

# Encrypt it yourself!

*QARCHLI Mouad*

*qarchlimouad@gmail.com*

RSA is an asymmetric encryption algorithm which is used to send a message securely over a network, particularly an insecure network, such as the Internet.

In this article, I'll go first over a quick overview of RSA history, concept and some of its applications. Then straight to some mathematical theory; stuff you have seen in high school or college but may have forgotten, that is necessary to fully understand the rest of the article before digging deeper into the algorithm per se.

## **I. Introduction:**

### **1. Prerequisites:**



None. I'll walk you through the necessary mathematics in order for you to understand the RSA algorithm. However, I'm supposing that you're familiar with some arithmetic and algebra basics.

### **2. RSA, what is it?**

RSA stands for the initials of Rivest, Shamir and Adleman, who are the three Massachusetts Institute of Technology mathematicians behind it. They first described it publicly in 1977. Nowadays, it is used widely all over the internet, in e-payments and digital signatures as well as establishing secure connections with a remote server. It's an asymmetric cryptography algorithm, which means that we need two different but mathematically linked keys, one for encrypting and another for decrypting. The principle behind is simple: Suppose Alice wants to communicate with Bob over an insecure network. First, they both have to generate their public and private keys; which are no more than pair of numbers that follow certain rules. Second, Alice has to look for Bob's public key – which is the padlock by which Alice should lock the message before she sends it, generally the public key is published in a key server or a repository accessible by everyone, and only Bob who has the private key corresponding to the public key used to encrypt the message – which is the padlock key, will decrypt it. So even if some third party, say Eve, is sniffing the network, she will have an “alien-understandable” message, which is actually hard to decrypt because she doesn't have access to the private key and it's hard to find it when the algorithm is applied correctly. To remove any doubt about the identity of the sender, Alice may proceed otherwise. She can first encrypt the message with her private key then with Bob's public key before she sends it. This way, Bob is going to be sure that it's Alice that has sent him the message and not someone else since she's the only one who has access to her private key. Bob can then make use of his private key and Alice's public key to entirely decipher the message. This way of proceeding is what we call **the digital signature**; Alice has digitally signed the message before sending it. To better understand this concept, I'll show you an illustration:

Let me introduce you our friends Alice, Bob, and the villain Eve.



This is Alice. Alice has her public key  and her private key .

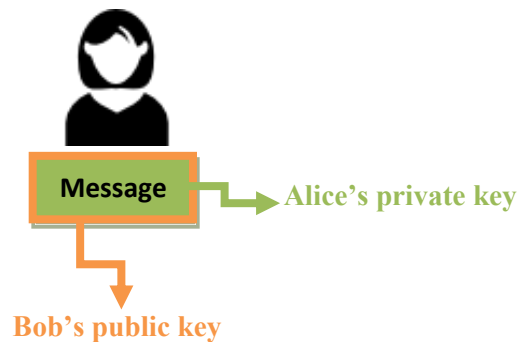


This is Bob. Bob has his public key  and his private key .

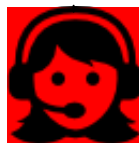


This is Eve. The villain eavesdropper.

- First Alice encrypts the message with her private key, which means that she **digitally signs** the message, then with Bob's public key.

