Secure File Encryption System QORGAN

Zhetpiisov Alkhakiim, Yergazy Abdullayev, Lukman Bulatkan

Faculty of Engineering and Natural Sciences SDU

ARTICLE INFO

Keywords:
Cryptography,
Encryption,
Decryption

ABSTRACT

This document presents the design, architecture, implementation, and security analysis of a hybrid cryptography platform capable of file encryption, decryption, digital signature generation, and signature verification. The system leverages industry-standard algorithms, including AES-256-GCM, RSA-OAEP, RSA-PSS, and PBKDF2-HMAC-SHA256, providing both confidentiality and authenticity. The platform is implemented as a CLI tool and a web-based application. The goal of the project is to demonstrate applied cryptography using modern secure primitives, suitable for academic, research, and practical use..

1. Introduction

The objective of this project is to build a complete, user-oriented cryptographic solution that provides real security, while remaining simple to operate. Our platform demonstrates:

- Hybrid file encryption for confidentiality
- RSA-based digital signatures for authenticity
- Password-based protection of locally generated symmetric keys
- Web and CLI interfaces for usability in different scenarios

2. Project Overview

This project implements a full-stack cryptography platform that enables file encryption, decryption, digital signature generation, and signature verification.
The system uses a hybrid cryptographic model:

- **AES-256-GCM** — symmetric encryption
- **RSA-OAEP** — key encapsulation
- **RSA-PSS** — digital signatures
- **PBKDF2-HMAC-SHA256** — password-derived key protection

The project includes:

- Command-line interface (Python CLI)
- Web application with file upload capability (Flask)

The goal was to demonstrate practical cryptography using industry-standard primitives in a user-friendly application.

3. Architecture Design

- Core Components

Component	Specification
Symmetric Encryption	AES-256-GCM
Key Encapsulation	RSA-2048-OAEP
Digital Signature	RSA-PSS
Password Key Derivation	PBKDF2-HMAC-SHA256

Web Framework	Flask
CLI Framework	argparse

Hybrid Encryption Flow

```
User uploads file →
System generates AES key →
File encrypted with AES-GCM →
AES key encrypted using RSA-OAEP →
Result packaged and returned as .bin
```

Hybrid Decryption Flow

```
User uploads .bin →
RSA decrypts AES key →
AES decrypts file →
Original file returned
```

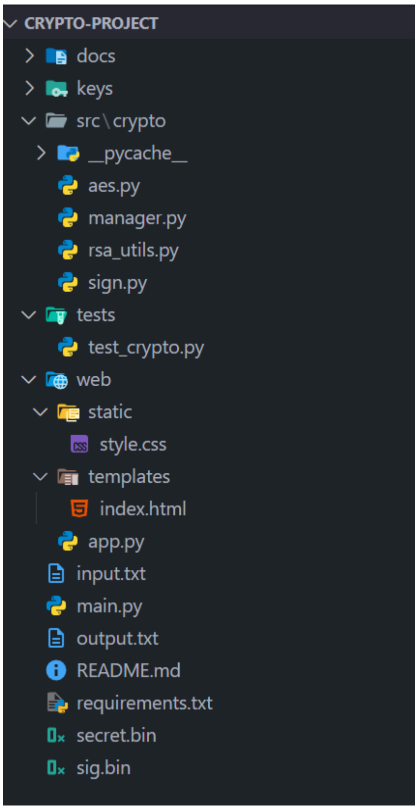
Digital Signature Flow

```
File → RSA-PSS → Signature (.sig)
```

Signature Verification Flow

```
File + .sig → Public RSA Key → VALID or INVALID
```

Project File Structure



4. Security Analysis Document

Security Assumptions

Assumption
RSA private key remains confidential
User password is not trivial
Application runs on trusted machine
HTTPS or local environment used
Public key distribution is authentic

Threat Model (STRIDE)

Threat	Type	Example
Spoofing	Identity	Signing as someone else
Tampering	Data	Altering ciphertext
Repudiation	Responsibility	Denying authorship

Information Disclosure	Privacy	Reading encrypted file
Denial of Service	Availability	Excessive verification
Elevation of Privilege	Integrity	Replacing public.pem

Potential Vulnerabilities

Vulnerability
Public key replacement (MITM key injection)
User chooses weak password
Filename integrity not protected
OS-level temp file exposure

Mitigation Strategies

Mitigation	**Addresses**
RSA-PSS signature	Spoofing & Repudiation
PBKDF2 200k iterations	Password brute-force
UUID temp file naming	Predictability attacks
Small output binary format	Tampering
No hardcoded keys	Source exposure

5. User Manual

Web Interface Usage

Operation	Result
Encrypt	Upload → Optional password → Download .bin
Decrypt	Upload .bin → Get original
Sign	Upload → Download .sig
Verify	Upload file + signature → VALID/INVALID

Warnings:

- Password must match
- Modified file will fail validation
- Large files take more time

CLI Manual (main.py)

Command	Example
Generate RSA keys	`python main.py generate-keys`
Encrypt a file	`python main.py encrypt input.txt out.bin --password secret`
Decrypt	`python main.py decrypt out.bin output.txt`
Sign	`python main.py sign file.txt file.sig`
Verify	`python main.py verify file.txt file.sig`

6. API Documentation

POST /encrypt

Field	Description
file	Raw file
password	Optional string

Response: .bin encrypted file

POST /decrypt

Field	Description
file	.bin encrypted
password	Optional string

Response: original file

POST /sign

Field	Description
file	Raw file

Response: .sig signature file

POST /verify

Field	Description
file	Raw file

signature	.sig file

Response: VALID or INVALID

7. Conclusion

This project demonstrates the design and implementation of a practical, secure, and accessible cryptography solution. By combining hybrid encryption, password-based key protection, and RSA-based digital signatures, the system provides confidentiality and authenticity of user files without requiring specialized tools or cryptographic background.

The architecture is modular, allowing independent usage via CLI or integration with a web-based interface. The system incorporates modern cryptographic standards and follows security practices that mitigate common risks related to data protection.

8. References

[1] RSA Laboratories – PKCS#1: RSA Cryptography Standard.

[2] Cryptography Python Library – https://cryptography.io

[3] RFC 8017 – PKCS#1 for RSA-OAEP and RSA-PSS.

[4] Flask Web Framework – https://flask.palletsprojects.com