ST. FRANCIS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY
SECURITY LAB

Experiment – 4: Implementation of RSA cryptosystem

Aim: To implement and analyze RSA cryptosystem

Objective: After performing the experiment, the students will be able to understand the encryption and decryption using RSA cryptosystem.

Lab objective mapped:

L502.2: Students should be able to Analyse and implement public key algorithms like RSA

.

Prerequisite: Basic knowledge of RSA algorithm.

Requirements: C/C++/JAVA/PYTHON

Pre-Experiment Theory:

RSA was invented by Ron Rivest, Adi Shamir, and Len Adleman and hence, it is termed as RSA cryptosystem. We will see two aspects of the RSA cryptosystem, firstly generation of key pair and secondly encryption-decryption algorithms.

Generation of RSA Key Pair

Each person or a party who desires to participate in communication using encryption needs to generate a pair of keys, namely public key and private key. The process followed in the generation of keys is described below −

1. Generate the RSA modulus (n)
    a. Select two large primes, p and q.
    b. Calculate $n=p\times q$. For strong unbreakable encryption, let n be a large number, typically a minimum of 512 bits.
2. Find Derived Number (e)
    a. Number e must be greater than 1 and less than $\phi(n) = (p - 1)(q - 1)$.
    b. There must be no common factor for e and $\phi(n)$ except for 1. In other words two numbers e and $\phi(n)$ are co-prime.
3. Form the public key
    a. The pair of numbers (e, n) form the RSA public key and is made public.
    b. Interestingly, though n is part of the public key, difficulty in factorizing a large prime number ensures that attacker cannot find in finite time the two primes (p & q) used to obtain n. This is strength of RSA.
4. Generate the private key
    a. Private Key d is calculated from p, q, and e. For given n and e, there is unique number d.
    b. Number d is the inverse of e modulo $\phi(n)$. This means that d is the number less than $\phi(n)$ such that when multiplied by e, it is equal to 1 modulo $\phi(n)$
    c. This relationship is written mathematically as follows –

$$e\times d = 1 \bmod \phi(n)$$

The Extended Euclidean Algorithm takes p, q, and e as input and gives d as output.

Encryption and Decryption

Once the key pair has been generated, the process of encryption and decryption are relatively straightforward and computationally easy.

RSA Encryption

- Suppose the sender wish to send some text message to someone whose public key is (e,n).
    - The sender then represents the plaintext as a series of numbers less than n.
- To encrypt the first plaintext P, which is a number modulo n. The encryption process is simple mathematical step as −

$$C = P^e \bmod n$$

- In other words, the ciphertext C is equal to the plaintext P multiplied by itself e times and then reduced modulo n. This means that C is also a number less than n.

RSA Decryption

- The decryption process for RSA is also very straightforward. Suppose that the receiver of public-key pair (e, n) has received a ciphertext C.
- Receiver raises C to the power of his private key d. The result modulo n will be the plaintext P.

$$\text{Plaintext} = C^d \bmod n$$

Procedure:

1. Write a program in C/C++/Java or Python for key generation, encryption and decryption using RSA algorithm.
   a. For Key generation ask user to enter the value of prime numbers p & q and a public key 'e'. (Note that values of p, q & e cannot be random, they should satisfy criteria as per RSA algorithm)
   b. Program should calculate private key element 'd' using Extended Euclidean Algorithm.
   c. Provide a set of public (e, n) and private key (d, n) as the output to the user.
   d. For RSA Encryption, ask user to enter the Message, and the public key. Provide Ciphertext as the output.
   e. For RSA Decryption, ask user to enter the ciphertext, and the private key. Provide retrieved message as the output.

2. Test the output of program for following Exercise:

   If party A and party B wish to use RSA to communicate securely. 'A' chooses public key (e, n) as (7, 247) and 'B' chooses public key (e, n) as (5, 221). Find
   a. Value of prime numbers p, q, $\phi(n)$, public key for party A.
   b. Calculate A's Private key.
   c. Value of prime numbers p, q, $\phi(n)$, public key for party B.
   d. Calculate B's Private Key.
   e. What will be the cipher-text sent by A to B, if A wishes to send M=5 to B
   f. Show how B will decrypt the message.
   g. What will be the cipher-text sent by B to A, if B wishes to send M=3 to A
   h. Show how A will decrypt the message.

Output:

1. Attach the complete code performing key generation, encryption and decryption.
2. Attach the program output for key generation (display public key & private key), encryption and decryption of all the inputs given in the exercise above.

```
import random
import math

def is_prime(num):
    if num <= 1:
        return False
    if num <= 3:
        return True if num % 2 == 0 or
    num % 3 == 0: return False
```

```
    i = 5 while i * i <=
    num:
        if num % i == 0 or num % (i + 2) == 0:
            return False
        i += 6
    return True
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
def extended_gcd(a, b):
    if a == 0: return (b,
    0, 1) else:
        g, x, y = extended_gcd(b % a, a)
        return (g, y - (b // a) * x, x)
def mod_inverse(a, m): g, x, _ = extended_gcd(a, m)
    if g != 1: raise Exception('Modular inverse does
    not exist') else:
        return x % m
def generate_keys(p, q, e):
    if not (is_prime(p) and is_prime(q)): raise
        Exception("Both p and q must be prime")
    n = p * q
    phi = (p - 1) * (q - 1)
    if gcd(e, phi) != 1:
        raise Exception("Public key 'e' is not coprime to phi(n)")

    d = mod_inverse(e, phi)

    public_key = (e, n)

    private_key = (d, n) return

    public_key, private_key

def encrypt(message, public_key): e, n =
    public_key ciphertext =
    [pow(int(message),e)%n ] return
    ciphertext
def decrypt(ciphertext, private_key): d, n
    = private_key message =
    [pow(int(ciphertext),d)%n ] return
    message
if __name__ == "__main__":
    p = int(input("Enter prime number 'p': "))
    q = int(input("Enter prime number 'q': "))
    e = int(input("Enter public key 'e' (must be coprime with (p-1)*(q-1)): "))
    public_key, private_key = generate_keys(p, q, e)
    print("Public Key (e, n):", public_key)
    print("Private Key (d, n):", private_key)
    message = input("Enter the message to encrypt: ")
    ciphertext = encrypt(message, public_key)
    print("Ciphertext:", ciphertext)
```

```
ciphertext =input("Enter the message to decrypt: ")
decrypted_message = decrypt(ciphertext, private_key)
print("Decrypted Message:", decrypted_message)
```

```
Enter prime number 'p': 19
Enter prime number 'q': 13
Enter public key 'e' (must be coprime with (p-1)*(q-1)): 7
Public Key (e, n): (7, 247)
Private Key (d, n): (31, 247)
Enter the message to encrypt: 3
Ciphertext: [211]
Enter the message to decrypt: 211
Decrypted Message: [3]
>
```

```
Enter prime number 'p': 13
Enter prime number 'q': 17
Enter public key 'e' (must be coprime with (p-1)*(q-1)): 5
Public Key (e, n): (5, 221)
Private Key (d, n): (77, 221)
Enter the message to encrypt: 5
Ciphertext: [31]
Enter the message to decrypt: 31
Decrypted Message: [5]
>
```

Post Experimental Exercise:

Solve the RSA exercise mentioned in procedure on journal sheets. [**Theoretical result and attached code's output should match**].

Conclusion:

RSA is a strong encryption algorithm. RSA implements a public-key cryptosystem that allows secure communications and digital signatures, and its security rests in part on the difficulty of factoring large numbers.