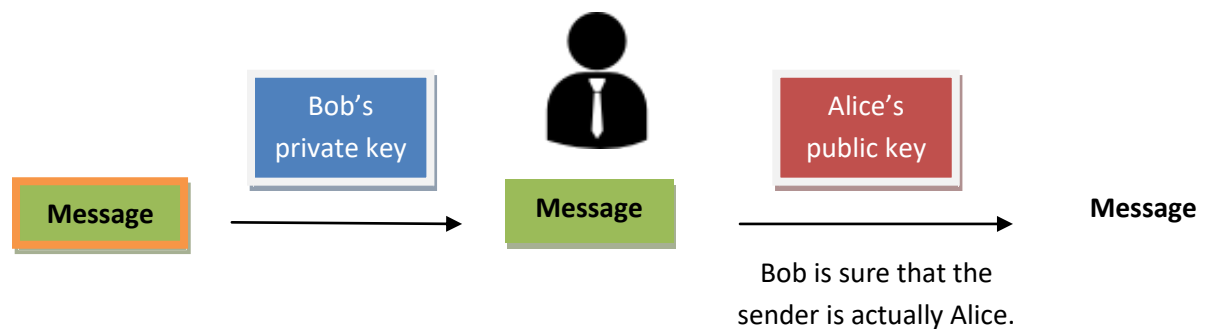


- The message is ready to be sent.



Cannot decrypt the message. Sorry for her.

- The message arrived to Bob, now he has to use his private key, and Alice's public key in order to decrypt the message entirely.



## II. Time for some theory:

### 1. Primes:

A prime is a number that is divisible only by itself and 1 (e.g. 2, 3, 5, 7, 11,...). This is a [list](#) of all prime numbers in the range 0 to 10.000.

How can we tell if a large number generated is a prime or not? Usually, when we are dealing with small integers, we run deterministic algorithms such as [AKS primality test](#), but when it's question of large numbers, we opt for probabilistic tests, which means, determining whether an input integer is a probable prime given a certain probability. The commonly used algorithm is [Rabin-Miller primality test](#).

### 2. Modular arithmetic:

$$\forall a, b, n \in \mathbb{Z}, a \equiv b \pmod{n} \Leftrightarrow \exists k \in \mathbb{Z}, a = b + k \cdot n$$

And we read,  $a$  and  $b$  are congruent modulo  $n$ .

What is the difference between  $=$  and  $\equiv$ ? Equal to ( $=$ ) means that  $a$  and  $b$  are the exact same thing. Congruent to ( $\equiv$ ) means that  $a$  and  $b$  have some property in common, in this case the same remainder when divided by the modulus  $n$ .

### 3. Coprime integers:

$$\forall a, b \in \mathbb{Z}, a \text{ is relatively prime to } b \text{ if and only if } \gcd(a, b) = 1.$$

With gcd: the [greatest common divisor](#) of  $a$  and  $b$ .

### 4. Euler's totient function, the $\varphi$ function:

*Given an integer  $N$ , the  $\varphi$  function (pronounced phi) counts the positive integers less than  $N$ , that are relatively prime to  $N$ .*

For example, say we want to calculate  $\varphi(N = 9)$ , this is the set of positive integers less than  $N = 9$ :  $\{1, 2, 3, 4, 5, 6, 7 \text{ and } 8\}$ . We can easily tell that the integers that are coprime with  $N$  are  $\{1, 2, 4, 5, 7, \text{ and } 8\}$ . So we can say that  $\varphi(9) = 6$ .

#### 4.1. Properties:

##### 4.1.1. $\varphi$ function of a prime number:

Let's take another example, this time with a prime number, say  $N = 11$ . The set of positive integers less than  $N=11$ :  $\{1, 2, 3, 4, 5, 6, 7, 8, 9, \text{ and } 10\}$ . We can easily tell that all these integers are coprime with  $N$ . So we can say that  $\varphi(11) = 10$ .

From this example, the following conclusion can be drawn, and it's a fundamental property of Euler's totient function:

$$\text{For any given integer } N, \text{ if } N \text{ is prime we have } \varphi(N) = N - 1.$$

##### 4.1.2. $\varphi$ is a multiplicative function:

$$\forall p, q \in \mathbb{Z}: \varphi(p \cdot q) = \varphi(p) \cdot \varphi(q)$$

*If moreover  $p$  and  $q$  are prime numbers, we'll have:*

$$\varphi(p \cdot q) = (p - 1) \cdot (q - 1).$$

5. Euler's theorem:

This is the fundamental theorem behind the RSA algorithm. It states that:

"Si fuerit  $N$  ad  $x$  numerus primus et  $n$  numerus partium ad  $N$  primarum, tum potestas  $x^n$  unitate minuta semper per numerum  $N$  erit divisibilis."

