19CSE331 CRYPTOGRAPHY

PRACTICAL ASSIGNMENT REPORT

SECURE CLOUD STORAGE

(GROUP 8)

Naveen	AM.EN.U4CSE22002
Aswathy Anand	AM.EN.U4CSE22011
Ch Geethika Gayatri	AM.EN.U4CSE22015
P Rithika	AM.EN.U4CSE22040

I. INTRODUCTION

Brief Overview:

The Secure Cloud Storage project ensures the secure transfer and storage of sensitive data in cloud environments. It uses 3DES for encrypting data, RSA for secure key exchange and digital signatures, and SHA-256 for verifying data integrity. Symmetric encryption with 3DES protects data confidentiality, while RSA secures key management and enables authentication. Digital signatures ensure sender authenticity, and SHA-256 prevents tampering. Together, these methods provide a robust and reliable solution for cloud security.

Objective:

The objective of the Secure Cloud Storage project is to design a reliable system that ensures the confidentiality, integrity, and authenticity of data stored and transmitted in a cloud environment. By integrating robust cryptographic techniques such as 3DES, RSA, SHA-256, and digital signatures, the project aims to protect sensitive data from unauthorized access, tampering, and forgery while ensuring secure key management and verification of data authenticity.

II. IMPLEMENTATION

Tools and Software Used:

- **Techniques:** 3DES, RSA, HASHING(SHA256), DIGITAL SIGNATURE(RSA)

- **Python**: Used to implement cryptographic algorithms and simulate secure cloud storage.
- **Libraries**: Includes hashlib for SHA-256 hashing.
- **Text Editor:** Tools like VS Code or PyCharm for coding and debugging.

Description of the Steps Taken:

RSA Key Generation:

Keys are generated in RSA to securely encrypt data and verify identities through digital signatures. A public key is used for encryption or signature verification, while a private key is kept secret for decryption or signing.

```python
def mod_inverse(a, m):
    """Find the modular inverse of a under modulus m."""
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

# RSA Key Generation Function
def generate_rsa_keys():
    """Generates RSA public and private keys."""
    print("\nRSA Key Generation")
    p = int(input("Enter a prime number (p): "))
    q = int(input("Enter another prime number (q): "))

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

    e = int(input("Enter a public key exponent (e): "))
    d = mod_inverse(e, phi)

    if d is None:
        print("Error: Invalid e. Modular inverse doesn't exist.")
        return None, None

    print(f"Keys generated successfully!\nPublic Key: (e={e}, n={n})\nPrivate Key: (d={d}, n={n})")
    return (e, n), (d, n)
```

Key Features:

- RSA keys are generated based on user-provided prime numbers, ensuring strong encryption.
- The keys are dynamically created during runtime, enabling secure and customized usage.
- Public and private keys are formatted for easy integration with encryption and digital signature functions.
- The modular inverse computation ensures the private key's correctness and functionality.

3DES Message Encryption and Decryption:

Encryption:

```python
# 2. Triple DES Encryption
def generate_round_keys(key, rounds=16):
    return [(key >> (i % 8)) & 0xFF for i in range(rounds)]

def feistel_function(right_half, round_key):
    return ((right_half ^ round_key) << 1) & 0xFF | ((right_half ^ round_key) >> 7)

def des_encrypt_block(block, key):
    """Simplified Feistel DES with permutations."""
    if len(block) != 8:
        raise ValueError("Block size must be 8 bytes")
    rounds = 16
    round_keys = generate_round_keys(key, rounds)
    left = int.from_bytes(block[:4], "big")
    right = int.from_bytes(block[4:], "big")

    for rk in round_keys:
        temp = right
        right = left ^ feistel_function(right, rk)
        left = temp

    return (right.to_bytes(4, "big") + left.to_bytes(4, "big"))

def triple_des_encrypt(data, key1, key2, key3):
    encrypted = b""
    for i in range(0, len(data), 8):
        block = data[i:i+8].ljust(8, b"\0")
        block = des_encrypt_block(block, key1)
        block = des_encrypt_block(block, key2)
        block = des_encrypt_block(block, key3)
        encrypted += block
    return encrypted
```

Decryption:

```
def triple_des_decrypt(data, key1, key2, key3):
    decrypted = b""
    for i in range(0, len(data), 8):
        block = data[i:i+8]
        block = des_encrypt_block(block, key3)
        block = des_encrypt_block(block, key2)
        block = des_encrypt_block(block, key1)
        decrypted += block
    return decrypted.rstrip(b"\0")
```

Key Features:

- **Triple DES Security:** Encrypts data using three keys (key1, key2, key3) to strengthen encryption.
- **Block-Based Processing:** Operates on 8-byte data blocks for encryption and decryption.
- **Key Scheduling:** Generates 16 round keys for each DES encryption round.
- **Reversible Decryption:** Decrypts by reversing the encryption process with keys applied in reverse order.

RSA Signing and Verification with Hashing:

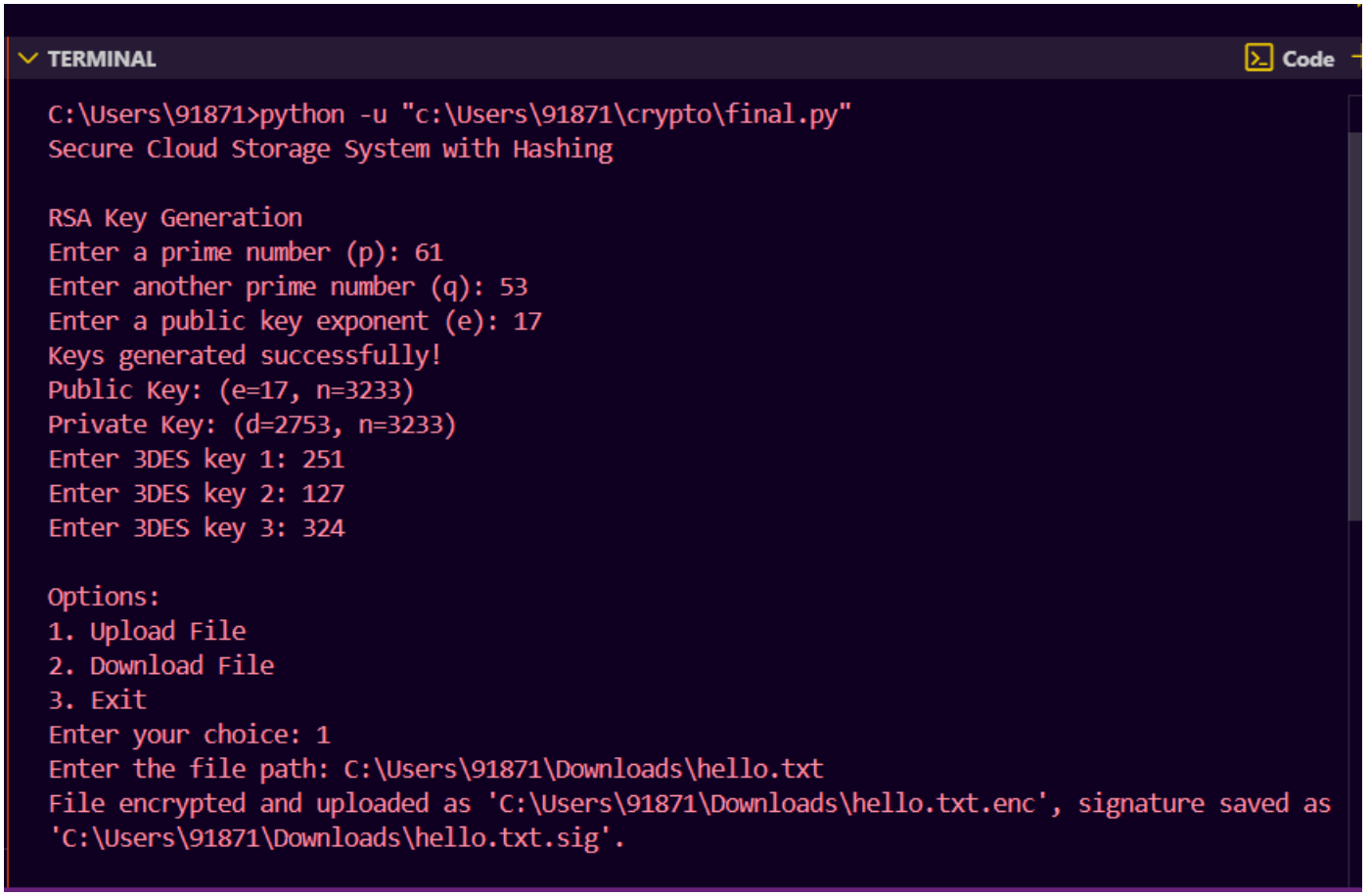
```python
# 3. RSA Signing and Verification with Hashing
def rsa_encrypt(data, key, n):
    return [pow(byte, key, n) for byte in data]

def rsa_decrypt(data, key, n):
    return [pow(byte, key, n) for byte in data]

def sha256_hash(data):
    """Generates SHA-256 hash of the data."""
    return hashlib.sha256(data).digest()

def sign_data(data, private_key):
    """Sign the hash of the data."""
    data_hash = sha256_hash(data)
    signature = rsa_encrypt(data_hash, *private_key)
    return signature

def verify_signature(data, signature, public_key):
    """Verify the RSA signature with hashing."""
    data_hash = sha256_hash(data)
    decrypted_hash = rsa_decrypt(signature, *public_key)
    return data_hash == decrypted_hash
```

Key Features:

- **SHA-256 Hashing:** Ensures data integrity by generating a secure hash.
- **Digital Signature:** Uses RSA private key to sign data hashes for authentication.
- **Signature Verification**: Confirms authenticity by comparing decrypted and recomputed hashes.

SCREENSHOTS AND DIAGRAMS:

- **ENCRYPTION:**



- **DECRYPTION:**

```
Options:
1. Upload File
2. Download File
3. Exit
Enter your choice: 2
Enter the encrypted file path: C:\Users\91871\Downloads\hello.txt.enc
Enter the signature file path: C:\Users\91871\Downloads\hello.txt.sig
Signature verification failed!
Decrypted file saved as 'C:\Users\91871\Downloads\hello.txt.dec'.
```

III. RESULTS:

The "Secure Cloud Storage" system successfully ensures **data confidentiality** (3DES encryption), **authentication and non-repudiation** (RSA digital signatures), and **data integrity** (SHA-256 hashing). It incorporates **digital signatures** for verifying the sender's identity and ensuring non-repudiation. The system simulates secure file upload and

download processes, combining encryption, signing, and verification for robust protection.

Observations:

The "Secure Cloud Storage" system ensures strong security with 3DES encryption, SHA-256 hashing, and RSA digital signatures, providing data confidentiality, integrity, and authentication. However, it faces performance issues due to 3DES and RSA, lacks robust error handling, and uses a simplified DES implementation with basic padding. While suitable for educational purposes, adopting modern standards like AES and ECDSA and enhancing error handling would make it more practical for real-world applications.

IV. CONCLUSION

The "Secure Cloud Storage" system effectively combines encryption, hashing, and digital signatures to ensure data security, integrity, and authentication, making it an excellent educational tool for understanding cryptographic principles. However, to be suitable for real-world applications, it requires improvements in performance, error handling, and the adoption of modern encryption standards