RSA Cryptosystem

Èrik Campobadal Forés

Table of contents

---

1. Introduction

The goal is to implement a RSA cryptosystem using Python 3.

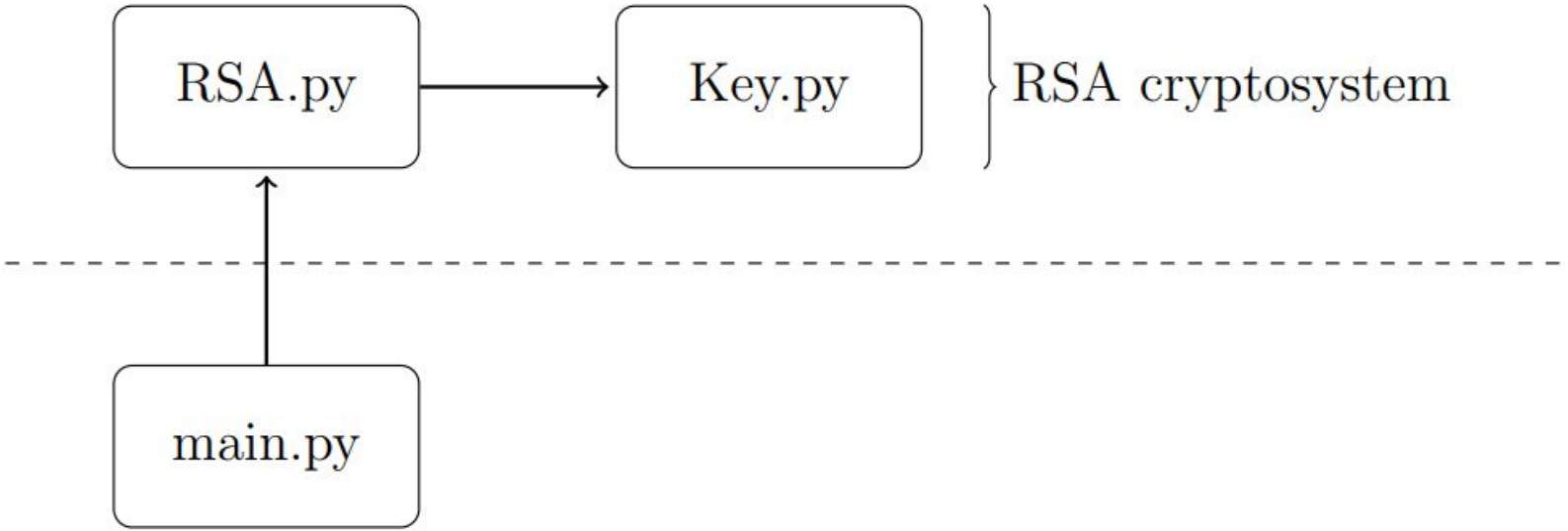
The library will contain two modules named:

1. RSA.py
2. Key.py

It must allow the encryption and decrypting of integers. Furthermore, string encryption and decryption could be made.

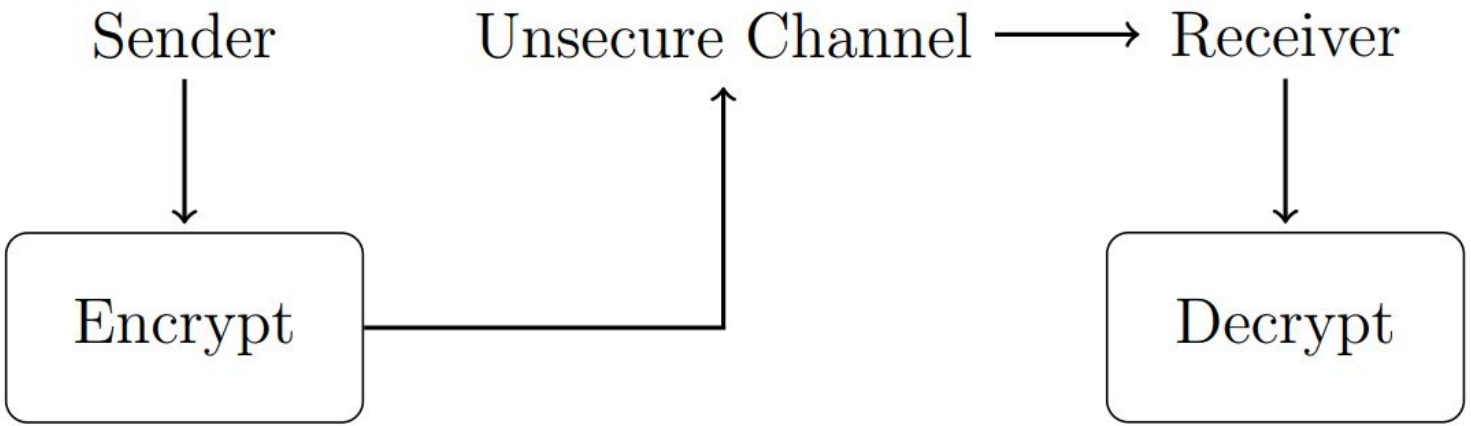
1. Introduction

The system architecture is the following:



1. Introduction

RSA is a public-key based cryptosystem meaning it requires a public and private key pair to work. The following scheme shows how the message travels by using RSA.



