Experiment No:2

Implementation and analysis of RSA cryptosystem

Course Outcome [CSL602.2]: Implement symmetric and asymmetric key cryptography

Aim: Implementation and analysis of RSA cryptosystem

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 –
- 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.
- 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 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 —

$$C = Pe \bmod n$$

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.

Returning to our Key Generation example with plaintext P = 10, we get ciphertext C —

$$C = 105 \bmod 91$$

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

Implementation:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int checkPrime(int n) {
int i;
int m = n / 2;
for (i = 2; i <= m; i++) {
if (n % i == 0) {
return 0; // Not Prime
}}
return 1; }
int findGCD(int n1, int n2) {
int i, gcd;
for(i = 1; i <= n1 && i <= n2; ++i) {
if(n1 % i == 0 && n2 % i == 0)
gcd = i;
}return gcd;
}int powMod(int a, int b, int n) {
long long x = 1, y = a;
while (b > 0) {
if (b % 2 == 1)
x = (x * y) % n;
y = (y * y) % n;
b /= 2;
}return x % n;
}int main(int argc, char* argv[]) {
int p, q, n, phin, data, cipher, decrypt;
while (1) {
```

```
printf("Enter any two prime numbers: ");
scanf("%d %d", &p, &q);
if (!(checkPrime(p) && checkPrime(q)))
printf("Both numbers are not prime. Please enter prime numbers only...\n");
else if (!checkPrime(p))
printf("The first prime number you entered is not prime, please try again...\n");
else if (!checkPrime(q))
printf("The second prime number you entered is not prime, try again...\n");
else
break;
}n = p * q;
phin = (p - 1) * (q - 1);
int e;
printf("Enter the value of e: ");
scanf("%d", &e);
for (e = 5; e <= 100; e++) {
if (findGCD(phin, e) == 1)
break;
}int d = 0;
for (d = e + 1; d <= 100; d++) {
if ( ((d * e) % phin) == 1)
break;
}printf("Value of e: %d\nValue of d: %d\n", e, d);
printf("Enter Plaintext:");
scanf("%d", &data);
cipher = powMod(data, e, n);
printf("The cipher text is: %d\n", cipher);
decrypt = powMod(cipher, d, n);
printf("The decrypted text is: %d\n", decrypt);
```

return 0; }

Output:

Enter any two prime numbers: 3

11

Enter the value of e: 7

Value of e: 7

Value of d: 23

Enter Plaintext:31

The cipher text is: 4

The decrypted text is: 31

Conclusion: RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and Private key is kept private.