

ADVANCED RSA IMPLEMENTATION

USING LARGE PRIME NUMBERS AND KEY GENERATION

BATCH-A GROUP-14



Team Members

01

AKSHAYAA BK

CB.EN.U4AIE21002

02

GAJULA SRI VATSANKA

CB.EN.U4AIE21010

03

M.D.S.RAMA SARAN

CB.EN.U4AIE21034

04

R.SAI RAGHAVENDRA

CB.EN.U4AIE21049



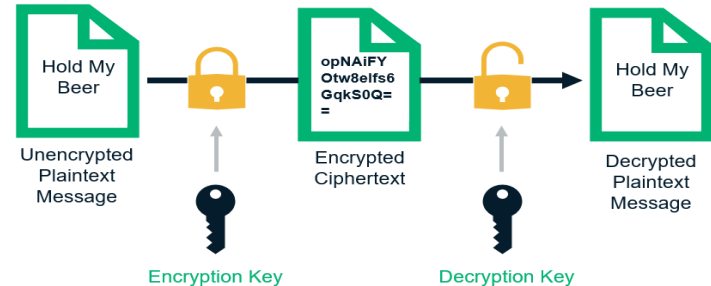
WHAT IS ENCRYPTION?

- Data can be protected via encryption by being changed into a format that can only be unlocked by an authorized user with the necessary decryption keys.
- Numerous encryption techniques exist, and they can be divided into groups according to a number of factors, including key management and usage. The following are some typical forms of encryption:

Types of Encryption

- Symmetric encryption
- Asymmetric encryption

How Encryption Works



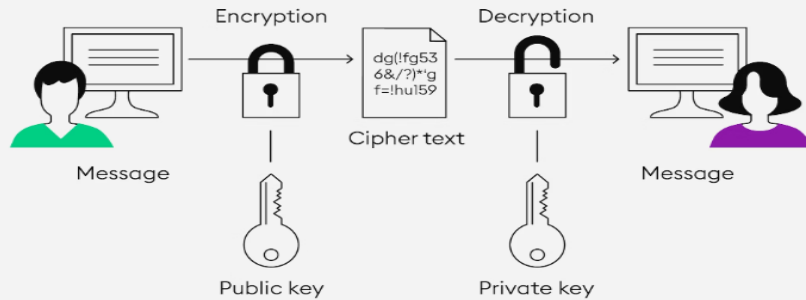


Asymmetric Key Encryption

- In Asymmetric Encryption algorithms, you use two different keys, one for encryption and the other for decryption. The key used for encryption is the public key, and the key used for decryption is the private key.

Working of Asymmetric key encryption

ASYMMETRIC ENCRYPTION



Step 1: sender uses receivers public key to encrypt the message

Step 2: The encrypted message is sent to receiver

Step 3: receiver uses his private key to decrypt the message



RSA(Rivest-shamir-adleman)

- It is a type of asymmetric key encryption.
- RSA gets its security from the difficulty of factoring large numbers.
- The public and private keys are functions of a pair of large (100 to 200 digits or even larger) prime numbers.
- Recovering the plaintext from the public key and the ciphertext is conjectured to be equivalent to factoring the product of the two primes

RSA ALGORITHM

KEY GENERATION

- Generate two large distinct prime numbers, p and q .
- Let $n = p * q$.
- Let $m = \phi(n) = (p - 1)(q - 1)$. (ϕ is totient function).
- Choose a small number e , co prime to $\phi(n)$, with $\text{GCD}(m, e) = 1$ and $1 < e < m$
- Find d which satisfies $d \times e \text{ mode } m = 1$.
- Public key = $\{e, n\}$
Private key = $\{d, n\}$

What is $\varphi(n)$?