(And right now you're like, is this guy kidding us?). This is the Latin version, in mathematical notation, what it actually means is:

*For any  $x$  relatively prime to  $N$  we have:  $x^{\phi(N)} \equiv 1 \pmod{N}$*

This theorem can be demonstrated by [Fermat's little theorem](#).

6. Modular multiplicative inverse:

*The modular multiplicative inverse modulo  $n$  of an integer  $a$  is an integer  $x$  such that  $a.x \equiv 1 \pmod{n}$ .*

- **A necessary condition** for an integer  $a$  to have an inverse modulo  $n$  is  $\gcd(a, n) = 1$ .

6.1. Methods to find an inverse modulo  $n$  of an integer  $a$ :

6.1.1. A naïve method:

As we saw earlier in the modular arithmetic definition, we can write the expression  $a.x \equiv 1 \pmod{n}$  differently.

$$\begin{aligned} a.x \equiv 1 \pmod{n} &\Leftrightarrow \exists k \in \mathbb{Z} \text{ such that } a.x = 1 + k.n \\ &\Leftrightarrow a.x - k.n = 1 \\ &\Leftrightarrow a.x + k'.n = 1 \text{ with } k' \in \mathbb{Z} \\ &\Leftrightarrow a.x \bmod n = 1 \end{aligned}$$

A naïve approach is to try all numbers  $x$  from 1 to  $(n - 1)$ , and whenever  $a.x \bmod n = 1$  we break and return  $x$ .

Example:

- Let  $a = 3$  and  $n = 5$ , we have  $\gcd(3, 5) = 1$ .  
for  $x = 1$ :  $3 \times 1 \bmod 5 = 3$ ;  
for  $x = 2$ :  $3 \times 2 \bmod 5 = 1$ ;  $\Rightarrow$  we break and return  $x = 2$ ;

Now let's test with the necessary condition not met.

- Let  $a = 2$  and  $n = 6$ , we have  $\gcd(2, 6) = 2 \neq 1$ .  
for  $x = 1$ :  $2 \times 1 \bmod 6 = 2$ ;  
for  $x = 2$ :  $2 \times 2 \bmod 6 = 4$ ;  
for  $x = 3$ :  $2 \times 3 \bmod 6 = 0$ ;  
for  $x = 4$ :  $2 \times 4 \bmod 6 = 2$ ;  
for  $x = 5$ :  $2 \times 5 \bmod 6 = 4$ ;

We say that the number 2 doesn't have an inverse modulo 6. This conclusion is valid for any two integers that don't meet the necessary condition stated earlier in the definition.

6.1.2. Extended Euclidean Algorithm (EEA):

A more sophisticated method to find a modular inverse is to make use of EEA.

The EEA allows us, given two integers  $a$  and  $b$ , to calculate  $x$  and  $y$ , such that  $a.x + b.y = \gcd(a, b)$ . In order to meet our need, we have to customize this algorithm. We're going to put:  $b = n$ , and given the necessary condition on the greatest common

divisor, we have  $\gcd(a, n) = 1$ . We end up with this equation:

$$\begin{aligned} a.x + n.y &= 1 \Leftrightarrow a.x = 1 - n.y \\ &\Leftrightarrow a.x = 1 + k.n \text{ with } k \in \mathbb{Z} \\ &\Leftrightarrow a.x \equiv 1 \pmod{n} \end{aligned}$$

So the  $x$  that we're going to find using EEA, with an input of two integers  $a$  and  $n$  such that  $\gcd(a, n) = 1$ , is the multiplicative inverse modulo  $n$  of  $a$ .

