Title: Programming Symmetric and Asymmetric Cryptography (Lab 4)

Author: Shattik Bandyopadhyaa
Reg No: 2019831039
Date: July 7, 2024

1. Introduction

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

2. Example code snippets

AES Encryption:

```python
# 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:

```python
# 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:

```python
# 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:

```python
# 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:

```python
# 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:

```python
# RSA signature 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:

```python
# 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:

```python
# Function to measure and report 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

When you run the program, you will be presented with a menu of options:

```
(base) PS E:\4-1\INS\Information_Network_Security_Lab_Tasks\Lab 4> & "e:/4-1/INS/Information_Network_Security_Lab_Tasks/Lab 4/.venv/Scripts/python.exe" "e:/4-1/INS/Information_Network_Security_Lab_Tasks/Lab 4/cs.py"
Choose an option:
1. Generate AES keys
2. Generate RSA keys
3. AES encryption
4. AES decryption
5. RSA encryption
6. RSA decryption
7. RSA signing
8. RSA signature verification
9. SHA-256 hashing
10. Exit
```

Choose an option:

1. Generate AES keys

2. Generate RSA keys

3. AES encryption

4. AES decryption

5. RSA encryption

6. RSA decryption

7. RSA signing

8. RSA signature verification

9. SHA-256 hashing

10. Exit

Here, you can choose any of the options according to your need.