- the totient function calculates the number of positive integers less than or equal to "n" that do not share any common factors (other than 1) with "n".
- The totient function $\varphi(n)$ can be defined as follows:
- $\varphi(n)$ = the count of positive integers k , where $1 \leq k \leq n$, and $\gcd(n, k) = 1$
- For example, if we calculate $\varphi(8)$, we are determining the number of positive integers from 1 to 8 that are coprime to 8:
- $\varphi(8) = \{1, 3, 5, 7\}$
- There are four positive integers (1, 3, 5, and 7) that are coprime to 8, so $\varphi(8) = 4$.



Encryption

$$\text{Cipher} = (\text{message})^e \bmod n$$

Decryption

$$\text{message} = (\text{cipher})^d \bmod n$$

RSA Example

STEP-1: $P=7, q=11$

STEP-2: $n = p*q = 7*11 = 77$

STEP-3: $\phi(n) = (P - 1)(q - 1)$
 $= (7-1)(11-1)$
 $= 6*10$
 $= 60$

STEP-4: Assume e such that it is a co-prime to $\phi(n)$ with $\text{GCD}(\phi(n), e) = 1$ $\text{Gcd}(7, 60) = 1$

$e = 7$

STEP-5: $d * e \bmod \phi(n) = 1$.

$$d * 7 \bmod 60 = 1$$

$$d = 43$$

STEP-6: *Public key:* $\{7, 77\}$

Private key: $\{43, 77\}$

STEP-7: *Encryption:*

Consider plain text $p = 9$

$$c = 9^7 \bmod 77$$

$$= 37$$

Decryption:

$$p = 37^{43} \bmod 77$$

$$= 9$$

RSA-PYTHON CODE

```
import random
import time

import math
from sympy import isprime
from Crypto.PublicKey import RSA as CryptoRSA
from Crypto.Cipher import PKCS1_OAEP
from base64 import b64encode

class RSA:
    def __init__(self): self.bitlength = 1024
        self.r = random.SystemRandom()
        self.generate_primes()
        self.generate_key_pairs()

    def generate_primes(self):
        self.p = self.random_prime()
        print(f"The value of prime number p is: {self.p}")
        if isprime(self.p):
            print("The big integer p is a probable prime number")
        else:
            print("The big integer p is not a prime number...please execute again")
        print(f"The length of p is - {len(str(self.p))}")

    self.q = self.random_prime()
    print(f"The value of prime number q is: {self.q}")
    if isprime(self.q):
        print("The big integer q is a probable prime number")
    else:
        print("The big integer q is not a prime number...please execute again")
    print(f"The length of q is - {len(str(self.q))}")

    def random_prime(self):
        while True:
```

```
num = self.r.getrandbits(self.bitlength)
if isprime(num):
    return num
```

```
def generate_key_pairs(self):
self.n = self.p * self.q
    print(f"The value of prime number n is: {self.n}")
    print(f"The length of n is - {len(str(self.n))}")
```

```
phi = (self.p - 1) * (self.q - 1) e = self.random_coprime(phi)
```

```
    while math.gcd(phi, e) > 1 and e < phi: e += 1
self.e = e
self.d = pow(e, -1, phi) # Calculate the modular multiplicative inverse of e modulo phi
```

```
def random_coprime(self, phi): while True:
    num = self.r.randint(2, phi - 1) if math.gcd(phi, num) == 1:
        return num
```

```
def encrypt(self, plaintext):
    rsa_key = CryptoRSA.construct((self.n, self.e))
    cipher = PKCS1_OAEP.new(rsa_key)
    ciphertext = cipher.encrypt(plaintext)
    return ciphertext
```

```
def decrypt(self, ciphertext):
    rsa_key = CryptoRSA.construct((self.n, self.e, self.d))
    cipher = PKCS1_OAEP.new(rsa_key)
    plaintext = cipher.decrypt(ciphertext)
    return plaintext
```

```

def main():
    start = int(time.time() * 1000)
    teststring = input("Enter the text to be encrypted: ")
    print("----- Generating very large prime numbers of given bitlength")
    rsa = RSA()

    print("\n----- Encrypting text")
    encrypted = rsa.encrypt(teststring.encode())
    print("Encrypted String:", b64encode(encrypted).decode())

    print("\n----- Decrypting text")
    decrypted = rsa.decrypt(encrypted)
    print("Decrypted String:", decrypted.decode())

    if teststring == decrypted.decode(): end = int(time.time() *
    1000)
    print(f"\nx----- RSA Algorithm is successful x")
    print(f"The run time for bitlength {rsa.bitlength} is {(end - start) / 1000:.2f} seconds")
    else:
    print(f"\nx----- RSA Algorithm is unsuccessful x")

if __name__ == "__main__":
    main()

