RSA Algorithm by multiplying large Prime Numbers

A project report

Submitted by

NAME	ROLL.NO
AKSHAYAA	CB.EN.U4AIE21002
Gajula Sri vatsanka	CB.EN.U4AIE21010
MDSR Saran	CB.EN.U4AIE21034
R. Sai Raghavendra	CB.EN.U4AIE21049

21AIE431

Centre for computational engineering and networking

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE-641112(INDIA)

5/11/23

ACKNOWLEDGMENT

We are deeply thankful to **Centre for Excellence in Computational Engineering and Networking** (CEN) at **Amrita Vishwa Vidyapeetham**, **Coimbatore** for providing us such a wonderful environment to pursue our research. We would like to express our sincere gratitude to **Mr SUNIL KUMAR Professor, Department of Centre for Excellence in CEN**, Amrita Vishwa Vidyapeetham. We have completed our project under her guidance. We foundthe project area, topic, and problem with his suggestions. We would also like to acknowledge our team members for supporting each other and be grateful to our university for providing this opportunity for us. Lastly special thanks to Centre for Excellence in CEN for providing this opportunity to present a project in this field.

ABSTRACT

In the digital age, cryptography is an essential technology that guarantees private and secure communication. A key element of contemporary cryptography is the RSA (Rivest-Shamir-Adleman) encryption algorithm. The application of RSA encryption to create secure communication between a server and several clients is investigated in this project. Three parts make up the project: a shared encryption technique, a client, and a server. To ensure data security and integrity, the server and clients exchange public keys, construct RSA key pairs, and encrypt and decrypt messages. We carefully work our way through the algorithm's fundamental elements, from the creation of big prime numbers to the identification of related public and private key pairs. We then show how to encrypt plaintext data using these keys, guaranteeing its confidentiality and integrity throughout transmission. Then we finally decrypt the encrypted text to verify it with the original text. The RSA approach is presented in this project as a python and java implementations.

INTRODUCTION

The Cryptography Project presented here revolves around the implementation of the RSA encryption algorithm for secure communication between a server and multiple clients. This project has been designed with a three-fold objective: the server, clients, and the shared encryption scheme.

ServerRSA and Server Program:

- ServerRSA is responsible for RSA key generation on the server side. It initializes an RSA key pair, exporting the public key for distribution to clients.
- The Server Program binds to a network socket, listens for incoming connections, and accepts client requests. It sends the server's public key to clients, and upon receiving encrypted data, decrypts it using the server's private key. This component showcases the server's role in key management and secure communication.

Client Program:

- The Client Program establishes a connection to the server, generates its own RSA key pair, and imports the server's public key. The client can then securely exchange messages with the server by encrypting data with the server's public key and decrypting data from the server using its private key. This component emphasizes the client's participation in the secure communication process.

Shared Encryption Scheme:

- The heart of the project is the shared encryption scheme, which involves the use of the RSA algorithm. The server and clients share public keys and use them to encrypt and decrypt messages. The RSA algorithm provides end-to-end encryption, ensuring data confidentiality and integrity.

OBJECTIVE

This project's main goal is to show and implement RSA encryption for safe communication between a server and several clients. It also has the following particular objectives:

- Key creation: To create safe public and private key pairs, use RSA key creation on the client and server sides.

- Key Exchange: To enable safe communication, allow the server and clients to exchange public keys in a secure manner.

- Encrypt communications with the server's public key so that clients can do the same to protect the privacy of their data.

- Message Decryption: To retrieve and process client messages, implement message decryption on the server using its private key.

- Establish a working system that demonstrates end-to-end encryption to enable private and secure communication between the server and clients.

HARDWARE AND SOFTWARE REQUIRMENTS

For this project, which entails implementing RSA encryption for secure communication between a server and clients, the following hardware and software requirements must be met:

Hardware specifications:

- Computer(s): One or more extra computers can serve as clients, whereas at least one computer is needed to function as the server. These PCs ought to be able to run the necessary applications with a minimal system demand.

- Network Connectivity: A network connection is required between the clients and the server. This could be the internet or a local area network (LAN), based on the size of the project.

Software specifications:

- Operating System: Windows, Linux, and macOS are just a few of the operating systems on which the applications below can be used. Make sure the operating system you have chosen works with the necessary software.

- Python: Python is used to implement this project. On all of the project's machines (clients and server), a Python interpreter must be installed. Python is available for download at the official website, https://www.python.org/.

- Python Libraries: Use pip to install the necessary Python libraries on the client and server computers. The following commands will enable you to install them:

- pip install pycryptodome: To implement RSA encryption, use this library's cryptographic functions.To find likely prime numbers, use the pip install sympy package.

- Integrated Development Environment (IDE) or text editor: In order to write, edit, and execute Python code, you'll need an IDE or text editor. IDLE (included with Python), PyCharm, and Visual Studio Code are popular options.

- Networking Software: Verify if your machines are permitted to communicate over a network. Security software and firewalls should be set up to allow communication between the server and clients.