So that's it. We've seen all the useful mathematics in order for us to understand the RSA algorithm. We're ready and well-equipped to dig into it.

### III. RSA algorithm:

#### 1. First step: Generating keys:

##### 1.1.1. Public key:

We select two prime integers  $p$  and  $q$ . **In order for RSA to be both effective and secure, these integers should be very large (1024bits), chosen at random, and should be similar in magnitude but different in length by a few digits to make factoring of the product  $p \cdot q$  harder. As for their primality, we make use of some algorithms that determine whether an integer is a prime or not. The commonly used algorithm is Rabin-Miller primality test discussed above in the prime number definition.**

In this article, we're going to do with small integers, to make the mathematics manageable.

- a. So in the first place, here are our two prime integers, say  $p = 83$  and  $q = 101$ ;
- b. We compute  $n = p \cdot q = 8383$ ;  $n$  is called the **modulus** and it will constitute the first component of the public key;
- c. The totient of the modulus  $n$  is  $\varphi(n) = (p - 1) \cdot (q - 1) = 8200$ ; (Remember Euler's totient function properties discussed above.);
- d. We select a random integer  $e$  such that:  $1 < e < \varphi(n)$  and  $\gcd(e, \varphi(n)) = 1$ .  
 $e$  is called **the encryption exponent** and it will constitute the second component of the public key. Let  $e = 947$ ;
- e. Our public key is the pair  $(e = 947, n = 8383)$ ;

**N.B:** The public key  $(e, n)$  is to be distributed publicly in order for anyone to communicate securely with us, but  $p, q$  and  $\varphi(n)$  are to be **DISCARDED**.

##### 1.1.2. Private key:

- a. The first component of the private key is the inverse modulo  $\varphi(n)$  of *the encryption exponent*  $e$ . In other words, we have to find an integer  $d$  such that  $e \cdot d \equiv 1 \pmod{\varphi(n)}$ . Since  $\gcd(e, \varphi(n)) = 1$ , the integer  $d$  exists. In our case, we're going to proceed using the naïve method, which gives us  $d = 7083$ ,  $d$  is called **the decryption exponent**.
- b. The second component of the private key is our modulus  $n = 8383$ ;
- c. Finally, our private key is the pair  $(d = 7083, n = 8383)$ ;

#### 2. Second step: Encryption:

Say Alice wants to send the message  $m$  = "Encrypt it yourself!" to Bob.

1. First, she has to look up Bob's public key. Say the public key we have computed earlier  $(e = 947, n = 8383)$ ;
2. Second, the message  $m$  should be converted to numbers. A way to do it is to convert each character to ASCII code. Here's an [ASCII table](#).

Character	ASCII code
E	69
n	110
c	99
r	114
y	121
p	112
t	116
space	32

Character	ASCII code
i	105
t	116
space	32

Character	ASCII code
y	121
o	111
u	117
r	114
s	115
e	101
l	108
f	102
!	33

3. The ciphertext  $c \equiv m^e \pmod{n}$ . So to find the ciphertext  $c$ , she has to compute  $c = m^e \pmod{n}$  for each character then concatenate the whole thing.

$m$ : Plaintext converted to ASCII code;

$e$  and  $n$ : Public key components;

Character	Encryption
E	6627
n	4972
c	8017
r	5458
y	4824
p	1947
t	5163
space	544

Character	Encryption
i	5282
t	5163
space	544

Character	Encryption
y	4824
o	7262
u	4031
r	5458
s	7682
e	6161
l	1770
f	2728
!	7155

Our ciphertext will be: 6627 4972 8017 5458 4824 1947 5163 544 5282 5163 544 4824 7262 4031 5458 7682 6161 1770 2728 7155. (Ignore the spaces; I've left them on purpose for good readability).

**N.B:** I haven't treated the digital signature phase for simplicity purposes. But the methodology stills the same. After converting the plaintext to numbers, Alice should first use his own private key to encrypt the original message then use Bob's public key to encrypt the resulting ciphertext.

