A} \text{mod}(n_A)]^{e_B} \text{mod}(n_B))^{d_B} \text{mod}(n_B) \quad (7)$$

As you can see, the entire left side of equation (7) is the same as just part of the right side of equation (7). This allows us to substitute this entire section with a placeholder integer, in this case, x .

$$x = ((x)^{e_B} \text{mod}(n_B))^{d_B} \text{mod}(n_B) \quad (8)$$

While it may be obvious that x is smaller than n_A because of the modulus, there is another important assumption that makes it smaller than any n regardless of the subscript. This is because of the initial stipulation that the size of the message must be smaller than our values of n , which in turn means that x , which in this case is a substitute for the signed message M_s , must be smaller than our values of n . This leads to the statement in equation (9) which we will use later.

$$x = x \text{mod}(n_B) \quad (9)$$

Since all of our terms are integers, we can take advantage of the distributive property of the modulus function on the right side of equation (8) and rearrange our terms.

$$x \text{mod}(n_B) = x^{e_B d_B} \text{mod}(n_B) \quad (10)$$

Now we can use substitution with our definition of n in equation (1) to change the modulus to pq . Also, since all the terms are recipient terms, we no longer need the subscripts and can do away with them for now.

$$x \text{mod}(pq) = x^{ed} \text{mod}(pq) \quad (11)$$

This allows us to separate one equation to prove into two equations we need to prove. If both sides are equal $\text{mod}(p)$ and $\text{mod}(q)$, then, by the properties of the modulus function, that means they are also the same $\text{mod}(pq)$. Equations (12) and (13) are the equations we will actually be proving in order to prove equation (4).

$$x \text{mod}(p) = x^{ed} \text{mod}(p) \quad (12)$$

$$x \text{mod}(q) = x^{ed} \text{mod}(q) \quad (13)$$

In order to prove equation (12), we will use Fermat's little theorem, which states that for any integer x and prime number p where p is not a factor of x , the following is true.

$$x^{p-1} \text{mod}(p) = 1 \quad (14)$$

The reason we are writing p and not q is arbitrary. The same proof will apply to both equations (12) and (13). This is because both p and q are the prime numbers multiplied together to get n in equation (1), with no distinction between which prime number is p and which is q . Using Fermat's little theorem means our proof needs to be separated into two cases depending on whether p is a factor of x . Since any integer modulus of one of its factors is 0, the left side of equation (12) equals 0, resulting in equation (15).

$$x \text{mod}(p) = 0 \quad (15)$$

Since e and d are integers, and taking an integer to integer exponents does not remove any factors from the integer, the right side of equation (12) is also 0.

$$x^{ed} \text{mod}(p) = 0 \quad (16)$$

Since both sides of the equation (12) are now equal to 0, we can use substitution with equations (15) and (16) to show that equation (12) is true, when p is a factor of x .

$$x \text{mod}(p) = x^{ed} \text{mod}(p) \quad (12)$$

Next, we will prove that equation (12) is also true in the case where p is not a factor of x . In order to be able to use Fermat's little theorem, we first need to get part of the equation in an appropriate form. We will start by taking one x out of the exponent of the right side of equation (12).

$$x^{ed} \bmod(p) = (x^{ed-1}x) \bmod(p) \quad (17)$$

Next, in order to have the integer p in the exponent, so we can use Fermat's little theorem, we will use the equation (3), which we previously used to generate the public exponent.

$$(e * d) \bmod((p - 1) * (q - 1)) = 1 \quad (3)$$

Before we can use equation (12) for substitution, we need one side of the equation to be $(ed-1)$. We can begin by subtracting 1 from both sides.

$$(ed - 1) \bmod((p - 1)(q - 1)) = 0 \quad (18)$$

Then, using the definition of the modulus function, we can say there exists an integer, h_0 , such that equation (19) is true. This enables us to isolate $(ed-1)$ on the left side of equation (19).

$$(ed - 1) = (p - 1)(q - 1)h_0 \quad (19)$$

Since the integers are closed under multiplication, and all our terms on the right side of equation (19) are integers, we can get rid of the $(q-1)$ term that we are not using by replacing $(q-1)h_0$ with a new integer, h .

$$(ed - 1) = (p - 1)h \quad (20)$$

Now we are finally ready to go back and substitute this into equation (17) in order to get $(p-1)$ in the exponent.