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: This is ecb 128
Enter output file name: ecb_128_encrypted
Encrypted text written to ecb_128_encrypted.
Execution time: 1.4697 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: ecb_128_encrypted
Decrypted text: This is ecb 128
Execution time: 0.0160 seconds
```

3.2 RSA encryption/decryption

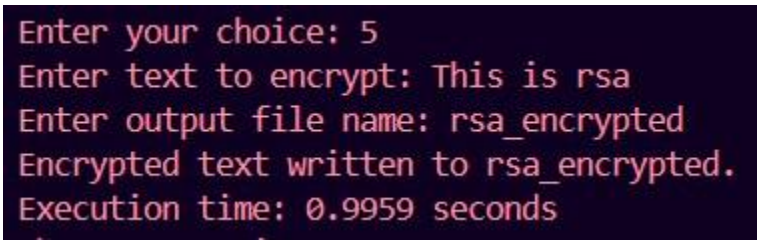
Encryption:

1. Choose option 2 for generating RSA keys.

2. Choose option 5 for RSA encryption.

3. Enter text to encrypt.

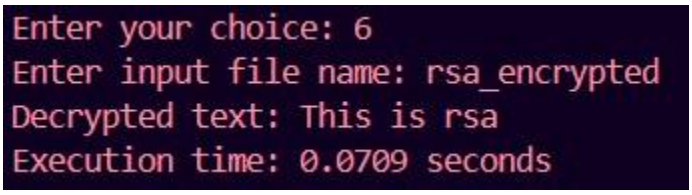
4. Enter output file name.



Decryption:

1. Choose option 6 for RSA decryption.

2. Enter input file name.



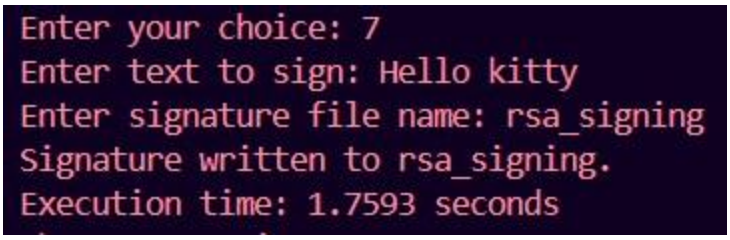
3.3 RSA Signature

Signing:

1. Choose option 7 for RSA signing.

