Experiment-6

Aim → Compare symmetric cryptographic process with respect to ~~assymet~~ asymmetric cryptographic process by implementing AES & RSA algorithm.

Objectives 1 → An overview on AES (Advanced Encryption (Standard) Algorithm).

AES is a widely used symmetric encryption algorithm that encrypts and decrypts data using the same key. It works on 128-bit data blocks and support key sizes of 128, 192, 256 bits, corresponding to 10, 12, or 14 rounds of encryption.

Each round include four main steps →

i) SubBytes - Byte substitution using an S-box
ii) ShiftRows - Shift rows in the data matrix.
iii) MixColumns - Mixes data within each column (except in the final round).
iv) AddRoundKey - Combines data with a round key.

AES is fast, secure and widely used in everything from internet encryption (SSL/TLS) to file & disk encryption.

Objective 2 → An overview on RSA algorithm.

RSA is an assymmetric encryption algorithm that uses a public key to encrypt data and a private key to decrypt it. It's based on the mathematical difficult of factoring large prime numbers.

Key Points →

i) Public key = for encryption (shared openly)
ii) Private key = for decryption (kept secret)
iii) Used in secure communication, digital signatures & key exchange
iv) Common key sizes: 2048 or 4096 bits.

RSA is slower than AES but crucial for secure key exchanges in many systems like HTTPS, after which faster algorithms (like AES) handle the actual data encryption.

Objective 3 → Execution of AES algorithm for encryption and decryption of text message.

Code →

```
from Crypto.Cipher import AES
import binascii

def encryption(aes, plaintext):
    while len(plaintext) % 16 != 0:
        plaintext += ' '
    plaintext = plaintext.encode()
    ciphertext = aes.encrypt(plaintext)
```

Name: _____ Regd No: _____

```
        return binascii.hexlify(ciphertext).decode()
def decryption(aes, ciphertext_hex):
    ciphertext = binascii.unhexlify(ciphertext_hex)
    plaintext = aes.decrypt(ciphertext)
    return plaintext.decode().strip()

if __name__ == "__main__":
try:
    key = input("Enter key (16 bytes): ").
    if len(key) != 16:
        print("key must be exactly 16 characters!")
        exit()
    plaintext = input("Enter plaintext: ")
    print("Plaintext is:", plaintext)
    key_bytes = key.encode()
    aes_cipher = AES.new(key_bytes, AES.MODE_ECB)
    ciphertext = encryption(aes_cipher, plaintext)
    print("Ciphertext after encryption:", ciphertext)
    decrypted_text = decryption(aes_cipher, ciphertext)
    print("Plaintext after decryption:", decrypted_text)
except ValueError as e:
    print(f"Error: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Output → Enter key (16 bytes): 1234567890123456
Enter plaintext: qwerty
Plaintext is: qwerty
Ciphertext after encryption: 2577bbfab2da6409d6fbaf3d0cbb3c7b
Plaintext after decryption: qwerty.

Objective 4→ Execution of RSA algorithm for encryption using public key and decryption using private key.

Code → import random
from math import gcd

```
def is_prime(n):
    if n <= 1 : return false
    for i in range(2, int(n**0.5)+1):
        if n % i == 0: return False
    return True
def generate_prime(start=100, end=300):
    while True:
        p = random.randint(start, end)
        if is_prime(p) : return p

def modinv(a,m):
    m0, x0, x1 = m, 0, 1
    while a>1:
        q = a//m
        a, m = m, a%m
        x0, x1 = x1-q*x0, x0
    return x1 % m0
def generate_key():
    p = generate_prime()
    q = generate_prime()
    while q == p:
        q = generate_prime()
    n = p*q
    phi = (p-1)*(q-1)

    e = 65537
    while gcd(e,phi) != 1:
        e = random.randrange(2, phi)
    d = modinv(e, phi)
    return (e,n) (d,n)
def encrypt(msg, public_key):
    e,n = public_key
```

```
        return [pow(ord(char), e, n) for char in msg]