$$(x^{ed-1}x) \bmod(p) = (x^{(p-1)h}x) \bmod(p) \quad (21)$$

Next, for clarity's sake, we will put brackets around the part of equation (21) where we will use Fermat's little theorem to replace the enclosed terms with the value 1.

$$(x^{(p-1)h}x) \bmod(p) = [(x^{(p-1)})]^h x \bmod(p) \quad (22)$$

You may notice that the modulus appears to be in the wrong place to use Fermat's little theorem, but we are still working exclusively with integers so we can simply take advantage of the distributive property of the modulus function again and apply Fermat's little theorem to replace the bracketed part of equation (22) with 1 to get equation (23).

$$((x^{(p-1)})^h x) \bmod(p) = 1^h x \bmod(p) \quad (23)$$

The right side of equation (23) can now be substituted into the right side of equation (12). The only difference between the two sides of the equation (24) is that the right side is multiplied by 1 taken to an integer exponent, h . Since 1 to any exponent is still 1 and any term multiplied by 1 is left unchanged, the two sides of equation (24) are equal.

$$x \bmod(p) = 1^h x \bmod(p) \quad (24)$$

So, we have now also proven that equation (12) is true when p is not a factor of x .

$$x \bmod(p) = x^{ed} \bmod(p) \quad (12)$$

Since equation (12) is true both when p is a factor of x and when p is not a factor of x , it is true in general. As mentioned before, this proof can also be generalized to equation (13), so equation (13) is true as well.

$$x \bmod(q) = x^{ed} \bmod(q) \quad (13)$$

Therefore, we have proven that equation (11) is true.

$$x \bmod(pq) = x^{ed} \bmod(pq) \quad (11)$$

That equation was created to be equivalent to equation (7) that we wanted to prove, so that is also true.

$$M^{d_A \bmod(n_A)} = ((M^{d_A \bmod(n_A)})^{e_B \bmod(n_B)})^{d_B \bmod(n_B)} \quad (7)$$

This also shows that equation (6) is also true.

$$M_S = C_S^{d_B \bmod(n_B)} \quad (6)$$

4.2: Second Encryption and Decryption Proof

Next, we will prove another very similar equation we will also use later in the decryption process. Similarly, we will start by rewriting the equation that we want to prove. Since we have already proved that equation (11) is true this will be a shorter proof. All we have to prove that equation (25) is true is prove that it is equivalent to equation (11).

$$M = M_S^{e_A \bmod(n_A)} \quad (25)$$

You may recognize that the terms have all appeared before, but some of them are now sender terms instead of recipient terms. Still, we can use the same substitution for M_S as before to get something we can work with.

$$M = (M^{d_A \bmod(n_A)})^{e_A \bmod(n_A)} \quad (26)$$

Once again, we will use a substitution for an integer x . Despite the subscripts being A now instead of B , this is still the same x as the relation between all the subscript A values is the same as the relation between the subscript B values. The subscripts are only there to clarify whether we are discussing the sender's or the receiver's values.

$$x = ((x)^{e_A \bmod(n_A)})^{d_A \bmod(n_A)} \quad (27)$$

We can remove the subscripts from equation (27) as they are all terms from the sender.

Then, using the distributive property of the modulus, we can change the right side of equation (27) to get equation (28).

$$x = x^{ed} \bmod(n) \quad (28)$$

Since x is still less than n , the left side of equation (28) can be replaced with $x \bmod(n)$ to get equation (29).

$$x \bmod(n) = x^{ed} \bmod(n) \quad (29)$$

Substituting n for pq , using the definition from equation (1), in equation (29) gets us equation (11).

$$x \bmod(pq) = x^{ed} \bmod(pq) \quad (11)$$

Consequently, equation (25) is also proven to be true.

$$M = M_s^{e_A} \bmod(n_A) \quad (25)$$

Equations (4) and (25), which we have proven here, will be instrumental to our explanation of the encryption and decryption process in the next section.

