

Lab Report: Programming Symmetric & Asymmetric Cryptography (Lab 4)

Introduction

This experiment demonstrates the implementation of both symmetric and asymmetric cryptography in Python using Google Colab.

The objective is to understand the working process of AES (Advanced Encryption Standard) and RSA (Rivest–Shamir–Adleman) algorithms for encryption and decryption.

It also includes generating and verifying digital signatures, hashing files using SHA-256, and comparing performance through timing analysis and graphical representation.

The program is menu-driven and easy to execute, showing results for each task such as encryption, decryption, hash generation, and time measurement.

Programming Language and Libraries Used

Language: Python 3 (Google Colab)

Libraries:

- **PyCryptodome** – for AES, RSA, SHA-256, and digital signatures
- **Matplotlib** – for plotting performance graphs
- **Hashlib, OS, Time** – for file handling, hashing, and measuring execution time

Installation Command:

```
!pip install pycryptodome matplotlib
```

After installing, the code file (Lab_Manual_4_code.ipynb) can be executed directly in Colab. A menu appears for selecting encryption, decryption, signature, or hashing options.

Program Menu:

===== CRYPTO LAB MENU =====

- 1) AES Encrypt/Decrypt (ECB/CFB)
- 2) RSA Encrypt/Decrypt
- 3) RSA Signature & Verification
- 4) SHA-256 Hash of Demo File
- 5) Timing Experiments + Plot
- 0) Exit

Working Procedure

1. AES (Option 1)

AES is a symmetric key encryption algorithm that uses the same key for encryption and decryption.

The program supports two modes:

- o ECB (Electronic Codebook)

- o CFB (Cipher Feedback)

The key size can be 128-bit or 256-bit.

The encrypted output is stored as a binary file, and decryption recovers the original message.

Execution time for both encryption and decryption is displayed.

2. RSA (Option 2)

RSA is an asymmetric cryptographic algorithm that uses two keys – a public key for encryption and a private key for decryption.

The program supports RSA key sizes of 1024, 2048, 3072, and 4096 bits.

A short text message is encrypted and then decrypted to show the original content along with encryption and decryption times.

3. RSA Signature (Option 3)

This option creates a digital signature for a file using an RSA private key and verifies it using the corresponding public key.

The hash of the file is generated using SHA-256 before signing.

The program displays whether the signature verification is valid or invalid and shows the time required for signing and verification.

4. SHA-256 Hash (Option 4)

This option reads a file and generates its SHA-256 hash value.

The hash output is displayed in hexadecimal format.

This ensures file integrity and data authenticity.

5. Performance Test (Option 5)

The program compares the performance of AES and RSA by measuring encryption and decryption times for different key sizes.

Execution times are plotted using Matplotlib to show the relationship between key size and processing speed.

Output and File Structure

After running the program, several folders and files are automatically created inside Colab:

```
crypto_lab/
    ├── aes_128.key
    ├── aes_256.key
    ├── rsa_2048_priv.pem
    ├── rsa_2048_pub.pem
    ├── rsa_4096_priv.pem
    ├── rsa_4096_pub.pem
    ├── rsa_2048.bin
    ├── rsa_2048.sig
    ├── demo.txt
    └── performance_analysis.png
```

Each operation displays the result in the Colab output, such as:

- Encrypted and decrypted messages
- Time taken for encryption and decryption
- Signature validation result
- SHA-256 hash value
- Performance graph showing AES and RSA comparison

Performance Analysis

AES performs faster than RSA because it is a symmetric key algorithm that uses smaller key sizes and simpler operations.

RSA takes more time as it involves larger key generation and complex mathematical calculations.

The performance graph shows that AES is efficient for short and fast data encryption, while RSA provides stronger but slower encryption suitable for secure key exchange and digital signatures.