2. Enter text to sign.

3. Enter signature file name.

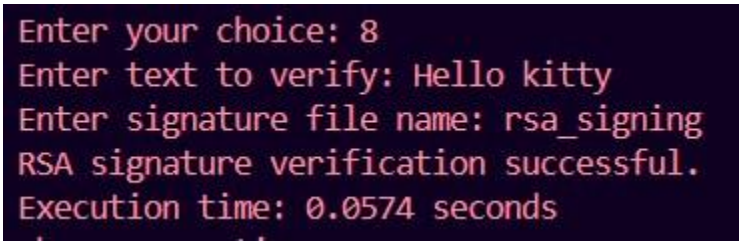


Signature verification:

1. Choose option 8 for signature verification.

2. Enter text to verify.

3. Enter signature file name.

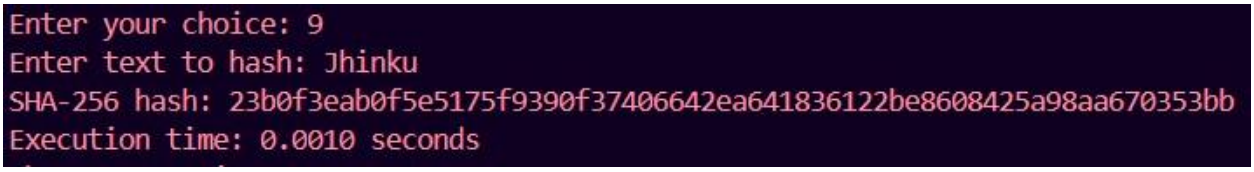


3.4 SHA-256 Hashing

Hashing:

1. Choose option 9 for sha256 hashing.

2. Enter text to hash.



4. Analyzing execution time

In this section, we will see the execution time for different cryptographic operations.

AES encryption:

Operation	Key Length (bits)	Mode	Time (seconds)
Encryption	128	ECB	1.4697
Encryption	128	CFB	1.2120
Encryption	256	ECB	0.0020
Encryption	256	CFB	0.0073

AES decryption:

Operation	Key Length (bits)	Mode	Time (seconds)
Decryption	128	ECB	0.0160
Decryption	128	CFB	0.0137
Decryption	256	ECB	0.0091
