Report on
Lab 4: Programming Symmetric & Asymmetric Crypto

Overview

The Cryptography.ipynb file provides implementations for various cryptographic operations as outlined below:

1. AES Encryption and Decryption
2. RSA Encryption and Decryption
3. RSA Digital Signature
4. SHA-256 Hashing
5. Execution Time Evaluation

Implementation Details

- Language: Python 3
- Libraries:
  - cryptography (hazmat primitives) for AES, RSA, and hashing
  - matplotlib for visualization
- Key Storage:
  - AES keys stored as Base64 files
  - RSA keys stored in PEM format

Task 1: AES Encryption & Decryption

Implements AES with 128-bit and 256-bit keys in ECB and CFB modes.

Key Features:

- Keys generated using os.urandom()
- Stored as Base64 files (e.g., aes_128_ECB.key)
- CFB mode prepends a random 16-byte IV
- ECB mode uses padding to meet 16-byte block size

Testing:

- Input: *sample.txt (27 bytes)*
- Output: *Encrypted binary files with correct decryption results*
- AES-128 and AES-256 successfully recovered plaintexts in both modes

Task 2: RSA Encryption & Decryption

RSA encryption and decryption use OAEP padding with SHA-256.

Key Features:

- Key pairs: 2048-bit RSA by default
- Keys stored as:
  - rsa_private.pem (private)
  - rsa_public.pem (public)
- Limited data size due to RSA's key constraint

Testing:

- Input: *rsa_plain.txt (27 bytes)*
- Output: *rsa_encrypted.bin (256 bytes)*
- Decryption confirmed accurate recovery of plaintext

Task 3: RSA Signature

Implements digital signature creation and verification using PSS padding with SHA-256.

Key Features:

- Signing uses the private key
- Verification uses the public key
- Signatures stored in binary (e.g., sign.sig)

Testing:

- Signature file: *sign.sig (256 bytes)*
- Verification output: *Signature VALID*

Task 4: SHA-256 Hashing

Generates SHA-256 message digests for file integrity verification.

Key Features:

- Reads file, computes digest, and prints hex output
- Deterministic — identical files yield identical hashes

Testing:

- Input: *sample.txt (27 bytes)*
- Output: *64-character SHA-256 hash*

Task 5: Execution Time Measurement

Measures the execution time of cryptographic operations based on key size.

Key Features:

- AES: Tests 128-bit and 256-bit keys in ECB & CFB modes
- RSA: Tests 1024-, 2048-, 3072-, and 4096-bit keys
- Uses matplotlib for visual analysis (creates timing_results.png)

Graph & Performance Analysis

- AES: Minimal difference between key sizes or modes; both operations equally fast.
- RSA: Decryption time increases exponentially with larger key sizes; encryption grows linearly.

Implications

1. AES: Best for large data encryption due to high speed.
2. RSA: More efficient for encrypting small data (like AES keys).
3. Hybrid Model: Use RSA for AES key encryption and AES for bulk data.
4. Key Trade-off: Larger keys boost security but slow performance
   Lab 4 Report.

References

1. Python Cryptography Library: https://cryptography.io/en/latest/

2. Cryptography Library Documentation: https://cryptography.io/en/latest/hazmat/primitives/

3. Python Official Documentation: https://docs.python.org/3/

4. Matplotlib Documentation: https://matplotlib.org/stable/contents.html

5. Github Copilot AI