Lab 4: Programming Symmetric & Asymmetric Crypto

Name: Amit Kumar Sharma

Registration: 2020831009

---

1. Objectives

- Implement **symmetric (AES)** and **asymmetric (RSA)** encryption/decryption in Python.
- Implement **RSA digital signatures**.
- Compute **SHA-256 hash** of a file.
- Measure **execution time** of cryptographic operations and visualize performance for different key sizes.

---

2. Project Structure

```
Lab4_CryptoProject/
│
├─ main.py                  # Main CLI program
├─ aes_module.py            # AES functions (encrypt/decrypt, key
generation)
├─ rsa_module.py            # RSA functions (encrypt/decrypt,
sign/verify)
├─ hash_module.py           # SHA-256 hashing
├─ benchmark.py             # AES & RSA benchmarking with
matplotlib graphs
├─ requirements.txt         # Python dependencies
│
├─ keys/                    # Stores AES and RSA keys
│   ├─ aes_key_128.key
│   ├─ aes_key_256.key
│   ├─ rsa_private.pem
│   └─ rsa_public.pem
│
├─ encrypted_files/          # Stores encrypted files
├─ decrypted_files/          # Stores decrypted files
└─ signatures/               # Stores RSA signatures
```

---

3. Dependencies and Installation

Requirements

- Python 3.10+
- Libraries: `pycryptodome`, `matplotlib`

Installation

1. Clone or copy project folder.
2. Create and activate a virtual environment:

```
python -m venv venv
venv\Scripts\activate   # Windows
```

3. Install required packages:

```
pip install -r requirements.txt
```

---

4. How to Run

1. Run the program:

```
python main.py
```

2. **Main Menu Options:**

```
1. AES
2. RSA
3. SHA-256
4. Benchmark
5. Exit
```

AES Usage

- Choose key size (128/256)
- Choose mode (ECB/CFB)
- Enter text to encrypt → encrypted file is saved in `encrypted_files/`
- Decrypted text saved in `decrypted_files/`

RSA Usage

- Enter text to encrypt
- Encrypted file saved in `encrypted_files/`
- Decrypted file saved in `decrypted_files/`
- Signature saved in `signatures/`
- Verification result printed in console

SHA-256 Usage

- Enter file path → hash printed in console

Benchmark Usage

- AES & RSA benchmark for different key sizes → plots generated as PNG

---

5. Example Output

AES Example

```
Choice: 1
AES key size (128/256): 256
Mode (ECB/CFB): CFB
Enter text to encrypt: Hello AES CFB test
Encrypted file: encrypted_files/aes_cfb.enc
Decrypted file: decrypted_files/aes_cfb_decrypted.txt
Time: 0.0032 sec
```

RSA Example

```
Choice: 2
Enter text for RSA: Hello RSA test
Encrypted file: encrypted_files/rsa.enc
Decrypted file: decrypted_files/rsa_decrypted.txt
Signature file: signatures/rsa.sig
Verified: True
Time: 0.0054 sec
```

SHA-256 Example

```
Choice: 3
Enter file path to hash: decrypted_files/rsa_decrypted.txt
SHA-256:
3a7bd3e2360a9c0f7a5f843d4e2b1c2f24d68e4f2c7b56d9d6aefb9e9f7f2f45
Time: 0.00045 sec
```

Benchmark Example

- AES and RSA execution times plotted in `AES_benchmark.png` and `RSA_benchmark.png`

---

6. Observations

1. **AES Execution Time**

    - CFB mode slightly slower than ECB due to chaining.
    - Execution time increases with key size.
2. **RSA Execution Time**
    - RSA encryption/decryption time increases significantly with key size.
    - RSA is slower than AES for large data.
3. **SHA-256 Hashing**
    - Very fast, independent of file size (small sample).
4. **Security Note**
    - AES 256-CFB is more secure than AES 128-ECB.
    - RSA key size ≥ 2048 bits recommended for practical security.

---

7. Resources Used

1. **Python AES & RSA Implementation**
   https://www.laurentluce.com/posts/python-and-cryptography-with-pycrypto/
2. **Java Crypto Concepts Reference**
   http://tutorials.jenkov.com/java-cryptography/index.html
3. **SHA-256 Hashing in Python**
   https://docs.python.org/3/library/hashlib.html
4. **Benchmark Plotting (Matplotlib)**
   https://matplotlib.org/stable/gallery/index.html

---

8. Conclusion

- Successfully implemented **AES, RSA, SHA-256** in Python.
- Able to generate keys, encrypt/decrypt files, create and verify signatures.

- Measured and visualized **execution times** for AES and RSA.
- Learned the difference in **performance and security** between symmetric and asymmetric algorithms.