MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Public Key Cryptographic Techniques
"Rivest, Shamir, Adleman's Algorithm (RSA)"

Lab Assignment 4

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

March 14, 2023

Contents

1 Aim

Write a program using JAVA or Python or C++ to implement RSA asymmetric key algorithm.

2 Objectives

To understand the concepts of public key and private key

3 Theory

3.1 Euler Totient Function

Euler's Totient function, denoted as $\varphi(n)$, is a number theoretic function that counts the number of positive integers less than or equal to n that are relatively prime to n. In other words, $\varphi(n)$ gives the number of integers in the range $1 \le k \le n$ such that $\gcd(k, n) = 1$.

Here is an example of how to calculate Euler's Totient function for a specific integer n:

Let's take $n = 12$. The prime factorization of n is $n = 2^2 \cdot 3^1$, so we can use the formula for calculating $\varphi(n)$ in terms of the prime factorization of n:

$$\varphi(n) = n \cdot \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

where the product is taken over all distinct prime factors of n. Plugging in the prime factorization of $n = 12$, we get:

$$\begin{aligned} \varphi(12) &= 12 \cdot \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{3}\right) \\ &= 12 \cdot \frac{1}{2} \cdot \frac{2}{3} = 4 \end{aligned}$$

Therefore, the value of Euler's Totient function for $n = 12$ is $\varphi(12) = 4$. This means that there are 4 positive integers less than or equal to 12 that are relatively prime to 12: 1, 5, 7, and 11.

3.2 Euclidean Algorithm

The Euclidean Algorithm is a method for finding the greatest common divisor (GCD) of two integers. It works by repeatedly finding the remainder when one integer is divided by the other, and then replacing the larger integer with the remainder. This process is repeated until the remainder is zero, at which point the GCD is the last non-zero remainder.

Here is an example of the Euclidean Algorithm applied to finding the GCD of 54 and 24:

$$\begin{aligned} 54 &= 2 \cdot 24 + 6 \\ 24 &= 4 \cdot 6 + 0 \end{aligned}$$

We start by dividing 54 by 24 and finding the remainder, which is 6. We then replace 54 with 24 and 24 with 6, and repeat the process by dividing 24 by 6 and finding the remainder, which is 0. Since the remainder is now zero, the GCD is the last non-zero remainder, which in this case is 6. Therefore, the GCD of 54 and 24 is 6.

3.3 Extended Euclidean Algorithm

The Extended Euclidean Algorithm is an extension of the Euclidean Algorithm that not only finds the GCD of two integers, but also finds two coefficients that can be used to express the GCD as a linear combination of the two integers. Specifically, given two integers a and b, the Extended Euclidean Algorithm finds integers x and y such that:

$$ax + by = \gcd(a, b)$$

Here is an example of the Extended Euclidean Algorithm applied to finding the GCD of 54 and 24, along with the coefficients x and y:

$$\begin{aligned} 54 &= 2 \cdot 24 + 6 \\ 24 &= 4 \cdot 6 + 0 \end{aligned}$$

To start, we apply the Euclidean Algorithm as we did before to find the GCD of 54 and 24, which is 6. We then work backwards through the steps of the algorithm to find the coefficients x and y.

Starting from the second-to-last step:

$$6 = 54 - 2 \cdot 24$$

We can rearrange this equation to isolate 6:

$$\begin{aligned} 6 &= 54 - 2 \cdot 24 \\ &= 54 - 2(54 - 2 \cdot 24) \\ &= 5 \cdot 54 - 2 \cdot 24 \end{aligned}$$

So, we have found that $x = 5$ and $y = -2$. Substituting these values into the original equation, we get:

$$54(5) + 24(-2) = 6$$

Therefore, the GCD of 54 and 24 is 6, and it can be expressed as a linear combination of 54 and 24 with coefficients 5 and -2, respectively.

3.4 RSA Algorithm

3.4.1 Key Generation

1. Selecting two large primes at random: p and q.
2. Computing their system modulus: $n = (p * q)$.
3. Compute: $\phi(n) = (p-1) * (q-1)$.
4. selecting at random the encryption key (public key): e such that $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$.
5. To find decryption key d such that $d * e \equiv 1 \pmod{\phi(n)}$.
6. Public Encryption key: $PU = e, n$
7. Private Decryption key: $PR = d, n$

3.4.2 RSA Encryption

Encrypt the plain text M using the public key PU.

