Lab Report – 4

Programming Symmetric & Asymmetric Cryptography

1. Objective

The objective of this laboratory exercise is to **implement both symmetric and asymmetric cryptographic operations** through programming and to understand their underlying mechanisms beyond using pre-built tools like OpenSSL. Specifically, the goals are:

- To implement **AES encryption and decryption** using **128-bit** and **256-bit keys** in **ECB** and **CFB** modes.
- To perform **RSA encryption and decryption** using public and private keys.
- To implement **RSA digital signature generation and verification**.
- To compute the **SHA-256 hash** of a file.
- To measure the **execution time** of cryptographic operations for various key sizes and plot the performance graphs.

This lab helps in understanding the fundamental working of cryptographic algorithms and their computational characteristics.

2. Introduction

Cryptography is the foundation of modern data security, ensuring confidentiality, integrity, and authenticity of digital communications.
Two major categories of cryptography are:

- **Symmetric Key Cryptography:** The same key is used for both encryption and decryption (e.g., AES).
- **Asymmetric Key Cryptography:** Uses a pair of mathematically related keys (public/private), as in RSA.

In this lab, both types are implemented programmatically using **Python and the PyCryptodome library**, and the computational performance of each is analyzed.

3. Tools and Environment

Tool	Purpose
Python 3.x	Programming language
PyCryptodome	Cryptographic library
Matplotlib	Plotting execution time graphs
OpenSSL (for comparison)	Reference for command-line encryption
System	Linux (any platform supporting Python 3)

4. Methodology

The experiment consists of five major parts, each executed through a **menu-driven Python program (`lab4_crypto.py`)**.

4.1 AES Encryption and Decryption

- AES (Advanced Encryption Standard) implemented with **128-bit** and **256-bit keys**.
- Modes used:
    - **ECB (Electronic Code Book)** – simplest block mode without chaining.
    - **CFB (Cipher Feedback)** – uses IV and provides randomness.
- Keys are generated once, stored in files, and later re-used.
- Encrypted output stored as `aes_encrypted.bin`.
- Decryption result displayed on the console.

4.2 RSA Encryption and Decryption

- RSA key pair generated (typically **2048 bits**).
- **Public key** encrypts data, and **private key** decrypts it.
- Files generated:

    - `rsa_public.pem`
    - `rsa_private.pem`
- Encrypted output saved as `rsa_encrypted.bin`.

4.3 RSA Signature and Verification

- The **SHA-256 digest** of the target file is computed.
- The **private key** signs the hash, producing `signature.bin`.
- Verification is done using the **public key**.
- Successful verification confirms authenticity and integrity.

4.4 SHA-256 Hashing

- Computes SHA-256 hash of a file directly using Python's `hashlib` module.
- Output displayed in hexadecimal format.

4.5 Performance Analysis

- Execution time measured for various key sizes:
    - AES: 16, 32, 64, 128, 256 bits
    - RSA: 512, 1024, 2048, 3072, 4096 bits
- The program records the time required for each encryption and plots the graph using Matplotlib.
- The generated graph (`timing_plot.png`) visualizes **time vs key size** for AES and RSA.

5. Code Overview

The program `lab4_crypto.py` is a single Python file with modular functions for each task. It uses the **PyCryptodome** library for AES and RSA operations.

Key Sections of Code:

- `aes_encrypt()` / `aes_decrypt()` – handles AES operations.
- `generate_rsa_keys()` / `rsa_encrypt()` / `rsa_decrypt()` – manages RSA.
- `rsa_sign()` / `rsa_verify()` – performs digital signing and verification.
- `sha256_hash()` – computes file hash.
- `measure_time()` – measures execution time and plots graphs.
- `main()` – user-interactive menu for all operations.

6. Results

6.1 AES Operations

- Successfully encrypted and decrypted files with AES-128 and AES-256 keys.
- ECB mode produced consistent ciphertext for identical blocks, whereas CFB added randomness through IV.

6.2 RSA Operations

- Encryption and decryption using RSA key pairs functioned correctly.
- Larger key sizes significantly increased encryption time.

6.3 RSA Signature

- Signature generation created a unique `signature.bin` file.
- Verification passed successfully, confirming data integrity.

6.4 SHA-256 Hash

- SHA-256 hash of a sample file computed successfully and matched OpenSSL results.

6.5 Execution Time Analysis

- **Observation:**
    - AES times remained nearly constant due to efficient symmetric operations.
    - RSA times grew exponentially with key size, highlighting the computational cost of asymmetric encryption.

7. Graphical Analysis

Graph 1: Execution Time vs Key Size for AES and RSA

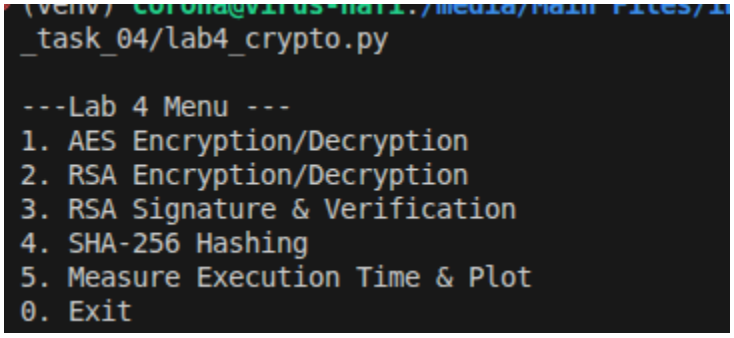
(Insert Screenshot of timing_plot.png here)
Figure 1: Comparison of execution times for AES and RSA across key sizes

Observation:
AES performs much faster than RSA for all key sizes, reaffirming why AES is preferred for bulk data encryption, while RSA is typically used for key exchange or signatures.

---

8. Screenshots

- Screenshot 1: Program Menu



- Screenshot 2: AES Encryption and Decryption Output

```
Select option: 1
Enter AES key length (128 or 256): 128
Enter mode (ECB/CFB): ECB
Enter input filename to encrypt: lab_task_04/aes_input.txt
AES Encryption complete. Output: aes_encrypted.bin
Decrypted content:
 This is a test message for AES encryption
```

- Screenshot 3: RSA Encryption/Decryption Output

```
Select option: 3
Enter file to sign: rsa_input.txt
Signature generated and saved as signature.bin
Signature verification successful ✅
```

- Screenshot 4: SHA-256 Hash Display

```
Select option: 4
Enter file for SHA-256 hashing: rsa_input.txt
SHA-256: e477c4553e5e8d3534a796886130d6243dbd91d478f19eaaddda2423cf92d323
```

9. Discussion

- This lab successfully demonstrated end-to-end implementation of key cryptographic primitives.
- AES and RSA show clear differences in computational complexity and efficiency.
- Hashing and signing ensure integrity and authenticity, vital for secure communication.
- Practical understanding gained in key generation, storage, and secure data handling.

10. Conclusion

Through this lab, the fundamental working of both symmetric and asymmetric cryptography was implemented and analyzed.

AES proved to be efficient for large data encryption, while RSA, despite being slower, provided secure key management and digital signatures.

The exercise reinforced the importance of combining symmetric and asymmetric cryptographic methods in modern security systems.