RSA Implementation

Ashutosh Samal

October 2018

1 Introduction

This project attempts to implement RSA asymmetric cryptosystem in python. The functions required for implementation are sequentially added, and the final implementation in done in RSA.py file. The following functions are implemented:-

- Modular Addition
- Modular Multiplication
- Modular Exponentiation
- Euler's Phi function
- Modular Inverse
- Modular Division
- RSA Key Generation
- RSA Encryption
- RSA Decryption

2 Overview

The RSA cryptosystem (named after its inventors Rivest, Shamir and Adleman) is an example of asymmetric cryptosystem used in widely in applications such digital signature and certificates.

The security of RSA-based systems relies on the fact that the prime-factorization of large numbers is computationally difficult (No polynomial time algorithm has been found yet, the best algorithm **GNFS - General Number Field Sieve** runs in sub-exponential time).

3 Algorithm

The RSA cryptosystem has the following components:-

- A semi-prime modulus (**N**), typically a product of two large prime numbers (say, **p** and **q**).
- A public key (**e**), known to everyone including the adversary.
- A private key (**d**), secret to everyone else other than the individual.

The encryption of plaintext message (**m**) is done using modular exponentiation with the public key (**e**) to obtain the ciphertext (**c**).

$$c = m^{\mathrm{e}} \pmod{N} \tag{1}$$

Subsequently, the decryption of ciphertext (**c**) to plaintext message (**m**) is done using modular exponentiation with private key (**d**).

$$m = c^{\mathrm{d}} \pmod{N} \tag{2}$$

Any cryptosystem must satisfy the correctness property:-

$$D_{\mathrm{d}}(E_{\mathrm{e}}(m)) = m \tag{3}$$

In RSA system,

$$m^{\mathrm{ed}} \equiv m \pmod{N} \tag{4}$$

or

$$m^{\mathrm{ed\text{-}1}} \equiv 1 \pmod{N} \tag{5}$$

4 Implementation

- Basic implementation of modular arithmetic is straightforward. Modular exponentiation is done using the inbuilt python function pow(a, b, n), which runs in the time-complexity of $\mathbf{O(\log_2(n))}$.
- **Modular Inverse**: The modular inverse of number **a** in field $\mathrm{F_p}$(field p) exists if and only if **GCD(a, p)=1**.

  In this implementation, the function known as Euler's Phi Function is used. If the prime-factorization of N (unique due to **Fundamental Theorem of Arithmetic**) is represented as:-

  $$N = {p_1}^{\mathrm{k_1}} {p_2}^{\mathrm{k_2}} ... {p_{\mathrm{m}}}^{\mathrm{k_m}} \tag{6}$$

  The Phi Function(Φ(N)) is defined as:-

  $$\Phi(N) = N \prod_{i=1}^{m} \left(1 - \frac{1}{p_{\mathrm{i}}}\right)^{\mathrm{k_i}} \tag{7}$$

Euler's Theorem states if GCD(m, N)=1:-

$$m^{\Phi(N)} \equiv 1 \pmod{N} \tag{8}$$

Dividing m both sides, we have:-

$$m^{\Phi(N)-1} \equiv m^{-1} \pmod{N} \tag{9}$$

Thus, modular inverse is obtained by modular exponentiation of base m, with exponent Φ(N)-1 and modulus N.

Alternative approach: According to Bezout's Theorem, there exists integers a and b for coprime integer pair (m, n) such that:

$$am + bn = GCD(m, n) = 1 \tag{10}$$

The integer coefficients a and b can be evaluated using Extended Euclid's algorithm for computing GCD (back tracing the steps to find GCD).

- **Modular Division**: Let the modulus be N and the modular inverse of b in F_N be l. Then, the modular division of a upon b is the modular multiplication of a and l in F_N.
- **RSA Key Generation**: Since, m and N are coprime, equations (5) and (8) hold true. Therefore, for some integer k, we have:-

$$m^{ed-1} \equiv \left(m^{\Phi(N)}\right)^k \pmod{N} \equiv 1 \pmod{N} \tag{11}$$

Thus, we have the relation as:-

$$ed \equiv 1 \pmod{\Phi(N)} \tag{12}$$

It is therefore, easy to see that the number of public keys (e) that can be generated is $\Phi(\Phi(N))$. It is due to the condition (12), in which private key (d) exists if and only if GCD(a, Φ(N))=1.

The private key (d) is the modular inverse of public key (e). The value e=1 should not be used, since the ciphertext becomes identical to the plaintext message and therefore, it does not provide any security. Hence, the number of possible keys is $\Phi(\Phi(N))$-1.

- **RSA Encryption and Decryption**: These functions are easily implemented using modular exponentiation as stated in equations (1) and (2), once the public key (e) and private key (d) is chosen.

During implementation, the private key computed by modular inverse is kept hidden from being displayed.

Note: In computation of Φ(N), the Euler's Phi Function is not used, rather it is the directly computed since its two prime factors are known.

5 Conclusion

On executing the final program, we can see both the encrypted and the decrypted texts. We also notice that the decrypted text is identical to the plaintext message in accordance to the correctness property.