Title : Programming Symmetric and Asymmetric Crypto (Lab Task-4)

Sajib Kumar Chowdhury
Reg: 2019831049

1. Introduction:

This project offers hands-on experience with both symmetric and asymmetric cryptography using Python. It involves implementing AES encryption and decryption in ECB and CFB modes with 128-bit and 256-bit keys, as well as RSA encryption and decryption, digital signatures, and SHA-256 hashing. The program includes a command-line interface akin to OpenSSL and has functionality to measure the execution times for these operations. This report covers the implementation, usage, and performance analysis of the cryptographic functions.

2. Example Code Snippets:

AES Encryption:

```
def aes_encrypt(mode, key_length, input_text, output_file):
    keys = load_aes_keys()
    key = keys[str(key_length)]
    data = input_text.encode()

    if mode == "ECB":
        cipher = AES.new(key, AES.MODE_ECB)
        ciphertext = cipher.encrypt(data.ljust(16 * ((len(data) + 15) // 16)))
    elif mode == "CFB":
        cipher = AES.new(key, AES.MODE_CFB)
        ciphertext = cipher.encrypt(data)

    with open(output_file, 'wb') as f:
        f.write(ciphertext)
    print(f"Encrypted text written to {output_file}.")
```

AES Decryption:

```
def aes_decrypt(mode, key_length, input_file):
    keys = load_aes_keys()
    key = keys[str(key_length)]

    with open(input_file, 'rb') as f:
        ciphertext = f.read()

    if mode == "ECB":
        cipher = AES.new(key, AES.MODE_ECB)
        data = cipher.decrypt(ciphertext)
    elif mode == "CFB":
        cipher = AES.new(key, AES.MODE_CFB)
        data = cipher.decrypt(ciphertext)

    print(f"Decrypted text: {data.strip().decode()}")
```

RSA Encryption:

```
def rsa_encrypt(input_text, output_file):
    private_key, public_key = load_rsa_keys()
    data = input_text.encode()

    cipher = PKCS1_OAEP.new(public_key)
    ciphertext = cipher.encrypt(data)

    with open(output_file, 'wb') as f:
        f.write(ciphertext)
    print(f"Encrypted text written to {output_file}.")
```

RSA Decryption:

```
def rsa_decrypt(input_file):
    private_key, public_key = load_rsa_keys()

    with open(input_file, 'rb') as f:
        ciphertext = f.read()

    cipher = PKCS1_OAEP.new(private_key)
    data = cipher.decrypt(ciphertext)

    print(f"Decrypted text: {data.decode()}")
```

RSA Signing:

```
def rsa_sign(input_text, signature_file):
    private_key, public_key = load_rsa_keys()
    data = input_text.encode()

    h = SHA256.new(data)
    signature = pkcs1_15.new(private_key).sign(h)

    with open(signature_file, 'wb') as f:
        f.write(signature)
    print(f"Signature written to {signature_file}.")
```

RSA Sign Verification:

```
def rsa_verify(input_text, signature_file):
    private_key, public_key = load_rsa_keys()
    data = input_text.encode()

    with open(signature_file, 'rb') as f:
        signature = f.read()

    h = SHA256.new(data)
    try:
        pkcs1_15.new(public_key).verify(h, signature)
        print("RSA signature verification successful.")
    except (ValueError, TypeError):
        print("RSA signature verification failed.")
```

SHA-256 Hashing:

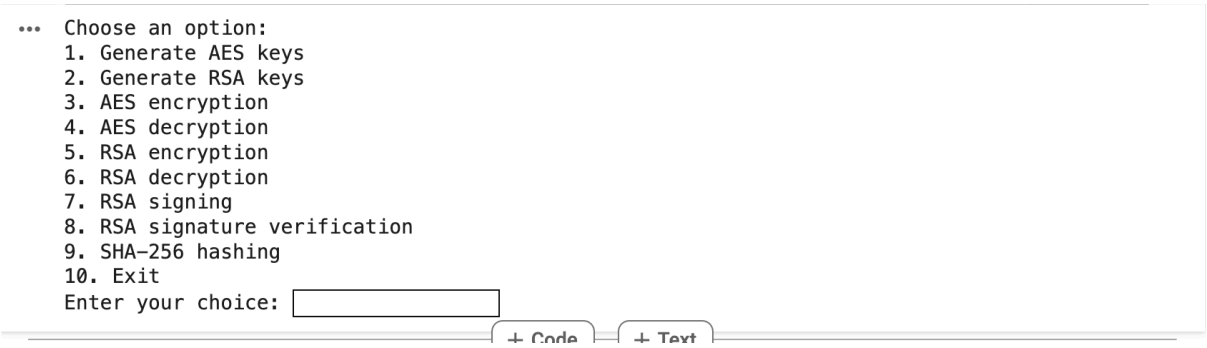
```
def sha256_hash(input_text):
    data = input_text.encode()
    h = SHA256.new(data).hexdigest()
    print(f"SHA-256 hash: {h}")
```

