Cryptography - A RSA Study

Simone Mione

Cryptography - Computer Science VFU

Introduction

One of the main problems with classical cryptography methods is key symmetry. This is a problem since the users have to exchange a common secret key before they can communicate privately, but how do they can secretly exchange this key? This is a problem that a public-key cryptography method solves, let's see an example:

Two users, Alice and Bob, want to exchange private messages. In this scenario, we have:

- *M*: plain message
- *C*: cipher message
- *A*: Alice
- A_e: Alice's public key
- A_d: Alice's secret key
- *B*: Bob
- B_e: Bob's public key
- B_d: Bob's secret key
- $E(M)$: encryption of M (to get C)
- $D(C)$: decryption of C (to get M)

A wants to send a message M to B. She encrypts M with bob's public key B_e, so M becomes C and Alice sends it to Bob. Now, if someone detects C, he won't be able to read it because it's encrypted. Once Bob receives C, he can decrypt it with his secret key B_d to get M back.

RSA is a method that makes a public-key scheme usable in practice. Furthermore, RSA's encrypt/decrypt function is a trap-door one-way permutation; it means that E and D are easy to compute if you have the private key and it is not so easy to compute if you don't have it (one-way function), and you can also first decrypt and then encrypt (permutation), and it works well, more formally: $M = D(E(M))$ and also $M = E(D(M))$. This result is good for "signature" (we will see it in section 1.4).

In section 1 we'll see the RSA protocol and its math. In section 2 we'll see a practical example of RSA using a software. In section 3 we'll have a small talk about RSA security and a possible attack approach.

1 RSA - Theoretical formulation

Before talking about RSA, we show some important maths facts.

- $\phi(n)$ is the Euler function. It computes the number of integers in [1,n] that are relatively prime with n. In the case of n is prime, it's simple to understand that $\phi(n) = n - 1$. However, the general formula is:

$$\phi(n) = n * \prod_{d|n}(1 - \frac{1}{p}) \tag{1}$$

  In particular, if n is the product of two prime numbers p and q, the formula becomes:

$$\phi(n) = (p - 1) * (q - 1) \tag{2}$$

  and it is quite simple to compute.

- if $gcd(p, q) = 1$ then p and q are relatively prime.
- if p is prime, then $\forall x \in \{1, ..., p - 1\},\ x^{p-1}\ mod\ p = 1$ (Fermat's Little Theorem)

1.1 RSA protocol

Recalling the example above, RSA protocol is the following:

- Choose randomly two large prime numbers p and q
- Compute $n = p * q$
- Compute $\phi(n) = (p - 1)(q - 1)$
- Choose $e \in \mathbb{Z}_n\ t.c.\ gcd(e, \phi(n)) = 1$, so e and $\phi(n)$ are relatively primes. Because e is public, it can be small and fixed. A default value for e is 65537.
  - where $\mathbb{Z}_n = \{0, ..., n - 1\}$
- Compute $d = e^{-1}\ mod\ \phi(n)$.
  - $e^{-1}\ mod\ \phi(n)$. The modular multiplicative inverse always exists if e and $\phi(n)$ are coprime, and they do.
- Publish e and n, and keep d secret. The pair (e, n) is the public key, while the pair (d, n) is the secret key.
- Encryption: Given M, compute $C = M^e\ mod\ n$
- Decryption: Given C, compute $M = C^d\ mod\ n$

Note that the parameter n must be large so it is (almost) impossible to go back from C to M without d. We need a large n because break the encryption is easy if you know the factorization of n, and the factorization of n is easy to compute if n is small. See it in detail in section 3.

1.2 How many different messages

Because of the $mod\ n$ operation, there are at most n different messages, those in $\mathbb{Z}_n$. Actually, they are even less for two reasons:

1. messages $M = 0$ and $M = 1$ are the same in ciphertext and plaintext
2. other "bad messages" are the multiples of p and of q

So, because the 1. and 2., the total count of bad messages is: $|\{0, 1\}| + p - 1 + q - 1 = p + q$ and the number of feasible messages is:

$$\phi(n) - 1 \tag{3}$$

Note that for n large, the number of bad messages becomes negligible.

