Task: Programming Symmetric & Asymmetric Cryptography

1. Objective

The goal of this lab was to understand how different cryptographic operations work internally by writing a single program that can perform:

- AES encryption and decryption (with 128-bit and 256-bit keys, ECB and CFB modes)
- RSA encryption and decryption
- RSA signature generation and verification
- SHA-256 hashing
- Additionally, the program measures and displays the execution time for each operation.

2. Tools & Environment

- **Programming Language:** Python
- **Libraries Used:** cryptography, pycryptodome, time, hashlib
- **Platform:** Ubuntu Subsystem (WSL) on Windows
- **Editor:** VS Code

3. Implementation

I created a single Python script that displays a menu when executed. From the menu, users can select which cryptographic operation to perform.

Functionalities Implemented

1. AES Encryption/Decryption

- Supports 128-bit and 256-bit keys.
- Allows users to select either ECB or CFB mode.
- Keys are generated and stored in files (aes_key.bin), and the ciphertext is saved in another file.
- Decryption reads the key and ciphertext, then restores the original plaintext.

2. RSA Encryption/Decryption

- Generates a public/private key pair.
- Encrypts and stores ciphertext in a file, and decrypts it back from that file.
- The keys are saved as rsa_pub.pem and rsa_priv.pem.

3. RSA Signature

- A file's SHA-256 digest is signed using the RSA private key.
- The signature is stored separately, and verification confirms file integrity.

4. SHA-256 Hashing

- Calculates the SHA-256 hash of any given file and displays it in the console.

5. Execution Time Measurement

- Before each operation, a timer starts using time.time().

- The elapsed time is displayed after the operation completes.
- Tested with multiple key sizes (16, 32, 64, 128, 256 bits).
- Observed that time increases with key size for both AES and RSA.

4. Commands & Testing

Example commands used for testing each part:

```
# AES Encryption (128-bit, ECB)
python3 crypto_lab.py --aes --mode ecb --keysize 128 --encrypt input.txt

# AES Decryption
python3 crypto_lab.py --aes --mode ecb --keysize 128 --decrypt aes_output.

# RSA Encryption/Decryption
python3 crypto_lab.py --rsa --encrypt message.txt
python3 crypto_lab.py --rsa --decrypt rsa_output.bin

# RSA Signature and Verification
python3 crypto_lab.py --sign file.txt
python3 crypto_lab.py --verify file.txt

# SHA-256 Hashing
python3 crypto_lab.py --hash file.txt
```

All commands worked successfully after proper key generation and file management.

5. Observations

- AES encryption and decryption completed quickly even for large files.
- RSA operations were noticeably slower for higher key sizes.
- Execution time consistently increased with larger keys.
- ECB mode was faster than CFB mode.
- Hash generation (SHA-256) was very fast and efficient.

6. Graph & Time Measurement

Execution time (in seconds) was recorded for both AES and RSA using different key sizes. A simple line plot was created to show that the processing time grows approximately linearly with key size.

I used code examples from online tutorials as references to understand AES and RSA implementation and then adapted them for this lab's requirements.

Code References:

- **AES & RSA implementation:** https://www.laurentluce.com/posts/python-and-cryptography-with-pycrypto/
- **RSA signature concept:** https://nitratine.net/blog/post/asymmetric-encryption-and-decryption-in-python/