### 3. Third step: Decryption:

In order for Bob to decrypt and read the message that Alice has sent him, he's going to make use of his private key, in our case ( $d = 2260, n = 8383$ ).

The message Bob has received is:

- 6627 4972 8017 5458 4824 1947 5163 544 5282 5163 544 4824 7262 4031 5458 7682 6161 1770 2728 7155.

The original message  $m \equiv c^d \pmod{n}$ . So in order for him to get the original message  $m$ , he has to take every ciphered block  $c$  and compute  $m' = c^d \pmod{n}$ , with  $(d, n)$  his private key, to find ultimately that  $m' = m$ . Then match the resulting ASCII code with the corresponding character.

Block	Decryption(ASCII code)
6627	69
4972	110
8017	99
5458	114
4824	121
1947	112
5163	116
544	32

Block	Decryption(ASCII code)
5282	105
5163	116
544	32

Block	Decryption(ASCII code)
4824	121
7262	111
4031	117
5458	114
7682	115
6161	101
1770	108
2728	102
7155	33

This is the result code after decrypting:

- 69 110 99 114 121 112 116 32 105 116 32 121 111 117 114 115 101 108 102 33.

If we match every resulting block with its corresponding ASCII code, we find our original message:

- Encrypt it yourself!

### Why does RSA work?

Quick recap:

- In the encryption phase we've seen that our ciphertext is  $c \equiv m^e \pmod{n}$ .
- In the decryption we've had  $m' \equiv c^d \pmod{n}$ .

In what follows, I'm going to demonstrate why RSA has worked.

We have:

$$\begin{aligned}
 m' &\equiv c^d \equiv m^{e \cdot d} \\
 &\equiv m^{1+k\varphi(n)} \quad (\text{Since } e \cdot d \equiv 1 \pmod{\varphi(n)}) \\
 &\equiv m \cdot m^{k\varphi(n)} \\
 &\equiv m \cdot (m^{\varphi(n)})^k \quad (*) \\
 &\equiv m \cdot 1^k \equiv m.
 \end{aligned}$$

The passage from the fourth to the fifth line (\*) is not immediately obvious and requires a condition on  $m$  that we haven't discussed in the encryption phase. In what follows, I'll try to show you the condition under which the passage (\*) is correct and therefore the original message is well recovered.

Imagine that the conversion of plaintext produces a number  $m > n$ . If we look at the decryption equation which is, I remind you,  $m' = c^d \bmod n$ , we can tell that it will never produce a number  $m'$  greater than  $n$  because of the modulo  $n$  operation, which is actually the Euclidean division remainder of  $c^d / n$ . Since  $m > n$  then  $m'$  won't be exactly equal to  $m$  but merely congruent to  $m \pmod{n}$ . To grasp this difference, take this example.  $7 \neq 3$  and  $7 \bmod 4 = 3$ . We say that 7 and 3 are not equal but are congruent  $\bmod 4$  ( $7 \equiv 3 \pmod{4}$ ).



The condition to put on plaintext conversion to numbers is  $m < n$ . If it isn't the case, the message  $m$  should be broken up into blocks smaller than  $n$ , encrypt the blocks then, after decrypting, concatenate the blocks to form the  $m$ .

Let  $P$  the proportion of numbers less than  $n$  that are relatively prime to  $n$ . Since there are  $n$  positive integers in the range 1 to  $n$ , we're going to have:

$$P = \frac{\varphi(n)}{n} = \frac{(p-1) \cdot (q-1)}{pq} = \frac{pq - p - q + 1}{pq} = 1 - \frac{1}{q} - \frac{1}{p} + \frac{1}{p \cdot q}$$

Now let's calculate  $\bar{P}$ , the proportion of numbers less than  $n$ , that are **NOT** relatively prime to  $n$ . We have

$$P + \bar{P} = 1 \Rightarrow \bar{P} = 1 - P \Rightarrow \bar{P} = \frac{1}{q} + \frac{1}{p} - \frac{1}{p \cdot q}$$

When  $p$  and  $q$  are large enough – which is the case in practice,  $\bar{P} \rightarrow 0$ . Choosing  $m$  less than  $n$  guarantees that  $m$  is most likely prime to  $n$ . Thus, Euler's theorem is applicable in the passage (\*) above, and the message is therefore well recovered.

