Lab Manual 4

Name: Shaeakh Ahmed Chowdhury

Registration Number: 2020831022

Date: November 8, 2025

Introduction

This lab report explains how I created a simple Python program to perform different cryptographic tasks. The program includes AES symmetric encryption, RSA asymmetric encryption, RSA digital signatures, and SHA-256 hashing. This report also explains how to run the program and what options are available.

Program Overview

The program can perform the following cryptographic tasks:

• AES Encryption/Decryption using 128-bit and 256-bit keys in ECB and CFB modes

• RSA Encryption/Decryption using 2048-bit key pairs

• RSA Digital Signatures for signing and verifying messages

• SHA-256 Hashing for generating message digests

• Execution Time Measurement for all operations

How to Run the Program

Before running, make sure you have Python 3 installed with the following packages:

pip install pycryptodome cryptography

Then run the program (example if it's in Jupyter Notebook): lab_manual_4.ipynb

The program will show a simple menu with options to choose from.

Program Output

When you run the program, it creates the following files automatically:
• aes_128.key - AES 128-bit key file
• aes_256.key - AES 256-bit key file
• rsa_private.pem - RSA private key
• rsa_public.pem - RSA public key

Program Options

Option 1: AES Encryption/Decryption
This feature lets the user encrypt and decrypt text using the AES (Advanced Encryption Standard) algorithm.

Steps:

1. Choose option 1 from the menu.
2. Select the AES key size — either 128 bits or 256 bits.
3. Choose the encryption mode — ECB or CFB.
4. Enter the text you want to encrypt.
5. The program will then show:
    - The encrypted text in hexadecimal format
    - The decrypted text (original message)
    - The time taken to complete the process

Example Output:

Enter AES key size (128 or 256): 128

Enter mode (ECB or CFB): ECB

Enter text to encrypt: testing

Encrypted: 1f24b4286386f23251d365d75a1bb944

Decrypted: testing

Elapsed time: 0.005234 seconds

Mode Explanation:

- ECB (Electronic Codebook): Encrypts data block by block (16 bytes each). It's simple but not very secure since identical input blocks create identical outputs.
- CFB (Cipher Feedback): Works like a stream cipher and uses an initialization vector (IV) for randomness. It's more secure than ECB because it produces different ciphertexts even for identical inputs.

Option 2: RSA Encryption/Decryption
This option performs asymmetric encryption using the RSA algorithm with a 2048-bit key pair.

Steps:

1. Choose option 2 from the menu.
2. Type the text you want to encrypt.
3. **The program will show:**
    - The encrypted message in hexadecimal format (256 bytes for a 2048-bit key)
    - The decrypted message (original text)
    - The time taken to complete the operation

Example Output:

Enter text to encrypt: testing

Encrypted: 1bf7bc3783ab25504131a291308e09c8... (256 bytes)

Decrypted: testing

Elapsed time: 0.008440 seconds

How It Works:

- The public key is used to encrypt the message.
- The private key is used to decrypt it.
- This ensures secure communication — anyone can encrypt using the public key, but only the private key holder can decrypt the message.

Option 3: RSA Signature
This feature is used to create and verify digital signatures using the RSA algorithm along with SHA-256 hashing.

Steps:

1. Choose option 3 from the menu.
2. Enter the text you want to sign.
3. The program will display:
    - The digital signature in hexadecimal format

- The verification result (whether the signature is valid or not)
- The execution time for the process

Example Output:

Enter text to sign: testing

Signature: 870f6fbaf857562839d197c74189359e... (256 bytes)

Signature verified successfully!

Elapsed time: 0.004687 seconds

How It Works:

- The program first creates a SHA-256 hash of the input message.
- This hash is then signed using the RSA private key.
- The RSA public key is used to verify the signature.
- This confirms that the message is authentic and has not been altered.

Option 4: SHA-256 Hash

This option generates SHA-256 cryptographic hashes of input data.

Steps

1. Select option 4 from the menu
2. Enter the text to hash
3. The program displays:

- SHA-256 hash in hexadecimal format (64 characters)
- Execution time in seconds

Example Output

```
Enter text to hash: testing
SHA-256 Hash:
cf80cd8aed482d5d1527d7dc72fceff84e6326592848447d2dc0b0e87dfc9a90
Elapsed time: 0.004588 seconds
```

Characteristics

- Produces a fixed 256-bit (64 hexadecimal character) output
- One-way function: impossible to recover input from hash
- Deterministic: same input always produces same hash
- Collision-resistant: extremely unlikely to find two different inputs with same hash

Option 5: Exit

Terminates the program execution.

References

1. PyCryptodome Documentation - AES, RSA, and Hashing:
https://pycryptodome.readthedocs.io
2. Python Official Documentation - hashlib, os, time modules: https://docs.python.org/3
3. Real Python Tutorials - AES and RSA Encryption: https://realpython.com
4. StackOverflow Examples for PyCryptodome usage

Technical Implementation

Libraries Used:
• PyCryptodome 3.23.0
• cryptography 43.0.3
• Python 3 standard modules: os, time, hashlib

Features:
• AES with 128/256-bit keys and ECB/CFB modes
• RSA 2048-bit encryption and signatures
• SHA-256 hashing
• Automatic key generation and file storage
• Execution time tracking for all operations

Security Notes

• ECB mode is less secure since it produces the same output for the same input.
• Keys are stored in plain files, which should be replaced by secure storage in real use.
• RSA 2048-bit keys are secure for now, but larger keys (3072/4096) are better for future security.
• The program uses secure random number generation (get_random_bytes).

Conclusion

This lab helped me understand how encryption and hashing actually work in practice. The Python program clearly shows the steps for AES and RSA operations, how to generate signatures, and how to measure their performance. Overall, it's a practical demonstration of key cryptography concepts.