REGNO : 18BIT0101

NAME : KUHOO SHARMA

Network and Information Security

ITE4001

Introduction :

RSARSA (Rivest–Shamir–Adleman) Algorithm is a well known and widely used public-key encryption algorithm based on exponentiation in a finite (Galois) field over integers modulo a prime.

RSA implements a public-key cryptosystem, as well as digital signatures. RSA has applications in various fields from Bluetooth, MasterCard, VISA, e-banking, e-communication, e-commerce platforms to a range of web browsers, email, VPNs, chat and other communication channels.

How does RSA work :

Key Generation:

For RSA we generate both a **public key** and a **private key**. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key.

1. Choose two distinct large random prime numbers **p** and **q**.
2. Calculate **N**=pq
   - This N is used as the modulus for both the public and private keys
3. Calculate **φ(N)**
   - φ(N)=(p-1)(q-1)

4. Choose an integer **e** such that :
    - 1<e<φ(N) and HCF(highest common factor) of e and φ(N) is 1.
    - e is released as the public key exponent
5. Calculate **d** :
    - d*e=1 (mod φ(N))
    - d is the private key exponent

Encryption:

KEYS OF SENDER A : (e,N)

Ciphertext C is computed as :

$$c=m^e \pmod n$$

C is then transmitted to Receiver B

Decryption:

KEYS OF RECEIVER B : (d,N)

Ciphertext C is deciphered as :

$$m=c^d \pmod n$$

Then original message m is deciphered and received successfully.

Security in RSA Algorithm:

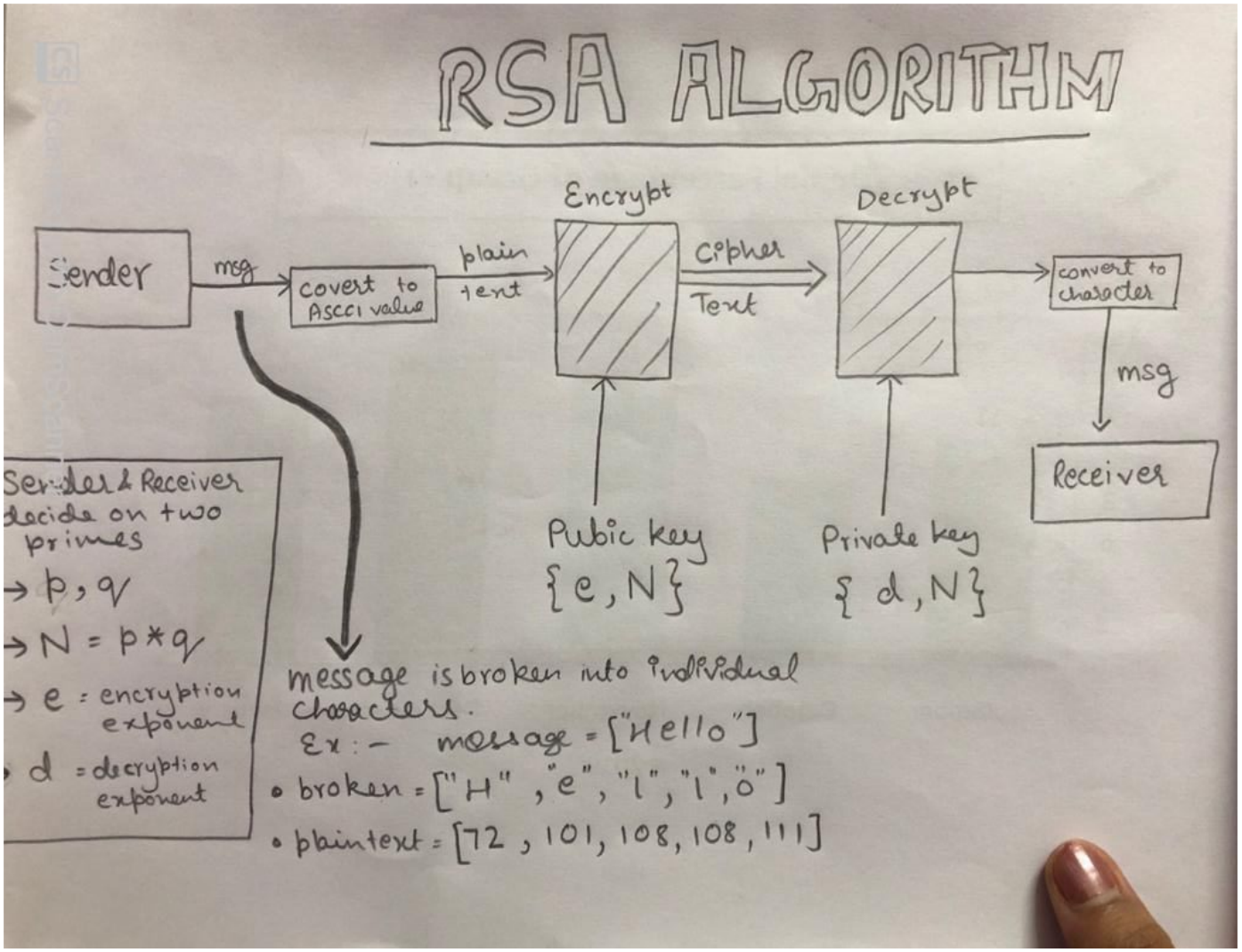
a. Brute force: This involves trying all possible private keys. RSA is generally used with larger keys