```

Server code


```
import socket
from Crypto.PublicKey import RSA as CryptoRSA
from Crypto.Cipher import PKCS1_OAEP

class ServerRSA:
    def __init__(self):
        self.bitlength = 1024
        self.private_key = None
        self.public_key = None
        self.rsa_keygen()

    def rsa_keygen(self):
        rsa_key = CryptoRSA.generate(self.bitlength)
        self.private_key = rsa_key
        self.public_key = rsa_key.publickey()

def server_program():
    host = socket.gethostname()
    port = 5004
    server_socket = socket.socket()
    server_socket.bind((host, port))
    server_socket.listen(1)
    conn, address = server_socket.accept()
    print("Connection from: " + str(address))

    server_rsa = ServerRSA()
    conn.send(server_rsa.public_key.export_key()) # Send the server's public key to the client
```



```
client_public_key_str = conn.recv(4096)
client_rsa_key = CryptoRSA.import_key(client_public_key_str)

while True:
    message = conn.recv(4096)
    if not message:
        break

    cipher = PKCS1_OAEP.new(server_rsa.private_key)
    print(message)
    decrypted_message = cipher.decrypt(message)
    print("Received and Decrypted message from client: " + decrypted_message.decode())

    # Process the data (if needed)
    response = input("Enter a response to send to the client: ")
    cipher = PKCS1_OAEP.new(client_rsa_key)
    encrypted_response = cipher.encrypt(response.encode())

    conn.send(encrypted_response)

conn.close()

if __name__ == '__main__':
    server_program()
```


Client code

```
import socket
from Crypto.PublicKey import RSA as CryptoRSA
from Crypto.Cipher import PKCS1_OAEP

def client_program():
    host = socket.gethostname()
    port = 5004

    client_socket = socket.socket()
    client_socket.connect((host, port))


    rsa_key = CryptoRSA.generate(1024) # Generate a new key pair for the client
    server_public_key_str = client_socket.recv(4096)
    server_public_key = CryptoRSA.import_key(server_public_key_str)

    client_socket.send(rsa_key.publickey().export_key()) # Send the client's public key to the server

    while True:
        message = input("Enter a message to send to the server: (to quit ctrl+c): ")

        cipher = PKCS1_OAEP.new(server_public_key)
        encrypted_message = cipher.encrypt(message.encode())
        client_socket.send(encrypted_message)

        response = client_socket.recv(4096)
        cipher = PKCS1_OAEP.new(rsa_key)
        decrypted_response = cipher.decrypt(response)
```