**N.B.:** I would particularly like to stress that co-primality of  $m$  and  $n$  is **NOT** a condition of well-decryption. Indeed, we can demonstrate that RSA will always work for every message  $m$  in the range 0 to  $n - 1$ , whether it's co-prime with  $n$  or not. For this, we're going to introduce a new theorem called the Chinese Remainder Theorem (CRT). It states that:

$$\text{if } p \text{ and } q \text{ are co-prime and } \begin{cases} x \equiv y \pmod{p} \\ x \equiv y \pmod{q} \end{cases} \Leftrightarrow x \equiv y \pmod{pq}$$

In RSA's case, we have  $n = pq$  with  $p$  and  $q$  are co-prime since they're both prime numbers, so we can say that:

$$\begin{aligned} \begin{cases} c \equiv m^e \pmod{n} \\ m' \equiv c^d \pmod{n} \end{cases} &\Leftrightarrow m' \equiv m^{ed} \pmod{p} \\ &\Leftrightarrow m' \equiv m^{1+k(p-1)(q-1)} \pmod{p} \\ &\Leftrightarrow m' \equiv m \cdot (m^{(p-1)})^{k(q-1)} \pmod{p} \\ &\Leftrightarrow m' \equiv m \cdot (m^{\varphi(p)})^{k(q-1)} \pmod{p} \end{aligned}$$

If  $m$  and  $p$  were co-prime then we'll be able to apply Euler's theorem for  $m$  and  $p$  and have  $m' \equiv m \pmod{p}$ , otherwise, i.e.  $m = kp$  with  $k \in \mathbb{Z}$ , then  $m \equiv 0 \pmod{p}$  which trivially proves that  $m' \equiv 0 \pmod{p}$ , implying that  $m' \equiv m \pmod{p}$ . So we were able to prove that, given any message in the range 0 to  $n - 1$ , we're capable of encrypting it with the public key and decrypting it using the corresponding private key. However, as mentioned above, the chances that a given message  $m$  less than  $n$  is **NOT** relatively prime to  $n$  are very very low.

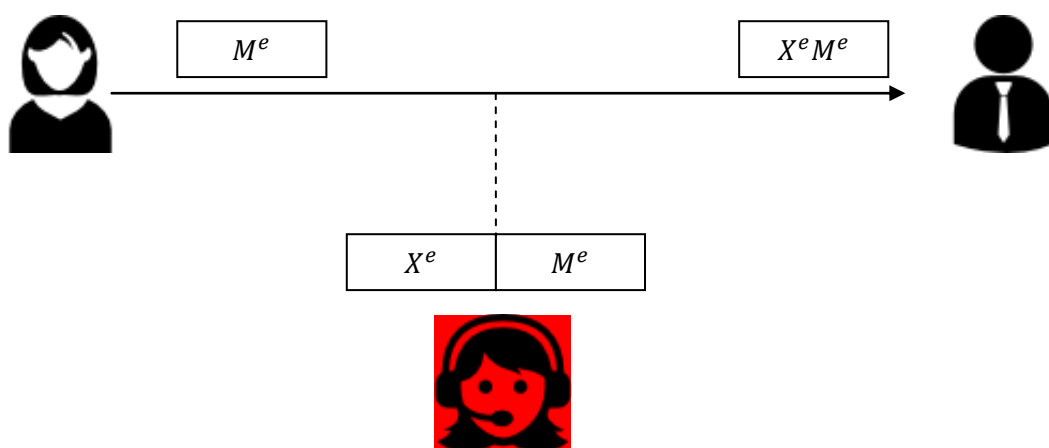
**Takeaway:** In the encryption phase, when converting the plaintext to numbers, make sure that every chunk converted is smaller than  $n$ .