1. Introduction

The explanation regarding to the previous image is the following:

1. The sender encrypts a message using the receiver’s public key.
2. The sender send the message to the unsecure channel.
3. The receiver gets the message and decrypts it using his private key.

2. RSA Cryptosystem

To generate a key you need to execute the following steps:

- Choose two prime numbers p and q. Those numbers should be randomly chosen and should have a similar length in bits.
- Calculate $n = p \cdot q$. The result n will be used as the modulus for both keys (private, public) and it's length in bits designates the **key length**.
- Calculate $\Phi = (p-1) \cdot (q-1)$. Euler's properties are used to calculate Φ.
- Calculate e given $e < \Phi$ and e to be coprime with Φ.
- Determine d given $e \cdot d \equiv 1 \mod \Phi$ (Modular multiplicative inverse)

The public key will be: `(n, e)` and the private key will be `(n, d)`

2. RSA Cryptosystem

To encrypt a message you need to use the public key of the receiver.

$$c \equiv m^e \mod n$$

Consider c to be the cipher text generated by the encryption.

2. RSA Cryptosystem

To decrypt text the receiver just need to apply his private key to the cipher text.

$$p \equiv c^d \mod n$$

$$p \equiv (m^e)^d \mod n$$

$$p \equiv m^{e \cdot d} \mod n \quad \rightarrow \quad p \equiv m^1 \mod n$$

$$\boxed{p \equiv m \mod n}$$

3. Algorithms needed

To build the cryptosystem, a few algorithms are needed. Those include:

- Extended Euclidean Algorithm

  Will be used to compute d given a = e and b = Φ considering $d \cdot e \equiv \mod \Phi$.

- Fast modular exponentiation

  to calculate $x \equiv a^z \mod n$.

4. RSA Key (code)

The Key class will be used inside the RSA class to determine the current used key. Keep in mind that the Key is able to automatically execute the steps mentioned in the key generation process given two integers p and q. This is done in the constructor and further saved into the class attributes.

4. RSA Key (code)

The RSA Key module will contain the following data

p, q, n, phi, e, d	The basic storage attributes for the RSA key as defined in the introduction.
__init__(self, p, q)	Basic constructor to generate a key given p and q.
__generate_e_and_d(self)	Method to generate an e and d given Φ and e.
__generate_e(self, phi)	Generate an e based on Φ.
__extended_euclidean_algorithm(self, a, b)	The Extended Euclidean Algorithm given $a = e$ and $b = \Phi$ to compute d considering $d \cdot e \equiv \mod \Phi$.
length(self)	Returnns the length of the key.
public(self)	Returns the public key.
private(self)	Returns the private key.
print(self)	Prints the whole key (Public + Private).

5. RSA Cryptosystem (code)

The RSA class will be responsible of generating keys, encrypting and decrypting messages. The key generation will be dispatched to the Key class, responsible of this things. The RSA will only store and use the key

5. RSA Cryptosystem (code)

The RSA Cryptosystem module will contain The following data

key	Stores the RSA key class used to encrypt / decrypt messages.
__fast_modular_exponentiation(self, a, z, n)	Method to calculate $x \equiv a^z \mod n$.
generate_key(self, p, q)	Method to generate a key given p and q
encrypt(self, plain_text)	Encrypts a message (Might be a number or a string)
decrypt(self, cipher_text)	Decrypts an encrypted number or sequence of numbers (string).

6. Library Usage (code)

To use the library you simply need to import it as a regular Python 3 library and use the RSA class to access the given public methods. That said, the following example may ilustrate a sample usage case.

6. Library Usage (code)

```python
from RSA.RSA import RSA

system = RSA()

key = system.generate_key(61, 53)
key.print()

message = 'Cryptography'
cipher_text = system.encrypt(message)
plain_text = system.decrypt(cipher_text)
print(f'Original: {message}')
print(f'Cipher text: {cipher_text}')
print(f'Plain text: {plain_text}')
print(f'Correct: {message == plain_text}')
```

7. Live Sample

8. Conclusion

Given a correct procedure and the correct algorithms, a simple playground to try the RSA cryptosystem can be built with very small effort. However, the theory behind it and this implementation serves as a proof of concept for a well escalated RSA cryptosystem that could be used in production.

9. References

Khan Academy. (2019). *Fast modular exponentiation*. [online] Available at: https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/fast-modular-exponentiation [Accessed 12 Jan. 2019].

Loyola University Chicago. (2019). *Extended Euclidean Algorithm*. 1st ed. [ebook] Boston, Massachusetts: Dr. Andrew Harrington, p.2. Available at: https://anh.cs.luc.edu/331/notes/xgcd.pdf [Accessed 12 Jan. 2019].

Wikipedia. (2019). *RSA (cryptosystem)*. [online] Available at: https://en.wikipedia.org/wiki/RSA_(cryptosystem) [Accessed 12 Jan. 2019].

Python. (2019). *Python 3.7.2 Documentation*. [online] Available at: https://docs.python.org/3/ [Accessed 12 Jan. 2019]

Thank you for listening

https://github.com/ConsoleTVs/RSA

Èrik Campobadal Forés
https://erik.cat
soc@erik.cat

Polytechnic University of Catalonia