

## Program – 8

**AIM:** To implement a program to show Encryption and Decryption in the RSA Algorithm.

### **Introduction and Theory**

---

RSA is a cryptosystem for public-key encryption, and is widely used for securing sensitive data, particularly when being sent over an insecure network such as the Internet.

RSA was first described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology. Public-key cryptography, also known as asymmetric cryptography, uses two different but mathematically linked keys, one public and one private. The public key can be shared with everyone, whereas the private key must be kept secret. In RSA cryptography, both the public and the private keys can encrypt a message; the opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric algorithm: It provides a method of assuring the confidentiality, integrity, authenticity and non-reputability of electronic communications and data storage.

Many protocols like SSH, OpenPGP, S/MIME, and SSL/TLS rely on RSA for encryption and digital signature functions. It is also used in software programs -- browsers are an obvious example, which need to establish a secure connection over an insecure network like the Internet or validate a digital signature. RSA signature verification is one of the most commonly performed operations in IT.

#### Explaining RSA's popularity

RSA derives its security from the difficulty of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy, but determining the original prime numbers from the total -- factoring -- is considered infeasible due to the time it would take even using today's super computers.

The public and the private key-generation algorithm is the most complex part of RSA cryptography. Two large prime numbers,  $p$  and  $q$ , are generated using the Rabin-Miller primality test algorithm. A modulus  $n$  is calculated by multiplying  $p$  and  $q$ . This number is used by both the public and private keys and provides the link between them. Its length, usually expressed in bits, is called the key length. The public key consists of the modulus  $n$ , and a public exponent,  $e$ , which is normally set at 65537, as it's a prime number that is not too large. The  $e$  figure doesn't have to be a secretly selected prime number as the public key is shared with everyone. The private key consists of the modulus  $n$  and the private exponent  $d$ , which is calculated using the Extended Euclidean algorithm to find the multiplicative inverse with respect to the totient of  $n$ .

## Program – 8

### Algorithm: Generate an RSA key pair.

INPUT: Required modulus bit length,  $kk$ .

OUTPUT: An RSA key pair  $((N,e),d)$  where  $N$  is the modulus, the product of two primes ( $N=pq$ ) not exceeding  $kk$  bits in length;  $e$  is the public exponent, a number less than and coprime to  $(p-1)(q-1)$ ; and  $d$  is the private exponent such that  $ed \equiv 1 \pmod{(p-1)(q-1)}$ .

1. Select a value of  $e$  from 3,5,17,257,65537,5,17,257,65537
2. repeat
3.  $p \leftarrow \text{genprime}(k/2)$
4. until  $(p \bmod e) \neq 1$
5. repeat
6.  $q \leftarrow \text{genprime}(k - k/2)$
7. until  $(q \bmod e) \neq 1$
8.  $N \leftarrow pq$
9.  $L \leftarrow (p-1)(q-1)$
10.  $d \leftarrow \text{modinv}(e, L)$
11. return  $(N,e,d)$

### Encryption

Sender A does the following:-

1. Obtains the recipient B's public key  $(n,e)$ .
2. Represents the plaintext message as a positive integer  $m$  with  $1 < m < n$ .
3. Computes the ciphertext  $c = m^e \bmod n$ .
4. Sends the ciphertext  $c$  to B.
- 5.

### Decryption

Recipient B does the following:-

1. Uses his private key  $(n,d)$  to compute  $m = c^d \bmod n$ .
2. Extracts the plaintext from the message representative  $m$ .

### Code

```
1  #include<iostream>
2  #include<cstdio>
3  #include<math.h>
4  #include<string.h>
5  #include<stdlib.h>
6
7  using namespace std;
8
9  long int p, q, n, t, flag, e[100], d[100], temp[100], j, m[100],
10 en[100], i;
11 char msg[100];
12 int prime(long int);
13 void ce();
14 long int cd(long int);
15 void encrypt();
16 void decrypt();
17 int prime(long int pr)
18 {
19     int i;
20     j = sqrt(pr);
21     for (i = 2; i <= j; i++)
```

## Program – 8

```
22     {
23         if (pr % i == 0)
24             return 0;
25     }
26     return 1;
27 }
28
29 void ce()
30 {
31     int k;
32     k = 0;
33     for (i = 2; i < t; i++)
34     {
35         if (t % i == 0)
36             continue;
37         flag = prime(i);
38         if (flag == 1 && i != p && i != q)
39         {
40             e[k] = i;
41             flag = cd(e[k]);
42             if (flag > 0)
43             {
44                 d[k] = flag;
45                 k++;
46             }
47             if (k == 99)
48                 break;
49         }
50     }
51 }
52
53 long int cd(long int x)
54 {
55     long int k = 1;
56     while (1)
57     {
58         k = k + t;
59         if (k % x == 0)
60             return (k / x);
61     }
62 }
63 void encrypt()
64 {
65     long int pt, ct, key = e[0], k, len;
66     i = 0;
67     len = strlen(msg);
68     while (i != len)
69     {
70         pt = m[i];
71         pt = pt - 96;
72         k = 1;
73         for (j = 0; j < key; j++)
74         {
75             k = k * pt;
76             k = k % n;
77         }
78         temp[i] = k;
```

