SUB: Information Security

AY 2023-24 (Semester-V)

Experiment No: 7

60009210105
Amitesh Sawarkar
D 12

Aim: To Implement Encryption and Decryption using RSA Algorithm.

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 –

- Generate the RSA modulus (n)
  - Select two large primes, p and q.
  - Calculate n=p*q. For strong unbreakable encryption, let n be a large number, typically a minimum of 512 bits.

- Find Derived Number (e)
  - Number e must be greater than 1 and less than $(p - 1)(q - 1)$.
  - There must be no common factor for e and $(p - 1)(q - 1)$ except for 1. In other words two numbers e and $(p – 1)(q – 1)$ are coprime.

- Form the public key
  - The pair of numbers (n, e) form the RSA public key and is made public.

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.

- Generate the private key
  - Private Key d is calculated from p, q, and e. For given n and e, there is unique number

SUB: Information Security

d.

○ Number d is the inverse of e modulo (p - 1)(q – 1). This means that d is the number less than (p - 1)(q - 1) such that when multiplied by e, it is equal to 1 modulo (p - 1)(q - 1).

○ This relationship is written mathematically as follows −

$ed = 1 \bmod (p - 1)(q - 1)$

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

Example

An example of generating RSA Key pair is given below. (For ease of understanding, the primes p & q taken here are small values. Practically, these values are very high).

● Let two primes be p = 7 and q = 13. Thus, modulus n = pq = 7 x 13 = 91.

● Select e = 5, which is a valid choice since there is no number that is common factor of 5 and $(p - 1)(q - 1) = 6 \times 12 = 72$, except for 1.

● The pair of numbers (n, e) = (91, 5) forms the public key and can be made available to anyone whom we wish to be able to send us encrypted messages.

● Input p = 7, q = 13, and e = 5 to the Extended Euclidean Algorithm. The output will be d = 29.

● Check that the d calculated is correct by computing −

$de = 29 \times 5 = 145 = 1 \bmod 72$

● Hence, public key is (91, 5) and private keys is (91, 29).

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 (n, e).

● 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

SUB: Information Security

reduced modulo n. This means that C is also a number less than n.

● Returning to our Key Generation example with plaintext P = 10, we get ciphertext C −
C = 105 mod 91

RSA Decryption

● The decryption process for RSA is also very straightforward. Suppose that the receiver of
public-key pair (n, e) 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.

Plaintext = Cd mod n

● Returning again to our numerical example, the ciphertext C = 82 would get decrypted to number 10 using private key 29 −

Plaintext = 8229 mod 91 = 10

```python
import math
import sympy


def read_file(file_path):
    file_path = "/content/Amitesh.txt"
    with open(file_path, "r") as file:
        file_contents = file.read()
    return file_contents



def isINT(num):
    if num % 1 == 0:
        return True
    return False


def isPrime(n):
    return sympy.isprime(n)
```

```python
def calc_e(t):
    for e in range(2, t):
        if math.gcd(e, t) == 1:
            return e
    return None


def calc_d(t, e):
    d = 0
    L = 0
    while True:
        d = (t * L + 1) / e
        if isINT(d):
            break
        L += 1
    return int(d)


def encrypt(public_key, message):
    e, n = public_key
    # pow(with three arguments) => x**y % z
    return pow(message, e, n)
```

```
def encrypt_string(public_key, input_string):
    list_ascii_p = format_message(input_string)
    print("list_ascii_numbers(BEFORE ENCRYPTION) : ", list_ascii_p)
    list_cipher_ascii = []
    for i in list_ascii_p:
        list_cipher_ascii.append(encrypt(public_key, i))
    print("list_ascii_numbers(AFTER ENCRYPTION) : ", list_cipher_ascii)
    return list_cipher_ascii


def decrypt(private_key, cipher_text):
    d, n = private_key
    # pow(with three arguments) => x**y % z
    p = pow(cipher_text, d, n)
    return p


def decrypt_string(private_key, list_cipher_ascii):
    list_decrypted_ascii = []
    for i in list_cipher_ascii:
        list_decrypted_ascii.append(decrypt(private_key, i))
    return list_decrypted_ascii

def format_message(input_string):
    list_ascii = []
    for i in range(len(input_string)):
        list_ascii.append(ord(input_string[i]))
    return list_ascii


def covert_ascii_to_char(list_items):
    char_list = ""
    for i in range(len(list_items)):
        char_list += (chr(list_items[i] % 0x10ffff))
    return char_list
```

```
def RSA(p, q, plain_text=""):
    n = p * q
    t = (p - 1) * (q - 1)
    e = calc_e(t)
    d = calc_d(t, e)
    publicKey = (e, n)
    print("Public Key ( e= ", e, ", n= ", n, ")")
    privateKey = (d, n)
    print("Private Key ( d= ", d, ", n= ", n, ")")
    if plain_text == "":
        plain_message = read_file('msg.txt')
    else:
        plain_message = plain_text
    print("Encrypting (", plain_message, ") ...")
    cipher_message = encrypt_string(publicKey, plain_message)
    print("Decrypting (", cipher_message, ") ...")
    p = decrypt_string(privateKey, cipher_message)
    print("plain_message(ASCII) : ", p)
    print("plain_message(CHAR) : ", covert_ascii_to_char(p))
    return cipher_message, p
```

```
def main():
    p = int(input("Enter p : "))
    if not isPrime(p):
        raise ValueError("P is not prime")
    q = int(input("Enter q : "))
    if not isPrime(q):
        raise ValueError("q is not prime")
    flag = input("Enter:-\n\t- 'f' to read from file => msg.txt.\n\t- 'm' to enter plaintext message.\n")
    if flag == 'f':
        Cypher, Plain = RSA(p, q)
    elif flag == 'm':
        m = input("Enter plaintext message:")
        Cypher, Plain = RSA(p, q, m)
    else:
        print("Error !")
        return None
    return Cypher, Plain

if __name__ == '__main__':
    Cypher, Plain = main()
    print("Encrypted (Cypher text) : ", ', '.join(map(str, Cypher)))
    print("Decrypted (Plain text) : ", ', '.join(map(str, Plain)))
```

SUB: Information Security

Conclusion:

In conclusion, the experiment successfully implemented RSA encryption and decryption, illustrating the key components of this widely used cryptographic algorithm. It showcased the importance of key generation, secure communication, and the mathematical principles that underlie RSA's security. While this experiment used small values for simplicity, in practice, larger prime numbers are essential for robust security. RSA remains a fundamental tool in safeguarding digital communications and data integrity over untrusted networks, with key management being a crucial consideration in real-world applications.