1. Computes Cipher text: $C = M^e \pmod{n}$, where $0 <= M < n$.

3.4.3 RSA Decryption

Decrypt the cipher text C using the private key PR.

1. Computes Plaintext: $M = C^d \pmod{n}$

3.4.4 Example of RSA Encryption

1. Select two large primes at random: $p = 3$ and $q = 11$.
2. Compute their system modulus: $n = (p * q) = 33$.
3. Compute: $\phi(n) = (p - 1) * (q - 1) = 20$.
4. Select at random the encryption key (public key): $e = 7$ such that $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$.
5. To find decryption key d such that $d * e \equiv 1 \pmod{\phi(n)}$.
6. Public Encryption key: $PU = e, n = 7, 33$
7. Private Decryption key: $PR = d, n = 3, 33$
8. Encrypt the plain text $M = 30$ using the public key PU.
9. Computes Cipher text: $C = M^e \pmod{n} = 5^7 \pmod{33} = 24$.
10. Decrypt the cipher text $C = 24$ using the private key PR.
11. Computes Plaintext: $M = C^d \pmod{n} = 24^3 \pmod{33} = 30$.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Input and Output

```
Enter the Message to be encrypted:
This Assignment's Due date is very near
private key is:  (14633, 31373)
public key is:  (89, 31373)
<encrypted text>
This Assignment's Due date is very near
```

6 Code

```
import math
import random

# Pre generated primes
first_primes_list = [
2,
3,
5,
7,
11,
13,
17,
19,
23,
29,
31,
37,
41,
43,
47,
53,
59,
61,
67,
71,
73,
79,
83,
89,
97,
101,
103,
107,
109,
113,
127,
131,
137,
139,
149,
151,
157,
163,
167,
173,
179,
181,
191,
193,
197,
199,
211,
223,
227,
229,
233,
```

```
239,
241,
251,
257,
263,
269,
271,
277,
281,
283,
293,
307,
311,
313,
317,
331,
337,
347,
349,
]

# Iterative Function to calculate
# (a^n)%p in O(logy)


def power(a, n, p):

    # Initialize result
    res = 1

    # Update 'a' if 'a' >= p
    a = a % p

    while n > 0:

        # If n is odd, multiply
        # 'a' with result
        if n % 2:
            res = (res * a) % p
            n = n - 1
        else:
            a = (a**2) % p

            # n must be even now
            n = n // 2

    return res % p


def nBitRandom(n):
    return random.randrange(2 ** (n - 1) + 1, 2**n - 1)


def getLowLevelPrime(n):
    """Generate a prime candidate divisible
    by first primes"""
    while True:
        # Obtain a random number
        pc = nBitRandom(n)
```

```

        # Test divisibility by pre-generated
        # primes
        for divisor in first_primes_list:
            if pc % divisor == 0 and divisor**2 <= pc:
                break
        else:
            return pc


def isPrime(n, k):
    """
    If n is prime, then always returns true, If n is composite than returns false
    with
    high probability Higher value of k increases probability of correct result

    works on Primality Test by Fermat's Little Theorem:
    If n is a prime number, then for every a, 1 < a < n-1,

    a^(n-1) = 1 (mod n)
    OR
    a^n-1 % n = 1

    """
    # Corner cases
    if n == 1 or n == 4:
        return False
    elif n == 2 or n == 3:
        return True

    # Try k times
    else:
        for i in range(k):

            # Pick a random number
            # in [2..n-2]
            # Above corner cases make
            # sure that n > 4
            a = random.randint(2, n - 2)

            # Fermat's little theorem
            if power(a, n - 1, n) != 1:
                return False

    return True


def eucleadean_gcd(x, y):
    if y == 0:
        return x
    if x == 0:
        return y

    else:
        return eucleadean_gcd(y, x % y)


def extended_eucleadean(x, y):
    # y is smaller than x
```

```python
    # we need to return g, a, b such that
    # g = gcd(x, y) = ax + by

    if y == 0:
        return (x, 1, 0)
    else:
        g, a, b = extended_eucleadean(y, x % y)
        return (g, b, a - (x // y) * b)


def make_keys(prime_no_bits=1024):
    """
    Return a public and private key pair.
    returns: tuple (private_key, public_key)
    """
    # while True:
    #     p = int(input("Enter the value of p: "))
    #     q = int(input("Enter the value of q: "))
    #     check_p = isPrime(p, 10)
    #     check_q = isPrime(q, 10)
    #     if check_p and check_q:
    #         break
    #     print("Primality: p: ", check_p)
    #     print("Primality: q: ", check_q)

    while True:
        p = getLowLevelPrime(prime_no_bits)
        if isPrime(p, 20):
            break

    while True:
        q = getLowLevelPrime(prime_no_bits)
        if isPrime(q, 20):
            break

    # computing their system mod:
    n = p * q

    # computing phi n
    phi_n = (p - 1) * (q - 1)

    # computing random key e
    list_of_ees = []
    for i in range(2, phi_n):
        if eucleadean_gcd(i, phi_n) == 1:
            if len(list_of_ees) < 50:
                list_of_ees.append(i)
            else:
                break

    e = random.choice(list_of_ees)
    public_key = (e, n)

    # remainder 1 = a * phi_n + b * e
    g, a, b = extended_eucleadean(phi_n, e)

    if b < 0:
        b = b + phi_n
    d = b
```

```python
    private_key = (d, n)

    return (private_key, public_key)


def rsa_encryption(plain_text, key):
    """
    Algorithm to encrypt a integer via RSA.
    """
    e, n = key

    cipher_text = pow(plain_text, e) % n
    return cipher_text


def rsa_decryption(cipher_text, key):
    """
    Decrypts the cipher_text using key.
    """
    d, n = key
    plain_text = pow(cipher_text, d) % n
    return plain_text


if __name__ == "__main__":
    # making keys first
    private_key, public_key = make_keys(8)

    messages = []
    cipher_texts = []
    plain_texts = []

    print("Enter the Message to be encrypted: ")
    message = input()
    for i in message:
        messages.append(ord(i))

    # print("The message to be encrypted is: ", messages)

    print("private key is: ", private_key)
    print("public key is: ", public_key)

    # this will be done by some one else who has my public key
    for i in messages:
        cipher_text = rsa_encryption(i, public_key)
        cipher_texts.append(cipher_text)

    # print(cipher_texts)
    cipher_text = "".join([chr(i) for i in cipher_texts])
    print(cipher_text)

    # once I get cipher_text, I would then decrypt it using my private key.

    cipher_texts = [ord(i) for i in cipher_text]
    for i in cipher_texts:
        plain_text = rsa_encryption(i, private_key)
        plain_texts.append(plain_text)

    plain_texts = [chr(i) for i in plain_texts]
```

```
    plain_text = "".join(plain_texts)
    print(plain_text)
```