- Terminal or Command Prompt: To run Python scripts, you'll need a terminal or command prompt.

IMPLEMENTATION

There are several phases involved in implementing the project, which uses RSA encryption to provide secure communication between a server and clients. A detailed implementation guide for the project can be found below:

Step 1: Configure Your Environment for Development

Install Python: Go to the official website (https://www.python.org/) to download and install Python if it isn't already installed.

Install Required Libraries: On all computers involved (clients and server), install the necessary Python libraries using pip. Execute the following commands after opening a terminal or command prompt:

Install PyCrytodome with pip

Install Sympy using pip

Step 2: Construct the Server Part

Write the Server Code: For the server component, write a Python script that implements the ServerRSA class, the Server programme, and the key management logic.

The server script may be written and saved using an Integrated Development Environment (IDE) or text editor.

Step Three: Assemble the Customer Part

Compose the client programme:

For the client component, which consists of the message exchange logic, key generation, and client programme, write a Python script.

If you intend to work with more than one client, write a different script for each.

Stage 4: Organise Network Contact

Establishing Network Communication

Make that the server can be accessed via the internet (for a remote setup) or that all computers are linked to the same network (for a local setup).

Set up the network configuration:

Clients must know the server's IP address or hostname in order to connect to it, therefore take note of this information.

Step 5: Evaluate the Application

Execute the Server:

On the machine assigned to serve, launch the server script. The server ought to be operational and ready to receive new connections.

Manage the Clientele:

On one or more client PCs, execute the client script. The public key of the server will be sent to clients upon their connection to it.

Secure Transmission:

Now, communications encrypted by clients can be sent to the server, which will decrypt and handle them.

THEORY

Why is the RSA algorithm used?

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 -- or *factoring* -- is considered infeasible due to the time it would take using even today's supercomputers.

Modular arithmetic

When we divide two integers we will have an equation
A/B = Q remainder RA is the dividend , B is the divisor, Q is the quotient and R is the remainder.
A number *x* mod *N* is the equivalent of asking for the remainder of *x* when divided by *N*. Two integers *a* and *b* are said to be congruent (or in the same equivalence class) modulo *N* if they have the same remainder upon division by *N*. In such a case, we say that $a \equiv b (\mathrm{mod} N)$.

Euler’s phi-function

Euler’s phi-function is a function defined on the set of positive integers: ϕ: $\mathbb{Z}+ \to \mathbb{Z}+$, by

$\phi(n)$ = the number of integers in the set $\{1,2,\ldots,n\}$ that are relatively prime to n. If n = $p1$
$\alpha 1 p2\ \alpha 2 \cdots pk\ \alpha k$

where $p1$, $p2$, $\cdots$, pk are distinct primes and $\alpha 1$, $\alpha 2$, $\cdots$, αk ∈□+ then $\phi(n)$ = $(p1 - 1)p1$
$\alpha 1{-}1$ $(p2 - 1)p2$ $\alpha 2{-}1$ $\cdots$ $(pk - 1)pk$ $\alpha k{-}1$

RSA ALGORITHM

The Algorithm contains the following methods.

Encryption:

$$c = p^e \, mod \, n$$

Decryption:

$$p = c^d \, mod \, n$$

Where

p = plain text

c = cipher key

message and the ciphertext are integers in the range 0 to n −1.

Thus, the encryption key is a pair of positive integers (e, n). Similarly, the decryption key is a positive integer pair (d, n). Each user makes his encryption key public while keeping his decryption key private.

Public key: {e , n}

Private key: {d , n}

Key generation:

Follow the below steps

1) Consider 2 large prime numbers namely p , q.
2) Calculate n = p*q.
3) Euler's Totient Function: $\phi(n) = (P-1)(q-1)$.
4) Choose a small number e ,co-prime to $\phi(n)$ with GCD ($\phi(n)$,e) = 1 and 1<e< $\phi(n)$.
5) Find d such that d x e mod $\phi(n)$ = 1.

Example:

implementing the above algorithm using 2 small prime numbers.

Key generation:

1) P =3, q =5
2) n = p*q = 3*5 =15
3) $\phi(n) = (P-1)(q-1)$

= (3-1)(5-1)

= 2*4

$\boldsymbol{\phi(n)}$= ***8***

4) Assume e such that it is a co-prime to $\phi(n)$ with GCD ($\phi(n)$,e) = 1
Gcd(3,8) = 1
Gcd(5,8) = 1
Gcd(7,8) = 1

e = 3

5) d * e mod $\phi(n)$ = 1.
d*3 mod8 = 1
assume *d=3*
3*3mod8 = 1
9mod8 =1
1=1
Therefore ***d =3***

By implementing the above steps we get:

Public key: {3 , 15}

Private key: {3 , 15}

Encryption:

Consider plain text p = 8

$$c = 8^3\, mod\, 15$$

$$= 512\, mod\, 15$$

$$\boldsymbol{c = 2}$$

Decryption:

$$p = 2^3\, mod\, 15$$

$$= 8\, mod\, 15$$

$$\boldsymbol{p = 8}$$

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()
```

Output

Server code

```
import socket
from Crypto.PublicKey import RSA as CryptoRSA
from Crypto.Cipher import PKCS1_OAEP
from base64 import b64encode

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.export_key()
        self.public_key = rsa_key.publickey().export_key()

def server_program():
    host = socket.gethostname()
    port = 5004
    server_socket = socket.socket()
    server_socket.bind((host, port))
    server_socket.listen(2)
    conn, address = server_socket.accept()
    print("Connection from: " + str(address))

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

    while True:
        data = conn.recv(1024)

        if not data:
            break

        rsa_key = CryptoRSA.import_key(server_rsa.private_key)
        cipher = PKCS1_OAEP.new(rsa_key)
        plaintext = cipher.decrypt(data)

        print("Received and Decrypted message from connected user: " + plaintext.decode())

        # Process the data (if needed)

        # Encrypt and send a response (if needed)
        response = "Server response: Thanks for your message!"
        cipher = PKCS1_OAEP.new(rsa_key.publickey())
```

```
        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
from Crypto.Cipher import PKCS1_OAEP
from base64 import b64encode

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

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

    rsa_key = RSA.generate(1024)  # Generate a new key pair for the client

    server_public_key = client_socket.recv(4096)
    server_rsa_key = RSA.import_key(server_public_key)

    while True:
        message = input("Enter a message to send to the server: ")

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

        data = client_socket.recv(4096)
        # The data received from the server is already encrypted, so no need to decrypt it here.
        print("Received encrypted response from the server: " + b64encode(data).decode())

    client_socket.close()

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

RESULT

```
Enter the text to be encrypted: rsa algorithm
------------ Generating very large prime numbers of given bitlength --------------
The value of prime number p is: 650886256109652460482837906572784924011850676831457
The big integer p is a probable prime number
The length of p is - 308
The value of prime number q is: 962163868864356536866633498906983271514617317619656
The big integer q is a probable prime number
The length of q is - 307
The value of prime number n is: 626259238369099633738912338234690955736854580267221
The length of n is - 615

------------ Encrypting text --------------
Encrypted String: A5sgRjyRSgXRCDfCrI2a9jbnUvb/wCiKeZdP+Pvkjoq5KQGGrz9VhZ6EWWO63lJfV5

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

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

```
... Connection from: ('192.168.81.1', 54513)
    Received and Decrypted message from connected user: hello
    Received and Decrypted message from connected user: world
    Received and Decrypted message from connected user: hi
    Received and Decrypted message from connected user: heman
```

```
... Received encrypted response from the server: kot2SDaSMUGM7ha401OkeEP0UnMt5rp1KI+VYvo3vc
    Received encrypted response from the server: acFfyHxymuLNG6E5mkxiQCmePxcMFs3xuKBMs2UTX2
    Received encrypted response from the server: ShIlEhmmpkvUpFYEOycA5ZNK+7PYihwTZU9nPzSCx+
    Received encrypted response from the server: OIGT4wcqQJ0vaH17/bWfCpmrosHz2bGznAuQXY7xaE
    Received encrypted response from the server: PY+uRubaZRTxp700ePkU84N81gCtCpFkQAjK75IKZL
```

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 it
 p = 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 prime
number...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 prime
number");
 } else {
 System.out.println("The big interger q is not a prime
number...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 the
algorithm
 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();
 }
}
```

RESULT

```
----------- Enter the text to be encrypted -----------------
further increasing the bitlength value!....

------------ Generating very large prime numbers of bitlength 6144 --------------
The big interger p is a probable prime number
The length of p is - 1850

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

The length of n is - 3699

------------ Encrypting text --------------
Encrypted String: [illegible]

------------ 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
```

POSSIBLE FUTURE EXTENSIONS

Expanding the project's scope beyond its original parameters can improve its functionality and increase its adaptability. The project can see the following significant future extensions:

Authentication of Users:

Put user authentication procedures in place to guarantee that the server can only be accessed by authorised users. This could entail more sophisticated techniques like two-factor authentication (2FA) or login and password-based authentication.

Checks for Message Integrity:

Message integrity checks can be used to improve communication security. Create digital signatures for messages using cryptographic hash functions so that the recipient may confirm the legitimacy and integrity of the messages they have received.

Several Clients at Once:

Change the server so that it can manage several clients at once. To efficiently handle and service multiple client connections, this may require implementing multithreading or asynchronous input/output.

Group Chat and Forwarding of Messages:

Extend the project to facilitate group conversation by letting clients message one another. To enable this feature, put in place a message forwarding mechanism on the server.

Interface Graphical (GUI):

Provide a graphical user interface that is easy to use for both the server and clients. Users unfamiliar with command-line interfaces may find the project easier to understand and more user-friendly with a graphical user interface (GUI).

****THANK YOU****