1.3 Proof RSA protocol

Let S_n be the space of feasible messages. Proving that RSA works is proving that, $\forall M \in S_n$: $if\ C = M^e\ mod\ n\ (e.g.,\ C = E(M)),\ then\ M = C^d\ mod\ n\ (e.g.,\ M = D(C))$

We have:

$$C^d\ mod\ n = (M^e\ mod\ n)^d\ mod\ n = M^{ed}\ mod\ n = M^{k\phi(n)+1}\ mod\ n = M^{k(p-1)(q-1)+1}\ mod\ n \tag{4}$$

since d is the inverse of $e\ mod\ \phi(n)$, so $ed = k\phi(n) + 1$ for some integer $k \in \mathbb{Z}_n$
So, we have to prove that:

$$M^{k(p-1)(q-1)+1}\ mod\ n = M\ (e.g.,\ C^d\ mod\ n = M) \tag{5}$$

or, equivalently:

$$M^{k(p-1)(q-1)}\ mod\ n = 1 \tag{6}$$

since $M \in S_n$ is coprime with n and thus has multiplicative inverse.
To prove:

$$M^{k(p-1)(q-1)}\ mod\ n = 1 \tag{7}$$

we show that:

$$M^{k(p-1)(q-1)}\ mod\ p = 1 \tag{8}$$

but this is easy because $M^{p-1}\ mod\ p = 1$ (by Fermat's Little Theorem), so:

$$M^{k(p-1)(q-1)}\ mod\ p = (M^{p-1}\ mod\ p)^{k(q-1)}\ mod\ p = 1^{k(q-1)}\ mod\ p = 1 \tag{9}$$

At the same way we can proof that $M^{k(p-q)(q-1)}\ mod\ q = 1$

1.4 Signature

A great feature of RSA is the trap-door permutation. Thanks to this feature, e and d can be exchanged. In this way, Bob can encrypt the message M with his private key B_d and getting C; then Alice can decrypt C with Bob's public key B_e. Of course, C can be decrypted by everyone, it's not secret since it can be decrypt with a public key, but the goal here is the signature! If the message is encrypted with B_d, it is for sure by Bob because only Bob can have his secret key. We call the ciphertext encrypted by the secret key S (signature). But that's not enough, because we need something to assure that M is really encrypted by B_d, so we have to check that $S = M^{B_d}\ mod\ n$, thus $S^{B_e}\ mod\ n = M$.

The protocol to sign a message is the following:

- Given M, Bob computes $H = hash(M)$
- He signs H with B_d and compute $S = H^{B_d}\ mod\ n$
- He sends the pair (M, S) to Alice
- Alice computes $H' = S^{B_e}\ mod\ n$ and also $H'' = hash(M)$
- Now, she will be sure that S has been signed by B_d if $H' = H''$

It's easy to understand why in this scenario the message is not secret. If Bob wants to send to Alice a secret signed message, the protocol becomes:

- Given M, Bob computes $C = M^{A_e}\ mod\ n$ (encrypt M with Alice's public-key)
- He also computes $S = C^{B_d}\ mod\ n$
- The pair (C,S) is sent to Alice
- Alice decrypts C with her private-key obating M' and she also decrypts S with Bob's public key
- Now, if $M' = M''$ the message is from Bob

