Lab Manual-4 Report: Programming Symmetric & Asymmetric Crypto

Introduction

This lab report presents the implementation of a comprehensive cryptographic program that demonstrates both symmetric and asymmetric encryption techniques. The program implements AES encryption/decryption with multiple modes, RSA encryption/decryption, RSA digital signatures, SHA-256 hashing, and performance analysis of these cryptographic operations. The implementation provides a command-line interface similar to OpenSSL, allowing users to perform various cryptographic operations interactively.

Programming Language and Libraries Used

Programming Language: Python 3 (executed in Google Colab)

Libraries Used:

- **PyCryptodome:** Primary library for all cryptographic operations including AES, RSA, SHA-256, and digital signatures
- **Matplotlib:** For generating performance analysis graphs and visualizations
- **OS:** For file and directory management operations
- **Time:** For measuring execution time of cryptographic operations

Installation Instructions

Before running the main program, install the required dependencies by executing the following command in a Google Colab cell:

```
!pip install pycryptodome matplotlib
```

After installing the dependencies, run the main program file (`Lab_Manual_4.ipynb`) uploaded to the GitHub repository. The program will display a menu with various cryptographic operation options.

Program Features and Usage

Main Menu Options

The program provides the following functionalities through an interactive menu:

1. **AES Encrypt (ECB Mode)**
2. **AES Decrypt (ECB Mode)**
3. **AES Encrypt (CFB Mode)**
4. **AES Decrypt (CFB Mode)**
5. **RSA Encrypt**
6. **RSA Decrypt**
7. **RSA Sign File**
8. **RSA Verify Signature**
9. **SHA-256 Hash File**
10. **Generate All Keys**
11. **Run Performance Test**
12. **Exit**

Detailed Usage Instructions

1. AES Encryption/Decryption (Options 1-4)

- **Supported Key Sizes:** 128 bits, 256 bits
- **Supported Modes:** ECB (Electronic Codebook), CFB (Cipher Feedback)
- **Process:**
    1. Select the encryption or decryption option
    2. Enter the desired key size (128 or 256)
    3. For encryption: Enter the plaintext message
    4. Encrypted data is saved to `encrypted/aes_[keysize]_[mode].bin`
    5. Execution time is displayed upon completion
    6. For decryption: The program reads from the encrypted file and displays the decrypted plaintext

2. RSA Encryption/Decryption (Options 5-6)

- **Supported Key Sizes:** 1024, 2048, 3072, 4096 bits
- **Process:**
    1. Select RSA encryption or decryption
    2. Enter the desired key size
    3. For encryption: Enter the plaintext message (keep it short due to RSA limitations)
    4. Encrypted data is saved to `encrypted/rsa_[keysize].bin`
    5. Execution time is displayed
    6. For decryption: The program reads from the encrypted file and displays the decrypted text

3. RSA Digital Signature (Options 7-8)

- **Signing Process (Option 7):**

    1. Enter the filename of the file to be signed
    2. Enter the RSA key size
    3. The program computes the SHA-256 hash of the file

4. A signature is generated using the private key
5. The signature is saved to `signatures/signature_[keysize].sig`
6. Execution time is displayed

- **Verification Process (Option 8):**

1. Enter the filename to verify
2. Enter the RSA key size used for signing
3. The program reads the signature file
4. Verification result (VALID or INVALID) is displayed
5. Execution time is shown

4. SHA-256 Hashing (Option 9)

- Enter the filename to hash
- The SHA-256 hash is computed and displayed in hexadecimal format
- Execution time is shown

5. Key Generation (Option 10)

- Generates all required keys for AES and RSA operations
- AES keys: 128-bit and 256-bit
- RSA key pairs: 1024, 2048, 3072, and 4096 bits
- Keys are stored in the `keys/` directory