Measuring Execution Time:

```
def measure_execution_time(func, *args):
    start_time = time.time()
    func(*args)
    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"Execution time: {elapsed_time:.4f} seconds")
```

3. Using the Functionalities:

After the running the program, we will get these options:



3.1 AES Encryption/Decryption:

Encryption:

1. Choose option 1 for generating AES keys.
2. Choose option 3 for AES encryption.
3. Select ECB/CFB according to your need.
4. Select key length 128/256.
5. Enter text to encrypt. 6. Enter output file name.

```
Enter your choice: 3
Enter AES mode (ECB/CFB): ECB
Enter AES key length (128/256): 128
Enter text to encrypt: Hunter x Hunter
Enter output file name: anime
Encrypted text written to anime.
Execution time: 0.0152 seconds
```

Decryption:

1. Choose option 4 for AES decryption.
2. Select ECB/CFB according to your need.
3. Select key length 128/256.
4. Enter input file name.

```
Enter your choice: 4
Enter AES mode (ECB/CFB): ECB
Enter AES key length (128/256): 128
Enter input file name: anime
Decrypted text: Hunter x Hunter
Execution time: 0.0018 seconds
Choose an option:
```

3.2 RSA Encryption/Decryption:

Encryption:

1. Choose option 2 for generating RSA keys.
2. Choose option 5 for RSA encryption.
3. Enter text to encrypt.
4. Enter output file name.

```
Enter your choice: 5
Enter text to encrypt: Death Note
Enter output file name: light
Encrypted text written to light.
Execution time: 0.0511 seconds
```

Decryption:

1. Choose option 6 for RSA decryption.
2. Enter input file name.

```
Enter your choice: 6
Enter input file name: light
Decrypted text: Death Note
Execution time: 0.0649 seconds
Choose an option:
```

3.3 RSA Signature Signing:

1. Choose option 7 for RSA signing.
2. Enter text to sign.
3. Enter signature file name.

```
Enter your choice: 7
Enter text to sign: Demon Slayer
Enter signature file name: hashira
Signature written to hashira.
Execution time: 0.0532 seconds
```

Signature verification:

1. Choose option 8 for signature verification.
2. Enter text to verify.
3. Enter signature file name.

```
Enter your choice: 8
Enter text to verify: Demon Slayer
Enter signature file name: hashira
RSA signature verification successful.
Execution time: 0.0485 seconds
```

3.4 SHA-256 Hashing Hashing:

1. Choose option 9 for sha256 hashing.
2. Enter text to hash.

```
Enter your choice: 9
Enter text to hash: AOT
SHA-256 hash: 7117e28b7816d84c650fc33a471bfe49e94610a69bfe49c2d70c9fdeac20cc5c
Execution time: 0.0012 seconds
```

4. Analysing execution time:

AES Encryption:

Operation	Key Length (bits)	Mode	Time (Seconds)
Encryption	128	ECB	0.0152
Encryption	128	CFB	0.0155
Encryption	256	ECB	0.0197
Encryption	256	CFB	0.0129

AES Decryption:

Operation	Key Length (bits)	Mode	Time (Seconds)
Decryption	128	ECB	0.0018
Decryption	128	CFB	0.0028
Decryption	256	ECB	0.0021
Decryption	256	CFB	0.0037

RSA Encryption/Decryption:

Operation	Time (seconds)
Encryption	0.0511
Decryption	0.0649

RSA Signature:

Operation	Time (seconds)
Signature generation	0.0532
Signature verification	0.0485

SHA-256 Hashing:

Operation	Time (seconds)
SHA-Hashing	0.0012

5. Conclusion:

The resources given below:

https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption

https://www.devglan.com/online-tools/rsa-encryption-decryption

https://stackoverflow.com/questions/9316437/how-to-decrypt-a-sha-256-encrypted-string