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

public, private = generate_key()
message = "hello"
ciphertext = encrypt(message, public)
decrypted = decrypt(ciphertext, private)
print("Message :", message)
print("Encrypted :", ciphertext)
print("Decrypted :", decrypted)
```

Output → Message : hello
Encrypted : [30452, 5040, 32731, 32731, 37707]
Decrypted : hello

Conclusion → In this experiment, we implemented and compared AES and RSA algorithms to understand symmetric & asymmetric encryption. AES proved to be faster and suitable for bulk data, while RSA offered secure key exchange using separate keys. This helped us grasp their pratical use in real-world cryptographic systems.

Exercise →

1) Transform the plaintext "AES USES A MATRIX" into a state matrix form.

Ans → Plaintext to ASCII then HEX →

Characters → A E S US E S A MATRIX

ASCII → 65 69 8332 8583 69 83 32 65 77 65 84 82 73 88

HEX → 41 45 53 20 55 53 45 53 20 41 4D 41 54 52 49 58

State Matrix → (in column wise)

$$\begin{bmatrix} 41 & 55 & 20 & 24 \\ 45 & 53 & 41 & 52 \\ 53 & 45 & 4D & 49 \\ 20 & 53 & 41 & 58 \end{bmatrix}$$

2) Compute the output of S-box with given input state matrix as $\begin{bmatrix} 4 & 5 \\ E & 2 \end{bmatrix}$ and S-box as

	00	01	10	11
00	1001	0100	1010	1011
01	1101	0001	1000	0101
10	0110	0010	0000	0011
11	1100	1110	1111	0111

Ans → Input State → $\begin{bmatrix} 4 & 5 \\ E & 2 \end{bmatrix}$

Break each hex into 4-bit binary and locate in the S-box →

i) 4 (0100) →

Row → 01 Column → 00

S-box → [01][00] → 1101

ii) 5 (0101) →

S-box [01][01] → 0001

iii) E (1110) →

Row

S-box [11][10] → 1111

iv) 2 (0010) →

S-box [00][10] → 1010

Output → $\begin{bmatrix} 1101 & 0001 \\ 1111 & 1010 \end{bmatrix} \xrightarrow[\text{Hex}]{\text{In}} \begin{bmatrix} D & 1 \\ F & A \end{bmatrix}$

3) Perform Shift Row transformation on the given current matrix.

Ans → After Shift Rows →

$$\begin{bmatrix} 63 & EB & 9F & 40 \\ 2F & 93 & 92 & C0 \\ AF & C7 & AB & 30 \\ A2 & 20 & CB & 2B \end{bmatrix}$$

$$\begin{bmatrix} 63 & EB & 9F & 40 \\ C0 & 2F & 93 & 92 \\ AB & 30 & AF & C7 \\ 20 & CB & 2B & A2 \end{bmatrix}$$

4) Find the output of Mix Column if the input state matrix is $\begin{bmatrix} D & 1 \\ A & F \end{bmatrix}$ with the predefined matrix as $\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}$ and the irreducible polynomial for GF(16) is $x^4 + x + 1$.

Ans → Step by step (in GF(2^4), hex notation) →

Let's output matrix be $\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} (1\times D) \oplus (4\times A) & (1\times 1) \oplus (4\times F) \\ (4\times D) \oplus (1\times A) & (4\times 1) \oplus (1\times F) \end{bmatrix} = \begin{bmatrix} D \oplus 2 & 1 \oplus 3 \\ 7 \oplus A & 4 \oplus F \end{bmatrix}$

$$= \begin{bmatrix} B & 2 \\ D & B \end{bmatrix}$$

5) Given the following values for the RSA algorithm:

- Two prime numbers: $p = 61, q = 53$
- Public key: $e = 17$

Calculate the private key d, where d is the modular inverse of e modulo $\phi(n)$.

Ans →

Name: _____ Regd No: _____

Ans-> Given, $p = 61$, $q = 53$, $e = 17$

$$n = p \times q = 61 \times 53 = 3233$$

$$\varphi(n) = (p-1)(q-1) = 60 \times 52 = 3120$$

$$d \times e \equiv 1 \bmod 3120$$

$$\Rightarrow d \equiv e^{-1} \bmod 3120$$

Using Extended Euclidean Algorithm to find the modular inverse of 17 mod 3120.

$$3120 = 183 \times 17 + 9$$
$$17 = 1 \times 9 + 8$$
$$9 = 1 \times 8 + 1$$
$$8 = 8 \times 1 + 0$$

Now back - substitute →

$$1 = 9 - 1 \times 8$$
$$8 = 17 - 1 \times 9$$
$$9 = 3120 - 183 \times 17$$

Substitute all :

$$1 = 9 - 1(17 - 1 \times 9) = 2 \times 9 - 1 \times 17 = 2(3120 - 183 \times 17) - 1 \times 17 = 2 \times 3120 - 367 \times 17$$

$$\Rightarrow 1 = 2 \times 3120 - 367 \times 17 \Rightarrow -367 \times 17 + 2 \times 3120 = 1$$

$$\text{So, } d \equiv -367 \bmod 3120 \Rightarrow d = 2753$$

$\therefore$ Private key $d = 2753$

Name: _____ Regd No: _____