Experiment 04

Aim: Analyze and implement RSA Algorithm
Tools: C/C++/Java/Python
Theory: **RSA Algorithm**

RSA (Rivest-Shamir-Adleman) is a widely used public key cryptosystem that enables secure communication and digital signatures. It was introduced in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman, and remains one of the most widely used asymmetric encryption algorithms.

Key Generation:

- Choose two large prime numbers, p and q.
- Compute their product, n = p * q. The security of RSA relies on the difficulty of factoring the product of two large prime numbers.
- Calculate Euler's totient function, φ(n) = (p-1)(q-1).
- Select an integer e, known as the public exponent, such that 1 < e < φ(n), and e is coprime with φ(n). Common choices for e are 3 or 65537 due to their efficiency.
- Compute the private exponent d, which is the modular multiplicative inverse of e modulo φ(n). In other words, d * e ≡ 1 (mod φ(n)).
- The public key is (n, e), and the private key is (n, d).

Encryption:

- Represent the plaintext message as an integer m, where 0 < m < n.
- Compute the ciphertext c ≡ m^e (mod n).

Decryption:

- Receive the ciphertext c.
- Compute the plaintext message m ≡ c^d (mod n).

Application of RSA:

- Secure Communication: RSA is used in SSL/TLS for key exchange and digital signatures, ensuring secure Internet communication.
- Digital Signatures: RSA creates digital signatures, verifying message authenticity and integrity, crucial for secure transactions and document validation.
- Secure Email: RSA encrypts and signs emails in PGP and S/MIME, enhancing confidentiality and confirming sender legitimacy.
- File Transfer: RSA secures file transfers by encrypting data, safeguarding sensitive information during transmission.
- Online Transactions: RSA protects online transactions, encrypting details like credit cards, fostering secure e-commerce and banking.
- VPN Security: RSA is employed in VPNs for key exchange and authentication, establishing secure connections over the Internet.

Implementation:Code-

Server side

```
import socket
import random

def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

def generate_keypair():
    p, q = 0, 0
    while not is_prime(p):
        p = random.randint(100, 1000)
    while not is_prime(q) or p == q:
        q = random.randint(100, 1000)
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537  # Commonly used value for e
    d = mod_inverse(e, phi)
    return (n, e), (n, d)

def mod_pow(base, exponent, modulus):
    result = 1
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        base = (base * base) % modulus
        exponent //= 2
    return result

def decrypt(encrypted, private_key):
    n, d = private_key
    decrypted = [chr(mod_pow(char, d, n)) for char in encrypted]
    return ''.join(decrypted)

def main():
    host = '127.0.0.1'
    port = 12345

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen()

    print("Waiting for client to connect...")
    client_socket, addr = server_socket.accept()
    print("Client connected!")

    public_key, private_key = generate_keypair()
    print("Public key generated:", public_key)
    print("Private key generated:", private_key)

    # Send public key to the client
    client_socket.sendall(str(public_key).encode())

    # Receive public key from client
    public_key_str = client_socket.recv(1024).decode()
    public_key = eval(public_key_str)
    print("Received public key from client:", public_key)

    encrypted_message = client_socket.recv(1024).decode()
    print("Received encrypted message from client:", encrypted_message)

    decrypted_message = decrypt(eval(encrypted_message), private_key)
    print("Decrypted message:", decrypted_message)
    server_socket.close()

if _name_ == "_main_":
    main()
```

Client Side:

```
import socket
import random

def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

def generate_keypair():
    p, q = 0, 0
    while not is_prime(p):
        p = random.randint(100, 1000)
    while not is_prime(q) or p == q:
        q = random.randint(100, 1000)
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537  # Commonly used value for e
    d = mod_inverse(e, phi)
    return (n, e), (n, d)

def mod_pow(base, exponent, modulus):
    result = 1
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        base = (base * base) % modulus
        exponent //= 2
    return result

def encrypt(message, public_key):
    n, e = public_key
    encrypted = [mod_pow(ord(char), e, n)
for char in message]
    return encrypted

def main():
    host = '127.0.0.1'
    port = 12345
    public_key, private_key =
generate_keypair()
    print("Public key generated:",
public_key)
    print("Private key generated:",
private_key)

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

    # Receive public key from server
    public_key_str =
client_socket.recv(1024).decode()
    public_key = eval(public_key_str)
    print("Received public key from server:",
public_key)

client_socket.sendall(str(public_key).encod
e())

    # Get the message to send
    message_to_send = input("Enter the
message to send: ")

    # Encrypt the message
    encrypted_message =
encrypt(message_to_send, public_key)
    print("Encrypted message:",
encrypted_message)

    # Send the encrypted message to the
server

client_socket.sendall(str(encrypted_messag
e).encode())

    client_socket.close()

if _name_ == "_main_":
    main()
```

Output:

Server Side-

PS C:\Users\Anshul\downloads> python server.py
>>
Waiting for client to connect...
Client connected!
Public key generated: (274793, 65537)
Private key generated: (274793, 184433)
Received public key from client: (274793, 65537)
Received encrypted message from client: [165046, 138111, 53912, 153432, 199435, 19523, 18397, 58062, 145712, 241304, 129907, 18397, 165046, 138111, 198025, 18397, 204485, 241304, 165046, 85413, 153432, 165046, 58062, 18397, 58062, 145712, 198025, 51611]
Decrypted message: anshul more and pratham mody

Client Side-

PS C:\Users\Anshul\downloads> python client.py
Public key generated: (121903, 65537)
Private key generated: (121903, 62153)
Received public key from server: (274793, 65537)
Enter the message to send: anshul more and pratham mody
Encrypted message: [165046, 138111, 53912, 153432, 199435, 19523, 18397, 58062, 145712, 241304, 129907, 18397, 165046, 138111, 198025, 18397, 204485, 241304, 165046, 85413, 153432, 165046, 58062, 18397, 58062, 145712, 198025, 51611]

Result and Discussion:

In this experiment, we implemented the RSA Algorithm for key generation, encryption, and decryption. The decrypted message matched the original message, indicating a successful implementation.

Learning Outcomes:

1. Understand the RSA Algorithm.
2. Analyze and implement the RSA Algorithm for secure data transmission.

Conclusion: After performing this experiment, we gained insights into the RSA Algorithm and its application in secure communication.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	Total
Marks Obtained				