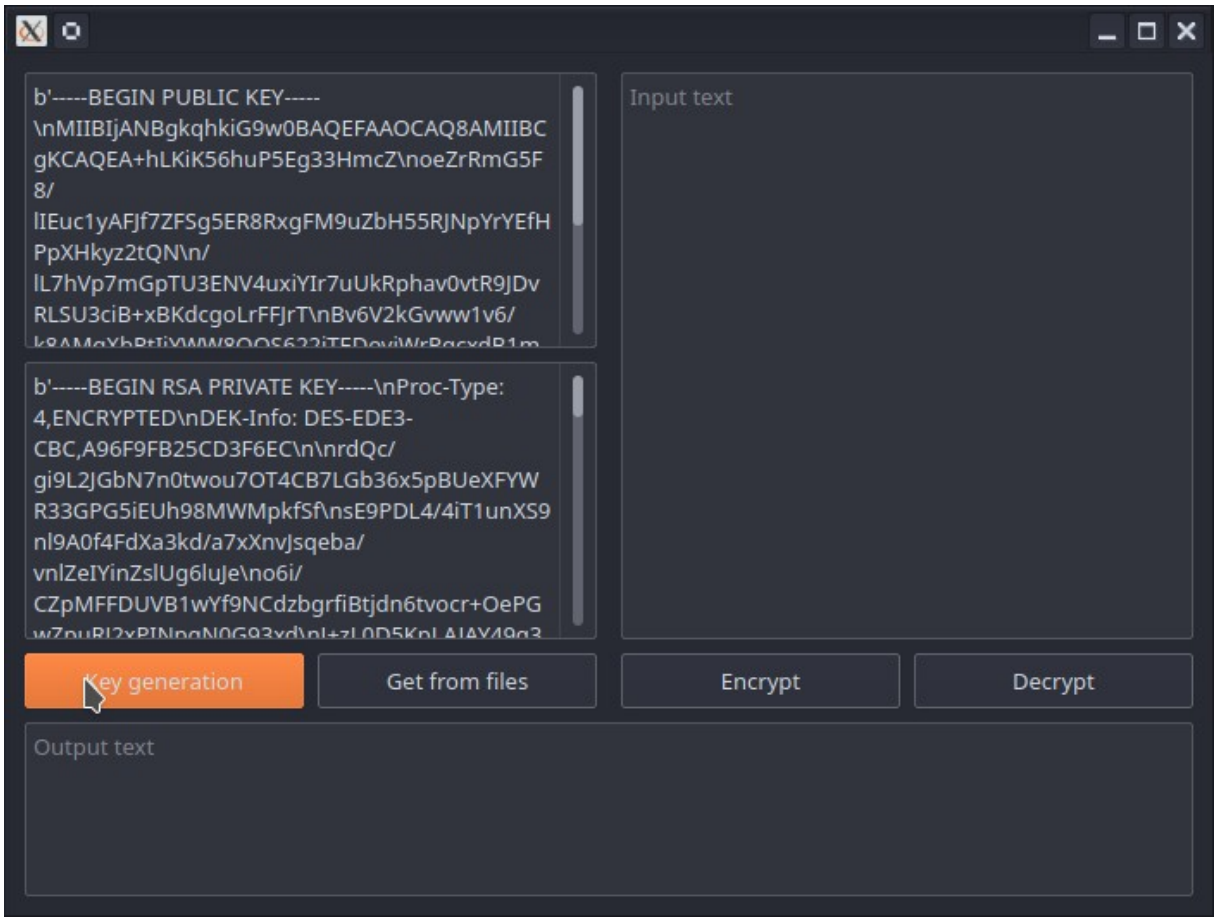

Figure 1: Keys generation

2 Using a RSA encryption/decryption software

In this section, we'll see how to use a software to encrypt and decrypt a message. The software used has been written by the author of this paper with the purpose to keep it simple in order to better explain the main steps. This software has been written using Python and using the package pycrypto to performs RSA stuff. Furthermore, package qt5 has been used to build a simple gui.

Let's try to encrypt our first message:

1. Generate public and private keys and save them on files "public_key" and "private_key" (Fig. 2)
2. Using the public key just generated, it performs the encryption of the message "Cryptology in Varna Free University" obtaining the cipher message as output (Fig. 2)
3. Using the private key just generated, it performs the decryption of the cipher message getting back the original message as output (2)

Now we'll see in detail how it works:

1. The key generation starts with the function $RSA.generate(bit\_lenght = 2048,\ e = 65537)$. This function computes p and q choosing two strong prime numbers. Then, it computes n as $p * q$. If the size of n is okay, it is accepted. Then, d is found by computing $e^{-1}\ mod\ \phi(n)$. To compute it, remember that $e * d = 1 + k\phi(n)$ for some $k \in \mathbb{Z}$. So the function iterates through $\mathbb{Z}$ to find the solution.
2. Using the keys generated, the encryption phase starts by $rsa.encrypt(message)$ (N.B. the key is embedded in rsa object). The encryption is performed using $RSAES - OAEP - ENCRYPT$, and is specified in section 7.1.1 of RFC3447 (https://datatracker.ietf.org/doc/html/rfc3447#section-7.1.1)