Listing 1: "RSA Algorithm"

7 Conclusion

Thus, learnt about the different kinds of public key cryptography works. Also, learnt about the RSA algorithm and its implementation in Python in depth. We also tried to implement RSA on a dummy client server model using sockets in python.

8 FAQ

1. *Compare symmetric key cryptography and asymmetric key cryptography*

   (a) *Symmetric Key Cryptography*
   - *Advantages*
     - *Fast*
     - *Easy to implement*
     - *Easy to share the key*
   - *Disadvantages*
     - *Key Distribution is a problem*
     - *Key Management is a problem*

   (b) *Asymmetric Key Cryptography*
   - *Advantages*
     - *Easy to share the key*
     - *Easy to manage the key*
   - *Disadvantages*
     - *Slow*
     - *Difficult to implement*

2. *Write advantages and disadvantages of RSA algorithm.*

   Advantages

   - *RSA is a public key algorithm* : The public key is available to everyone. The private key is kept secret. The public key is used for encryption and the private key is used for decryption.
   - *RSA is a secure algorithm* : The security of RSA is based on the difficulty of factoring large integers. The security of RSA depends on the fact that the factoring of the product of two large prime numbers is difficult. This is to say that it is a very difficult problem to find the two prime factors of a large composite number, and therefore it makes RSA very secure.
   - *RSA is a widely used algorithm* : RSA is used in many applications such as secure email, digital signatures, file encryption, etc.

   Disadvantages

   - *RSA is a slow algorithm* : The RSA algorithm is slow because of the large number of multiplications and modular exponentiations that are required.
   - *RSA is not suitable for bulk data encryption* : RSA is not suitable for bulk data encryption because of its slowness. It is suitable for encrypting small amounts of data.