#### IV. Textbook RSA vulnerabilities:

What we've did so far is called textbook RSA or schoolbook RSA. What it actually means is that we've just described the RSA algorithm from a mathematical point of view neglecting any real world implementations and its security flaws. The reality is textbook RSA has several weaknesses that I will point out in this paragraph. In what follows, I'll refer to textbook RSA simply by RSA.

The first weakness of RSA is determinism. Given a plaintext and a key, RSA will always produce the same ciphertext even over separated executions of the algorithm; there's no randomness introduced in the encryption process. An eavesdropper can gain information about the meaning of various ciphertexts by encrypting different plaintexts using the public key and constructing a dictionary of pairs plaintext/ciphertext, then collecting encrypted messages over the same channel and try to match ciphertexts using his dictionary.

The second weakness of RSA is malleability. A malicious third party can manipulate and transform the ciphertext transmitted leading to a modification in the plaintext that will be decrypted, with neither the sender nor the receiver realizing that the original message has been modified. This kind of attacks is very undesirable since it allows the attacker to modify the contents of a message. I'll try to further explain this weakness, from a mathematical point of view and using an illustration, in order for you to better understand the problem. Suppose Alice want to send Bob (public key:  $(e, n)$ , private key:  $(d, n)$ ) a message  $M$ .



Eve catches the original ciphertext, attaches to it a new ciphertext  $X^e$  and resends the whole thing to Bob.

As we saw earlier, Alice has to compute  $M^e$  in order to cipher the message before sending it to Bob who has to compute  $(M^e)^d$  once he receives it to obtain the original plaintext. While Alice has sent  $M$  as the original plaintext, Bob will receive  $XM$  as Eve has actually caught the original ciphertext; attached to it some  $X^e$ ; which is the encryption of an undesirable message  $X$  with Bob's public key, then resent the whole thing to Bob. So while deciphering, Bob will compute  $(X^e.M^e)^d = X.M \neq M$ . The resulting plaintext deciphered by Bob is different from the plaintext sent by Alice and neither of them is aware of this modification. In the case of a bank transaction for example, imagine that the message  $M$  is actually an amount of money, as for Eve's modification  $X$ ; imagine it's a certain factor that will modify the amount of money without Bob realizing that it's not the plaintext that Alice has tried to send in the first place. That's actually the problem.

To know more about other core RSA attacks, I strongly recommend an article by Dan Boneh entitled [\*Twenty years of attacks on the RSA cryptosystem\*](#). This article is an overview of the common attacks that has been run on RSA system and the solutions that you have to consider when implementing this algorithm.

To conclude, anyone who wants to implement RSA cryptosystem don't have to do it on their own especially when it's question of secure data. There are plenty of solid and tested implementations ready to be used. Cryptographic algorithms are *very* easy to get wrong, and the slightest mistake can completely undermine the security of the system.

### **References:**

- <http://certauth.epfl.ch/rsa/rsa.html>
- <https://www.khanacademy.org/computing/computer-science/cryptography>
- [https://en.wikipedia.org/wiki/RSA\\_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))
- <https://www.geeksforgeeks.org/rsa-algorithm-cryptography/>
- <http://cacr.uwaterloo.ca/hac/about/chap4.pdf>
- <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>
- [https://en.wikipedia.org/wiki/Deterministic\\_encryption](https://en.wikipedia.org/wiki/Deterministic_encryption)
- [https://en.wikipedia.org/wiki/Malleability\\_\(cryptography\)](https://en.wikipedia.org/wiki/Malleability_(cryptography))