5: Encrypting and Decrypting Messages

The equations and terms we defined during key generation were sufficient for extremely basic encryption and decryption. However, encrypting the messages usefully comes with an extra wrinkle. The receiver needs to have some secure way to know who the message is from. This is solved by signing the messages. First, a signed message, M_s , must be generated using equation (5).

$$M_s = M^{d_A} \bmod(n_A) \quad (5)$$

In equation (5), M is the message to be signed by the sender, d_A is the private key of the sender, and n_A is the public modulus of the sender. Next, the sender encrypts the signed message into the signed ciphertext, C_s , with equation (6).

$$C_s = M_s^{e_B} \bmod(n_B) \quad (6)$$

In equation (6), e_B is the public exponent of the receiver and n_B is the public modulus of the receiver. This ciphertext signed message is then sent to the receiver who can decrypt it into M_s with equation (4).

$$M_s = C_s^{d_B} \bmod(n_B) \quad (4)$$

In equation (4), d_B is the private key of the receiver. This step of the decryption process is why we had to prove equation (4) using the definitions of the encryption process in equations (5) and (6) in the previous section. Finally, the receiver calculates M from M_s using equation (25).

$$M = M_s^{e_A} \bmod(n_A) \quad (25)$$

In equation (25), e_A is the public exponent of the sender. This step of the decryption process is why we proved equation (25) in the previous section.

Many variations and improvements of this part of the process have occurred over time. The most basic form of RSA signing, which RSA encryption was created with, has a potential security vulnerability which causes it to be quite slow in practice, which we will cover later, along with one solution to it.

That this process works confirms for the receiver that the signed message was in fact sent by the sender, as only the sender could have encrypted the data this way with their private key and only the receiver can decrypt the message with their private key. If a message does not require a signature, the process can be simplified into equations (6) and (4) using the unsigned versions of the terms in order to get the most basic form of RSA encryption. However, this used to be rarely done in practice as the signature contains incredibly

useful information and skipping it opens up vulnerabilities in a wide range of contexts.

Now several methods exist as alternatives or shortcuts to original RSA signing in order to have faster computation for a variety of trade-offs.

6: Proof for Vulnerability Due to a Low e

In this section we will create a mathematical proof that will aid in explaining a vulnerability which occurs when equation (30) is true.

$$M_s^{e_B} < n_b \quad (30)$$

It follows that equation (31) is true due to the properties of the modulus function.

$$M_s^{e_B} \bmod(n_b) = M_s^{e_B} \quad (31)$$

This means that we can substitute the right side of equation (31) into equation (6) in order to get equation (32).

$$C_s = M_s^{e_B} \quad (32)$$

By taking the e_B root of both sides, we get the vulnerability in equation (33).

$$\sqrt[e_B]{C_s} = M_s \quad (33)$$

The nature of this vulnerability is that whenever an attacker can use equation (33), due to equation (30) being true, it allows them to use exclusively public information to get information which should be private.

7: Vulnerability Due to a Low e

We will look at a vulnerability caused by the value of e for equation (3) being too small.

If the left side of equation (30) is ever smaller than n_B , it results in a vulnerability. This is because it effectively causes the modulus function in equation (6) to do nothing. This

leads to the vulnerability of equation (33). The reason equation (33) is a vulnerability since it uses only information an attacker would know to get information they should not know. The left side of equation (33) includes the public exponent and the signed ciphertext, which the attacker has access to since they were both sent publicly. However, the right side of the equation is the signed message, which is something the attacker should not know. The attacker can then simply use equation (25) to fully decrypt the message. This vulnerability means a small message size with a small value of e , renders the security of RSA encryption ineffective. This creates a trade-off where smaller values of e are more computationally efficient, but those values are only secure if the minimum size of the message is large enough. This means that if small messages need to be sent, they need to be artificially lengthened in order to not compromise security despite the inefficiency of doing so. This is why the most efficient value of e that is still secure varies based on context. Depending on how many short messages are sent versus how many large messages are sent, a larger or smaller e could be more efficient overall. Most improvements to the RSA system focus on circumventing this simple but computationally intense solution, while minimizing the loss in security.

8: Conclusion

RSA encryption is a cryptographic system grounded firmly in the mathematics of prime numbers. This proven basis acts as a sturdy foundation which protects the transfer of digital information to this day. There are a variety of potential compromises and varieties of the RSA system in use because the core is so secure and sound. It securely communicates important information sent across the internet and all other computers every single day. Furthermore, it does so while giving a guarantee of who sent that information. All thanks to the mathematics it is built upon.

9: References

- Jonsson, J., & Kaliski, B. (2003). Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1 (Request for Comments RFC 3447). Internet Engineering Task Force. <https://doi.org/10.17487/RFC3447>
- Rivest, R. L., Shamir, A., & Adleman, L. M. (1983). Cryptographic communications system and method (US Patent 4405829). uspto.gov.