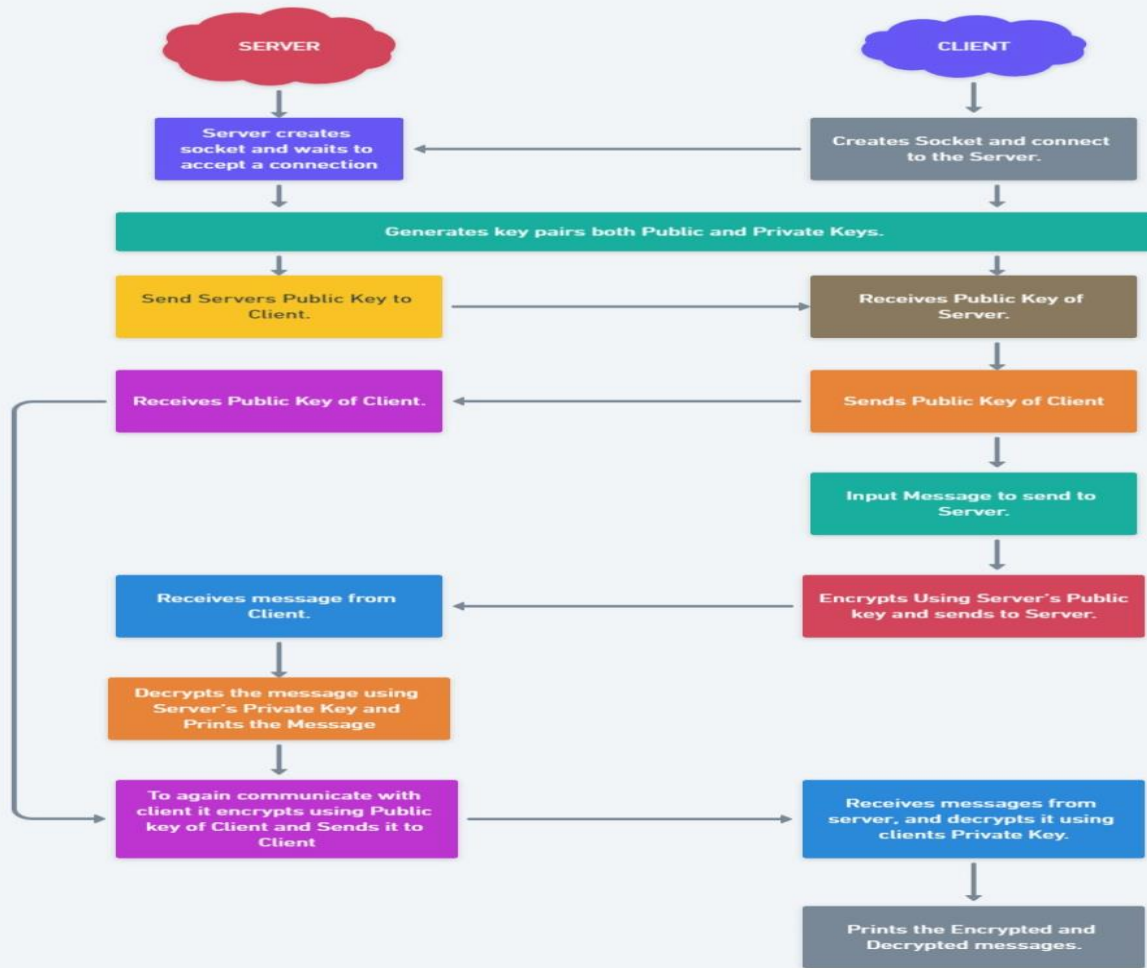


```
# Print the encrypted response received from the server
    print("Encrypted response received from the server: " + repr(response))

    print("Received decrypted response from the server: " + decrypted_response.decode())

client_socket.close()

if __name__ == '__main__':
    client_program()
```



RSA-output

```
Enter the text to be encrypted: rsa algorithm
----- Generating very large prime numbers of given bitlength -----
The value of prime number p is: 6508862561096524604828379065727849240118506768314575
The big integer p is a probable prime number
The length of p is - 308
The value of prime number q is: 9621638688643565368666334989069832715146173176196560
The big integer q is a probable prime number
The length of q is - 307
The value of prime number n is: 6262592383690996337389123382346909557368545802672218
The length of n is - 615

----- Encrypting text -----
Encrypted String: A5sgRjyRSgXRCDfCrI2a9jbnUvb/wCiKeZdP+Pvkjoq5KQGGrz9VhZ6EWW063lJfV5

----- Decrypting text -----
Decrypted String: rsa algorithm

x----- RSA Algorithm is successful -----x
The run time for bitlength 1024 is 16.98 seconds
```