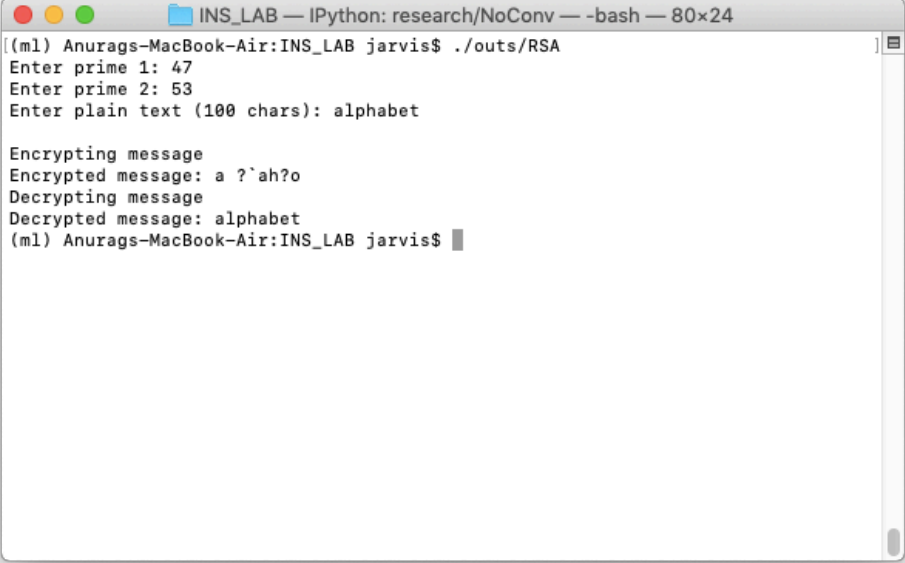
## Program – 8

```
79         ct = k + 96;
80         en[i] = ct;
81         i++;
82     }
83     en[i] = -1;
84     cout << "Encrypted message: ";
85     for (i = 0; en[i] != -1; i++)
86         printf("%c", en[i]);
87 }
88 void decrypt()
89 {
90     long int pt, ct, key = d[0], k;
91     i = 0;
92     while (en[i] != -1)
93     {
94         ct = temp[i];
95         k = 1;
96         for (j = 0; j < key; j++)
97         {
98             k = k * ct;
99             k = k % n;
100         }
101         pt = k + 96;
102         m[i] = pt;
103         i++;
104     }
105     m[i] = -1;
106     cout << "Decrypted message: ";
107     for (i = 0; m[i] != -1; i++)
108         printf("%c", m[i]);
109 }
110
111 int main()
112 {
113     cout << "Enter prime 1: ";
114     cin >> p;
115     flag = prime(p);
116
117     if (flag == 0)
118     {
119         cout << "Entered Value is not PRIME Exiting";
120         exit(1);
121     }
122
123     cout << "Enter prime 2: ";
124     cin >> q;
125     flag = prime(q);
126     if (flag == 0 || p == q)
127     {
128         cout << "Entered Value is not PRIME Exiting";
129         exit(1);
130     }
131
132     cout << "Enter plain text (100 chars): ";
133     fflush(stdin);
134     cin >> msg;
135     for (i = 0; msg[i] != '\0'; i++)
```

## Program – 8

```
136         m[i] = msg[i];
137     n = p * q;
138     t = (p - 1) * (q - 1);
139     ce();
140     cout << endl << "Encrypting message" << endl;
141     encrypt();
142     cout << endl << "Decrypting message" << endl;
143     decrypt();
144     cout << endl;
145     return 0;
146 }
```

## Results and Outputs:



A terminal window titled "INS\_LAB — IPython: research/NoConv — -bash — 80x24" showing the execution of a program. The user enters prime values and a message, and the program outputs the encrypted and decrypted results.

```
(ml) Anurags-MacBook-Air:INS_LAB jarvis$ ./outs/RSA
Enter prime 1: 47
Enter prime 2: 53
Enter plain text (100 chars): alphabet

Encrypting message
Encrypted message: a ?`ah?o
Decrypting message
Decrypted message: alphabet
(ml) Anurags-MacBook-Air:INS_LAB jarvis$
```

## Findings and Learnings:

1. We have implemented RSA encryption and decryption.