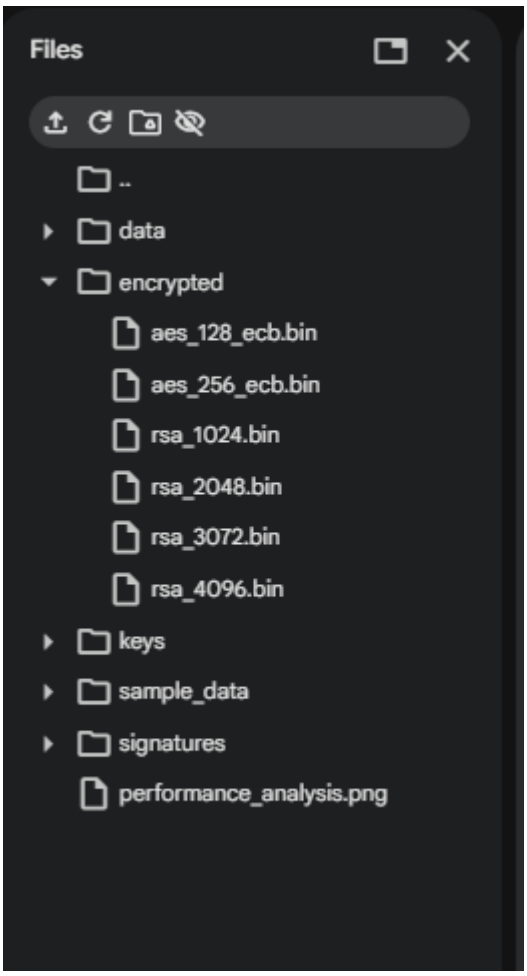
6. Performance Testing (Option 11)

The performance test measures and visualizes the execution time of cryptographic operations as a function of key size:

- **AES Testing:** Tests both 128-bit and 256-bit keys for encryption and decryption
- **RSA Testing:** Tests 1024, 2048, 3072, and 4096-bit keys for encryption and decryption
- Generates four graphs showing:
  - AES Encryption Performance
  - AES Decryption Performance
  - RSA Encryption Performance
  - RSA Decryption Performance
- The performance graphs are saved as `performance_analysis.png`

Directory Structure

The program automatically creates and manages the following directory structure:



Performance Analysis

The performance testing functionality provides insights into how key size affects the execution time of cryptographic operations:

Observations

1. **AES Performance:** AES operations (both encryption and decryption) show minimal variation between 128-bit and 256-bit keys, demonstrating the efficiency of symmetric encryption.

2. **RSA Performance:** RSA operations show significant increase in execution time as key size increases, particularly noticeable with 3072-bit and 4096-bit keys. This is expected due to the computational complexity of asymmetric cryptography.

3. **Encryption vs Decryption:** For RSA, decryption typically takes longer than encryption when using public key encryption schemes due to the mathematical operations involved with private key operations.

Code References and Attribution

The code implementation utilized the following resources and documentation:

Primary Documentation

- **Overall library structure and API reference:**
  https://pycryptodome.readthedocs.io/

Specific Implementations

1. **AES Implementation (ECB and CFB modes), padding mechanisms:**
   https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html
   Sections used: Key generation, cipher initialization, padding/unpadding

2. **RSA Key Generation, encryption, and decryption:**
   https://pycryptodome.readthedocs.io/en/latest/src/public_key/rsa.html
   Sections used: RSA.generate(), key export/import methods

3. **RSA Encryption with OAEP padding:**
   https://pycryptodome.readthedocs.io/en/latest/src/cipher/oaep.html
   Sections used: PKCS1_OAEP cipher encrypt/decrypt methods

4. **RSA Signature Generation and Verification:**
   https://pycryptodome.readthedocs.io/en/latest/src/signature/pkcs1_v1_5.html
   Sections used: PKCS#1 v1.5 signature scheme implementation

5. **SHA-256 Hash Implementation:**
   https://pycryptodome.readthedocs.io/en/latest/src/hash/sha256.html
   Sections used: Hash object creation and hexdigest methods

6. **Performance Visualization:**
   https://matplotlib.org/stable/contents.html
   Sections used: Subplot creation, line plotting, and graph customization

Conclusion

This lab successfully demonstrates the implementation of various cryptographic operations including symmetric encryption (AES), asymmetric encryption (RSA), digital signatures, and hashing. The program provides a user-friendly command-line interface and includes comprehensive error handling. The performance analysis reveals the trade-offs between security (key size) and computational efficiency, particularly evident in RSA operations. The modular code structure makes it easy to extend and maintain for future enhancements.