Decryption	256	CFB	0.0196

RSA encryption/decryption:

Operation	Time (seconds)
Encryption	0.9959
Decryption	0.0709

RSA Signature:

Operation	Time (seconds)
Signature generation	1.7593
Signature verification	0.0574

SHA-256 Hashing

Operation	Time (seconds)
SHA-Hashing	0.0010

5. Plotting

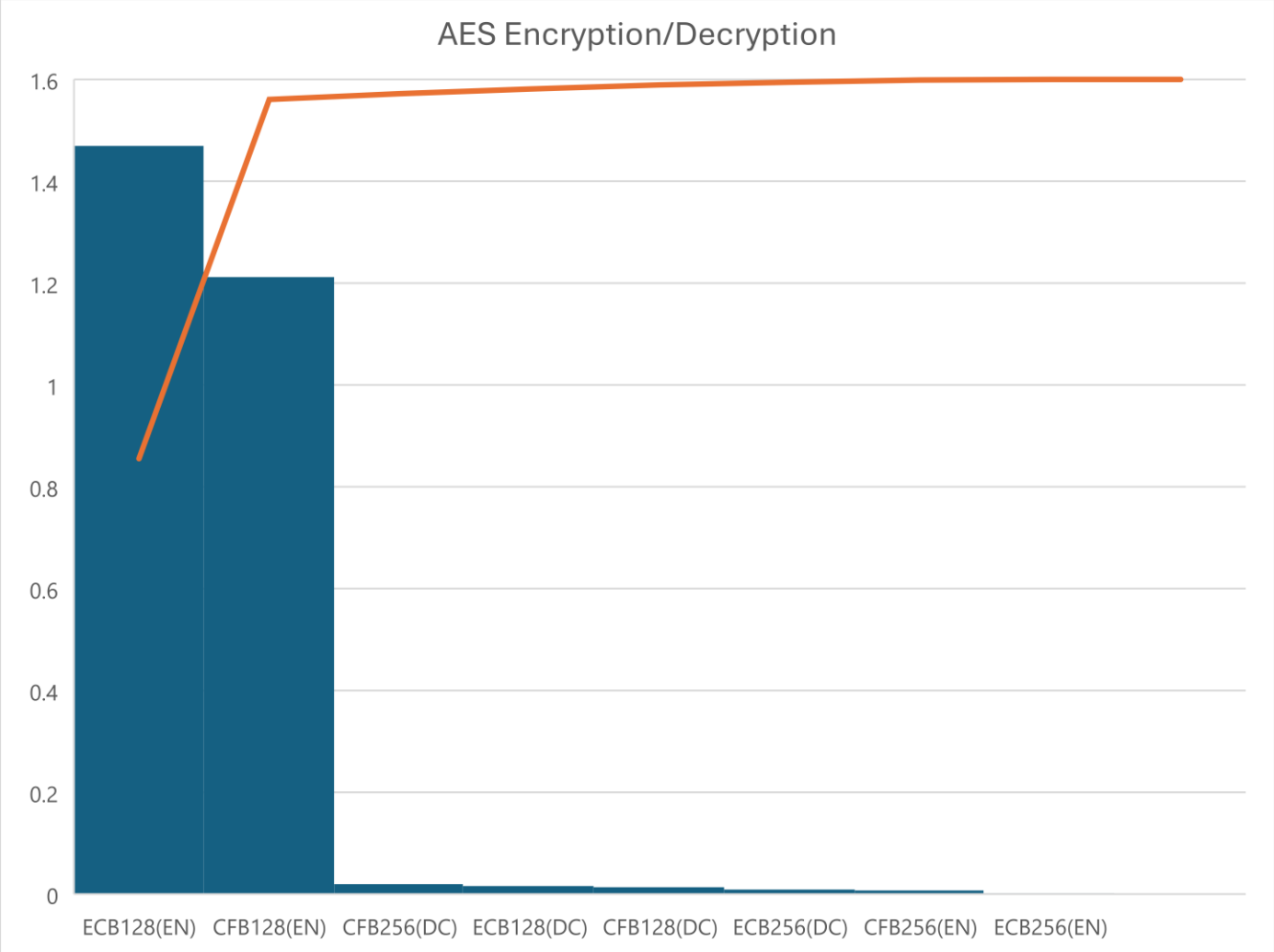


Fig: AES encryption/decryption

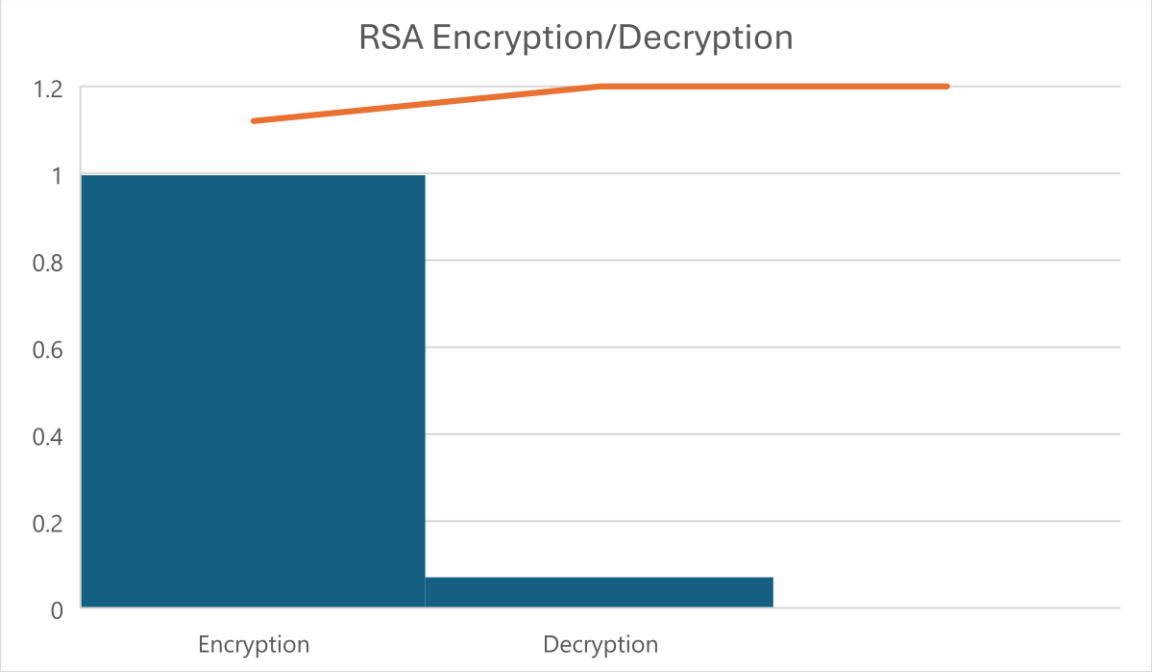


Fig: RSA encryption/decryption

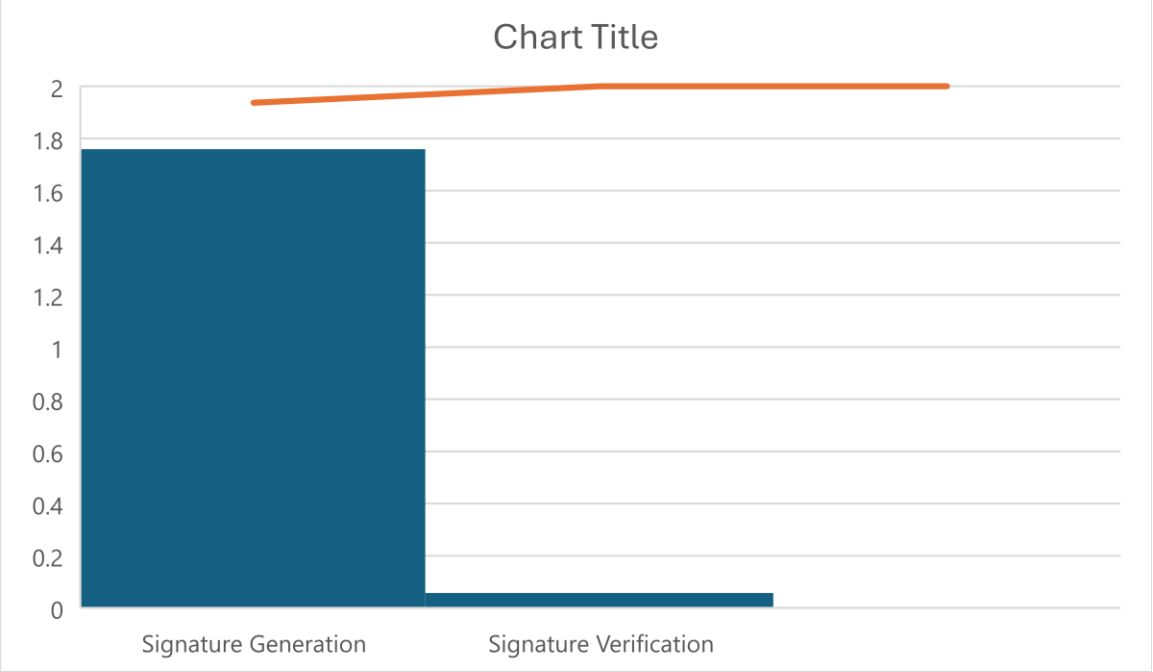


Fig: RSA Signature Generation/Verification

6. Conclusion

I have collected the necessary resources from the following sites:

https://brilliant.org/wiki/rsa-encryption/

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

https://cryptobook.nakov.com/digital-signatures/rsa-sign-verify-examples