Server-output

```
PS C:\Users\Dell\Downloads\finalday> python3 .\serversfin.py
Connection from: ('10.12.85.147', 51150)
Encrypted response received from client: b'\x1c\xdf\x80\x9c\xfd\xe4e?\xea7\xb2&\x8c\x9f5\x9b\t\x12\xab5\xfc\x1c\xfb\xfa4\xacI\xb5\xc0Mlh\x88>\x99\x1e\xe3\x03\rDL\xe3QA\x00\t\xff\x1c2\xdd\xfd\x14fi$\xa6\xdf\xb0b2xg\xd0G\xc5\x10\xec\xfa6\xa8\rqP1\xbfrvk\xfa5t2\x00J\xd4\xe2\x00\xe4X\x0e\x9d\xe1_\xd4\x81_\xdd\xc5\x83\xfa9\x1f!\x13@V8#\x92\x84X\xee.\xa1\xfa-\x17\xa8\xc3bt\xd25?\xa7T\x99@\xe6}\xc7'
Received and Decrypted message from client: hi
Enter a response to send to the client: hello
Encrypted response received from client: b'Wj9\x85\x16r\x84\xb34\xa2\xc6\x148/l\x7f\xel1tu\xee\x14\xfa1\x04W\x8f\x14=K?\xb1m\x8a\xbe\xa3\x99FY\xba\xc9l\x9c&\xb3#w$\xe9\xbf\xd8\x0c\x7mY($\xd3T\xea8\xe4\x89\x12\xc1:\xbd\xc1\xc8\x9c\x7f$\xa9\xd3\xa8;\n\xc4[\xf4\xee\r\x7e7\x0b\xfa8\x99_\x0eY\x89m\xd2\xd8\xe9\n\xa8Q\x82}L\xb8\n\x17.\xa8\xe7R32%\xfab5\xa8[\xe6-\xd1\xdf\xc3\n\xfa9U\x16,#D\x1b\xb3'
Received and Decrypted message from client: 123445bksvdbk
Enter a response to send to the client: working fine
█
```

Client-output

```
Enter a message to send to the server: (to quit ctrl+c): hi
Encrypted response received from the server: b"\xc4\x97\xe6\r\xac\xfa2\xcf\x0b\xdb\xee\x8c\xfa\x01\xfa8\xb0\x1e\x15&c?\x0c\xb8o\x0c\x05\xd7\x97\x97\x19\xfa\xab2h\x94L\xb0\xd3pR\xc1M\xfa3{\x83c\x18Tc\xa2\xeb'\xd4\xfa8\xe8\xca\xea\x99IA\x07\x0b)i\x10\xeaF\xe3{\x8f>\x9e\x81dV\xd9\t\xbe \xe6Qy\x0e\x02\xe4\xc6\xa4\x0c\x98|Q\x17Ba\xdc6\x17z\xb6I\xa7P\xfa5\xfa\x99\xa9\xad\x00\xfa2(w&\xfek\xda\x08\x15r\x81\x83h\xbb\xad\x8e\xd3Z\x11\x97#\x9e"
Received decrypted response from the server: hello
Enter a message to send to the server: (to quit ctrl+c): 123445bksvdbk
Encrypted response received from the server: b'p\x83\xeb\x00G\x99\xc0\xb2dJ\x15\xc2\xe6o\\\x1b\xb4\xfa\n\x9b}\x1b\xda\x1aEI\xc4\x10j\xb8\x12\xed\x9b\xbdJ\xbb\xb0\xbcE\x8c"<\xa4\xd6t\xed0\x9c_\xff\xb1\xa1\xc3?e\x06J\x81\xa0\xe8\xb3"\xe8\xa5M\x07\xb5>=&53|\x9b\xfe?\x14*1s\xb7\x0c\x9chf\xcd0\t@t\xeb\xce5\xfa5\xe1\xd0?\xf6\xeeC\x8a\x04\x0fG\x07\xa9\xfb\xca\xfa6\xa4\xd9\xb3\xc5\x1d\xe1r\xe3\x98\xfa4\x84\xd6\xfa2z\x0e\x8f2\xe8V\xfa6'
Received decrypted response from the server: working fine
Enter a message to send to the server: (to quit ctrl+c): █
```