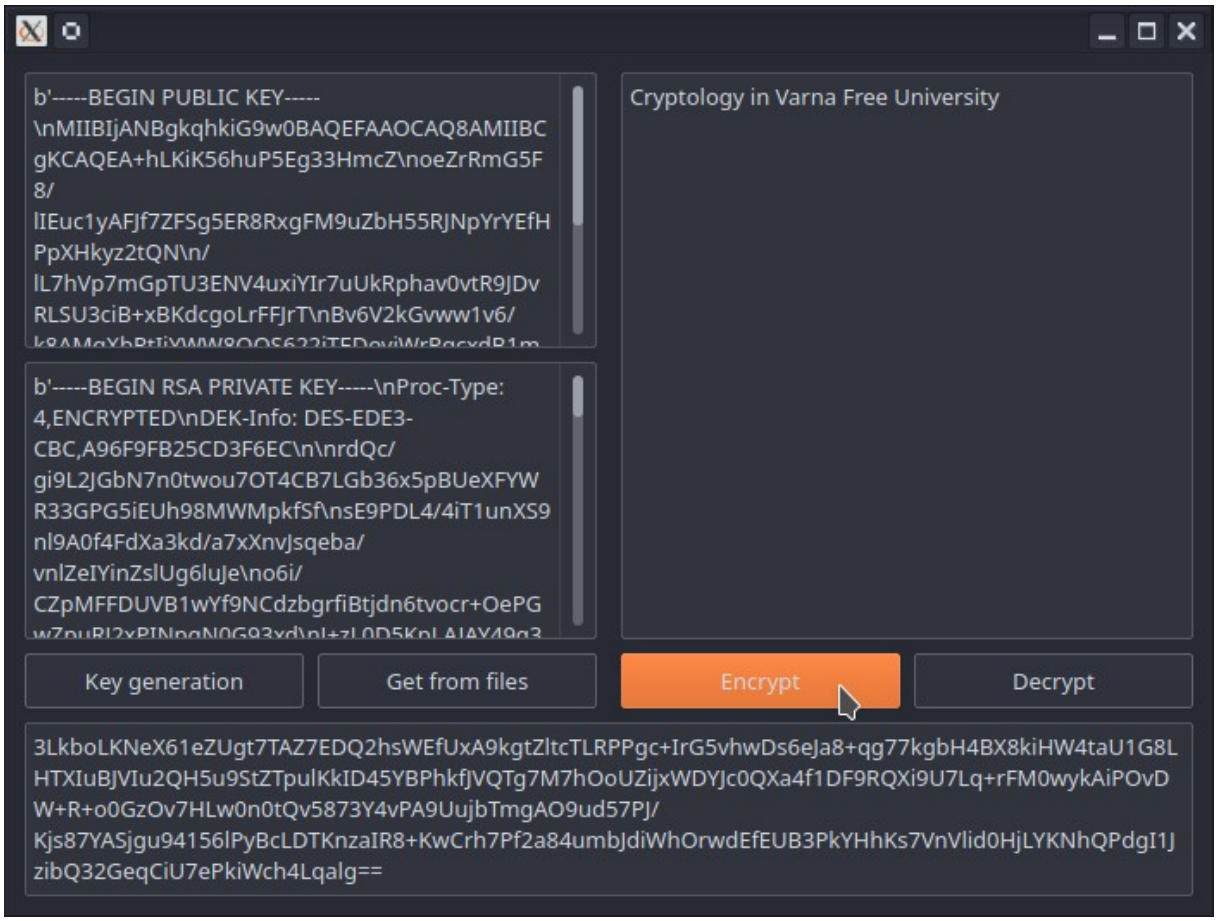

Figure 2: Encryption

3. Similarly, using the keys generated, the decryption phase starts by $rsa.decrypt(cipher\_message)$. The decryption is performed using $RSAES - OAEP - DECRYPT$, and is specified in section 7.1.2 of RFC3447 (https://datatracker.ietf.org/doc/html/rfc3447#section-7.1.2)

Attached to this paper, you will find the code of this software and the keys used for the encryption of our first message.

2.1 RSA - Security

One of the most important security factor of RSA is the key size. If the key size is not enough, it can be simple to break RSA with a brute-force attack. A good key size is 2048bits. Currently, with such a key size is not possible break RSA with a brute-force attack in a reasonable amount of time.

Another important point is the true random prime numbers. If the numbers are not sufficiently random, it can be much easier for the attacker to break the encryption because it is easier to factor them.

2.2 Factoring n

As said before, if the size of n is too small, it can be easy to break the encryption. That is true because, with a small key size, one can find the factoring of n, so he can find p and q, and from them obtain d. Remember that factoring n is as hard as big n is; but on the other hand, some numbers are harder to factor than others, that's why we want large random prime numbers.

The faster-known algorithm for factoring n is approximately $O(ln(n)^{\sqrt{(\frac{ln(n)}{ln(ln(n))})}})$, it means that to factor a n of 160 bits (=50 decimal digits) it takes $4\ hours$, while to factor a n of 1000 bits (=300 decimal digits) it takes about $4.9 * 10^{15} years$. So, it is reccomend to use a 2048 bits lenght key.