b. Mathematical attacks: There are several approaches, all equivalent in effort to factoring the product of two primes.

c. Timing attacks:Decreasing the computational complexity helps

d. Protocol attacks: Protocol attacks exploit weaknesses in the way RSA is being used.

Architecture :



Implementation of RSA as a Digital Signature mechanism :

A digital signature algorithm uses a public key system. The Sender signs his/her message with his/her private key and the Receiver verifies it with the transmitter's public key. A digital signature can provide message **authentication**, message **integrity** and **non-repudiation** services.

I have used relatively smaller numbers p(17) and q(11) for implementation because of the complexity constraints.

The original message is divided character by character to encode the ascii value of each for transmission.

- A creates her digital signature (**Sig**) as:

Sig=data^d mod N

(where data is the original message)

- A sends Message data and Signature Sig to B
- B computes Message as:

Message=Sig^e mod n

- If Message=data then B accepts the data sent by A.

CODE:

```
def gcd(a,b):
        if b==0:
                return a
        else:
                return gcd(b,a%b)
print("\nWelcom to RSA IMPLEMENTATION\n")

p = int(input('Enter the value of p = '))
q = int(input('\nEnter the value of q = '))
String = input('\nEnter the value of text = ')

l=[]
ciphertextlist =[]
deciphertextlist=[]
for i in String:
   l.append(ord(i))

n = p*q
t = (p-1)*(q-1)

for e in range(2,t):
   if gcd(e,t)== 1:
      break

for i in range(1,10):
   x = 1 + i*t
   if x % e == 0:
      d = int(x/e)
      break
print('\nn = '+str(n)+'\n\ne = '+str(e)+'\n\nt = '+str(t)+'\n\nd =
'+str(d)+'\n\ninput(data) ==>'+String+'\n\n')
for data in l:

   Sig = pow(data,e) % n
   ciphertextlist.append(chr(Sig))

   Message = pow(Sig,d) % n
```

```
    deciphertextlist.append(chr(Message))

    print(' cipher text = '+str(Sig)+' decrypted text = '+str(Message))
print("\nFinal Cipher Text (Sig)= ")
print(''.join(ciphertextlist))
print("\nFinal Deciphered Text (Message)= ")
print(''.join(deciphertextlist))
if(Message==data):
    print('message==data hence signature and data is accepted')
```

OUTPUT SCREENSHOT: