Report on
Lab 4: Programming Symmetric & Asymmetric Crypto

Overview of task: The Cryptography.ipynb file includes implementations for different cryptographic operations listed below:
- AES Encryption and Decryption
- RSA Encryption and Decryption
- RSA Digital Signature
- SHA-256 Hashing
- Execution Time Evaluation

Implementation Details
Language: Python 3.13

Libraries:
- cryptography (hazmat primitives) for AES, RSA, and hashing
- matplotlib for visualization

Key Storage:
- AES keys are stored as Base64 files
- RSA keys are stored in PEM format

Task - 1 (AES Encryption & Decryption)

This task implements AES with 128-bit and 256-bit keys in ECB and CFB modes.

Key Features:
- Keys are generated using os.urandom()
- They are stored as Base64 files (e.g. aes_128_ECB.key)
- CFB mode adds a random 16-byte IV
- ECB mode uses padding to match the 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)

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

Key Features:

- Key pairs are 2048-bit RSA by default
- Keys are stored as:
  - rsa_private.pem (private)
  - rsa_public.pem (public)
- Limited data size is 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)

Creates and verifies digital signatures using PSS padding with SHA-256.

Key Features:

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

Testing:

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

Task - 4 (SHA-256 Hashing)

This task generates SHA-256 message digests to verify file integrity.
Key Features:

- It reads the file, computes the digest, and prints the hex output
- It is deterministic; identical files will yield identical hashes

Testing:

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

Task - 5 (Execution Time Measurement)

This task measures the execution time of cryptographic operations based on key size.

Key Features:

- AES: Tests 128-bit and 256-bit keys in ECB and CFB modes

- RSA: Tests 1024-, 2048-, 3072-, and 4096-bit keys
- Uses matplotlib for visual analysis (creates timing_results.png)

Graph & Performance Analysis
AES: There is minimal difference between key sizes or modes; both operations are equally fast.
RSA: Decryption time increases significantly with larger key sizes; encryption grows at a slower rate.

Implications
AES is best for encrypting large data because of its high speed.
RSA is more efficient for encrypting small data, like AES keys.
In a Hybrid Model, use RSA for AES key encryption and AES for larger data.
There is a key trade-off; larger keys enhance security but can slow performance.

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