RSA-JAVA CODE

```
import java.io.DataInputStream;

import java.io.IOException;
import java.math.BigInteger;
import java.util.Random; public class RSA {

    // Initializing big intergers p,q...etc to store large integers
    private BigInteger p, q, n, phi, e, d;
    // bitlength of the above large prime numbers
    private static int bitlength = 1024;
    private Random r; // Random variable
    boolean result;
    public RSA() {
        r = new Random();

        // Generating a large random probable prime number p and verifying itp = BigInteger.probablePrime(bitlength, r);
        int length_p = String.valueOf(p).length();
        //System.out.println("The value of prime number p is: " + p);result = p.isProbablePrime(1);
        if (result == true) {
            System.out.println("The big interger p is a probable prime number");
        } else {
            System.out.println("The big interger p is not a primenumber...please execute again");}
        System.out.println("The length of p is - " + length_p);
```

```
// Generating a large random probable prime number q and verifying it
q = BigInteger.probablePrime(bitlength, r);
int length_q = String.valueOf(q).length();
//System.out.println("The value of prime number q is : " + q);
result = q.isProbablePrime(1);
if (result == true) {
    System.out.println("\nThe big interger q is a probable primenumber");
} else {
    System.out.println("The big interger q is not a primenumber...please execute again");}
System.out.println("The length of q is - " + length_q);
```

```
// Multiplying p and q to obtain one part of public key 'n' of thealgorithm
n = p.multiply(q);
int length_n = String.valueOf(n).length();
//System.out.println("The value of prime number n is: " + n);
System.out.println("\nThe length of n is - " + length_n);
// Totient function which consist set of integers that are relative to 'n'
phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
// an integer such thats is co-prime with phi
e = BigInteger.probablePrime(bitlength / 2, r);
while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0)
{e.add(BigInteger.ONE);
}
d = e.modInverse(phi);
}
public RSA(BigInteger e, BigInteger d, BigInteger n)
{this.e = e;this.d = d;this.n = n;
}
```

```
@SuppressWarnings("deprecation")

public static void main(String[] args) throws IOException {

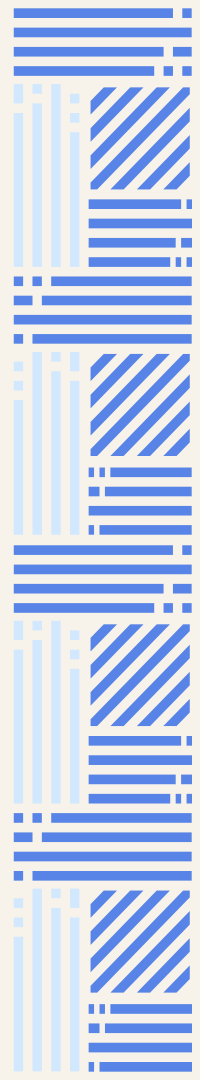
    long start = System.currentTimeMillis();


    DataInputStream in = new DataInputStream(System.in);
    String teststring;
    System.out.println("----- Enter the text to be encrypted -----
    ");
    teststring = in.readLine();

    System.out.println("\n-----Generating very large prime numbers of
    given bitlength----- ");
    RSA rsa = new RSA();