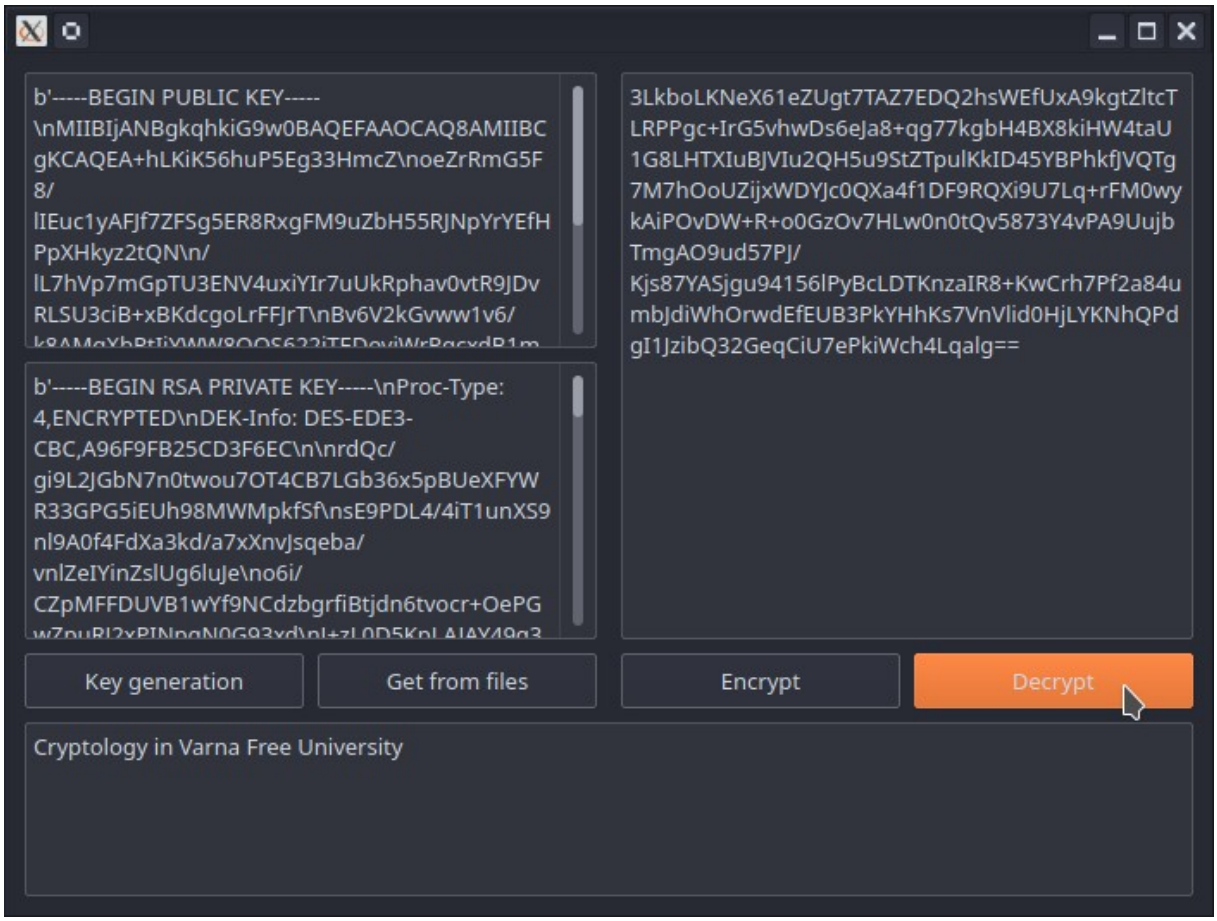

Figure 3: Decryption

A basic factoring algorithm implementation:

factoring.py

```
def factoring(n):
  factors = list()
  for i in range(3, n/2, step=2):
    if n%i == 0:
        print(i)
```

Suppose we choose $p = 863$ and $q = 977$, so $n = p * q = 863 * 977 = 843151$. Factoring n we will get:

Command Line

```
$ factoring(n)

        863
        977
```

the result is exactly our p and q, and thanks to them the attacker can compute the secret key.

No one has proven that an efficient algorithm to factor a large number is impossible, so this method could be feasible to break RSA if someone finds a way.

3 Conclusion

RSA is a very efficient encryption/decryption algorithm since it is very safe and secure; it is hard to crack because it involves factorization of prime numbers which is very difficult to do. Furthermore, RSA uses the public-key approach so it is easy to start a private communication through an insecure channel. Thanks to the trap-door one-way permutations, RSA can also performs digital signatures. RSA is also simple to implement and to run.

In some case, if data are very large, RSA can be very slow. Moreover, it requires a third party to verify the reliability of public keys. The public-key approach could be a double-edged sword due to a middlemen could temper the public key system.

So, there are pros and cons to use a public-key approach as there are pros and cons to use a symmetry-key approach, in conclusion, both techniques are important in encryption field, and they can be used together to be stronger.