Lab Manual-4

Amit Hassan Joy
Registration Number: 2020831013
Date: November 8, 2025

Objectives

1. To program symmetric and asymmetric cryptography operations.
2. To measure and study execution time for various cryptographic functionalities.
3. To gain a deeper understanding of cryptographic mechanisms beyond built-in tools like OpenSSL.

Implemented Functionalities

The program provides a menu-driven interface with the following functionalities:

- AES Encryption and Decryption (128-bit and 256-bit keys, ECB and CFB modes)
- RSA Encryption and Decryption (2048-bit key pair)
- RSA Digital Signature Generation and Verification
- SHA-256 Hashing
- Performance measurement of all operations

AES Operations

- Supports 128-bit and 256-bit keys.
- Two modes implemented: ECB and CFB.
- Encrypted data is stored in a file and decrypted output is displayed on the console.

Example Execution:

```
Enter choice: 1
Enter AES key length (128 or 256): 128
Enter AES mode (ECB or CFB): ECB
Enter data to encrypt: Deep Learning is fascinating!
AES encryption took 0.0111 seconds
```

Mode Descriptions:

- **ECB (Electronic Codebook):** Each block encrypted independently. Deterministic; identical plaintexts produce identical ciphertexts.
- **CFB (Cipher Feedback):** Stream cipher mode using an IV for randomization. More secure than ECB.

RSA Operations

- RSA key pair generated at program start.
- 2048-bit key used for encryption/decryption.

Example Execution:

```
Enter text to encrypt: testing
Encrypted: 1bf7bc3783ab25504131a291308e09c8... (256 bytes)
Decrypted: testing
Elapsed time: 0.00844 seconds
```

Working Principle:

- Public key encrypts plaintext.
- Private key decrypts cipher text.
- Enables secure communication where anyone can encrypt but only private key holder can decrypt.

RSA Digital Signatures

- Creates and verifies digital signatures using RSA and SHA-256.

Example Execution:

```
Enter text to sign: testing
Signature: 870f6fbaf857562839d197c74189359e... (256 bytes)
Signature verified successfully!
Elapsed time: 0.00469 seconds
```

How It Works:

- Computes SHA-256 hash of input text.
- Signs the hash using RSA private key.
- Verifies signature using RSA public key.
- Confirms message integrity and authenticity.

SHA-256 Hashing

- Generates a SHA-256 cryptographic hash of input text.

Example Execution:

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

Characteristics:

- Fixed 256-bit (64 hexadecimal character) output
- One-way function
- Deterministic
- Collision-resistant

Performance Measurement

The program measures execution time for each cryptographic operation.

Operation	Key Size (bits)	Avg Time (s)
AES Encryption	128	0.0111
AES Decryption	128	0.0013
RSA Encryption	2048	0.0014
RSA Decryption	2048	0.0017
RSA Signature	2048	0.0029
RSA Verification	2048	0.0004
SHA-256 Hashing	—	0.0011

Program Execution Instructions

1. Install Python 3 and dependencies:

```
pip install pycryptodome cryptography
```

2. Open `Lab_4.ipynb` in Jupyter Notebook.
3. Run the notebook. The interactive menu will appear with options:
    1. AES Encryption/Decryption
    2. RSA Encryption/Decryption
    3. RSA Signature
    4. SHA-256 Hashing
    5. Exit

Program Output Files

- `aes_128.key` – 128-bit AES key
- `aes_256.key` – 256-bit AES key
- `rsa_private.pem` – 2048-bit RSA private key

- `rsa_public.pem` – 2048-bit RSA public key

Security Considerations

1. **ECB Mode Limitation:** Produces identical ciphertexts for identical plaintext blocks. CFB is preferred.
2. **Key Storage:** Keys stored in plain files; production use requires secure key storage (KMS/HSM).
3. **RSA Key Size:** 2048-bit keys provide adequate security; for long-term security beyond 2030, consider 3072 or 4096-bit keys.
4. **Random Number Generation:** Uses cryptographically secure random number generation (`get_random_bytes()`).

Conclusion

The program successfully implements:

- AES symmetric encryption (ECB, CFB)
- RSA asymmetric encryption and digital signatures
- SHA-256 hashing
- Execution time measurement for each operation

The interactive menu interface enables understanding of each cryptographic operation, persistence of keys, and accurate performance measurement. This lab demonstrates practical cryptography programming in Python.