    //System.out.println("String in Bytes:" + bytesToString(teststring.getBytes()));

    // encrypt
    System.out.println("\n----- Encrypting text ");
    byte[] encrypted = rsa.encrypt(teststring.getBytes());
    //System.out.println("\nEncrypted Bytes: " + bytesToString(encrypted));
    System.out.println("Encrypted String: " + new String(encrypted));

    // decrypt
    System.out.println("\n----- Decrypting text ");
    byte[] decrypted = rsa.decrypt(encrypted);
    //System.out.println("\nDecrypted Bytes: " + bytesToString(decrypted));
    System.out.println("Decrypted String: " + new String(decrypted));
```





```
if (teststring.equals(new String(decrypted)) == true) {
    System.out.println("\nx----- RSA Algorithm is successful --
    x");
}
```

```
long end = System.currentTimeMillis();
System.out.println("\nThe run time for bitlength " + bitlength + " is " + (end - start) / 1000F + "
seconds");
}
}
```

```
@SuppressWarnings("unused")
private static String bytesToString(byte[] encrypted) {String test = "";
    for (byte b : encrypted) {test += Byte.toString(b);
    }
    return test;
}

// Encrypt message
public byte[] encrypt(byte[] message) {
    return (new BigInteger(message)).modPow(e, n).toByteArray();
}

// Decrypt message
public byte[] decrypt(byte[] message) {
    return (new BigInteger(message)).modPow(d, n).toByteArray();
}
}
```

```

----- Enter the text to be encrypted -----
further increasing the bitlength value!...
|
----- Generating very large prime numbers of bitlength 6144 -----
The big integer p is a probable prime number
The length of p is - 1850

The big integer q is a probable prime number
The length of q is - 1850

The length of n is - 3699

----- Encrypting text -----
Encrypted String: NfBdA0Icii-00i;D0%y'< 0rA$AgëBäVXp05B"-@?•ëqk&`Äü"²Z"m>[bB, B0püBdi,x206[Hc]It"m»[!°,Éä~"k³Rµ,,
ëicBg0uñkQ4»io-ïueðIÊ«E«W'Q'&0~?,,äXxajñB3 SX9"÷" [Y<4IiµS4*5a@soBXCm^İFžp2s|)æ?PBBB;6idëb?ä°pX^..E"Vİ³~vB
B/"Eñ20"00A=f=4İüAx < 2~%Bžçø#ÿ/,€*tB?"0*U|ÄxBBBÜYAa0iñyZis"ku46İ"BBÄE0BçEpäiçB" ÿ-=-ð,d\&H".æB0°İ"RR!ByY<ð%0ZTÄ±:|ø±İBİ8İ"f0äB"óu"0B,,uH0ZYB|,"É0ÄBž±N<,B±kµU%
,
ð04pµEß. • B/ëUðVE"?"
İB?ý,
İ[7"i<TÜ0xÄ)ÉYB0%<"Tßäü²"çÜi.Ö<.:°-Bßëh±"kß>uYH89(xLÜ/B."B-±lc|.B<İB",X3>BÖQ<žN-]BÜäÄ[;äöi@ræöupQç-İñ?-N±Z=g4•-æä{.Ü"øÄ"ß?BžBÿ' Ü-~²f[
3)ÖYr19³µv1B" æöiÜ)öİ"K'çj"æ%260$;BÖİY-Bİñ$*xß)äİ,4
÷äroe09DZ|³æYrðÖÜüðm³><BÜž)qN$Y±..SÜëçC*0BßçE#İZ*.etY'±€]dİ+vÖİüJ-İw23B-İ>$
xH2?...f"?ÿyðBÄSëüiä²?ÿpç$B.ž0%±0Bß±..#BÄİ$|<ÜBÜ"4°<|Vİödi-BRÄçæ;iz4$İBÜB0 jgim"1W-Ä>òt"ZÉ/*Ö0B³BëBxU?5;ëf"Z²%İSä)YæBİµö"äBßEñ"N"çëY.'-¥T1<?Aò(QG±CI"L'Ä-ç0X0".
----- Decrypting text -----
Decrypted String: further increasing the bitlength value!...

x----- RSA Algorithm is successful -----x

The run time for bitlength 6144 is 259.863 seconds

```



-Thank you-