Aegis Transfer

19 July , 2025

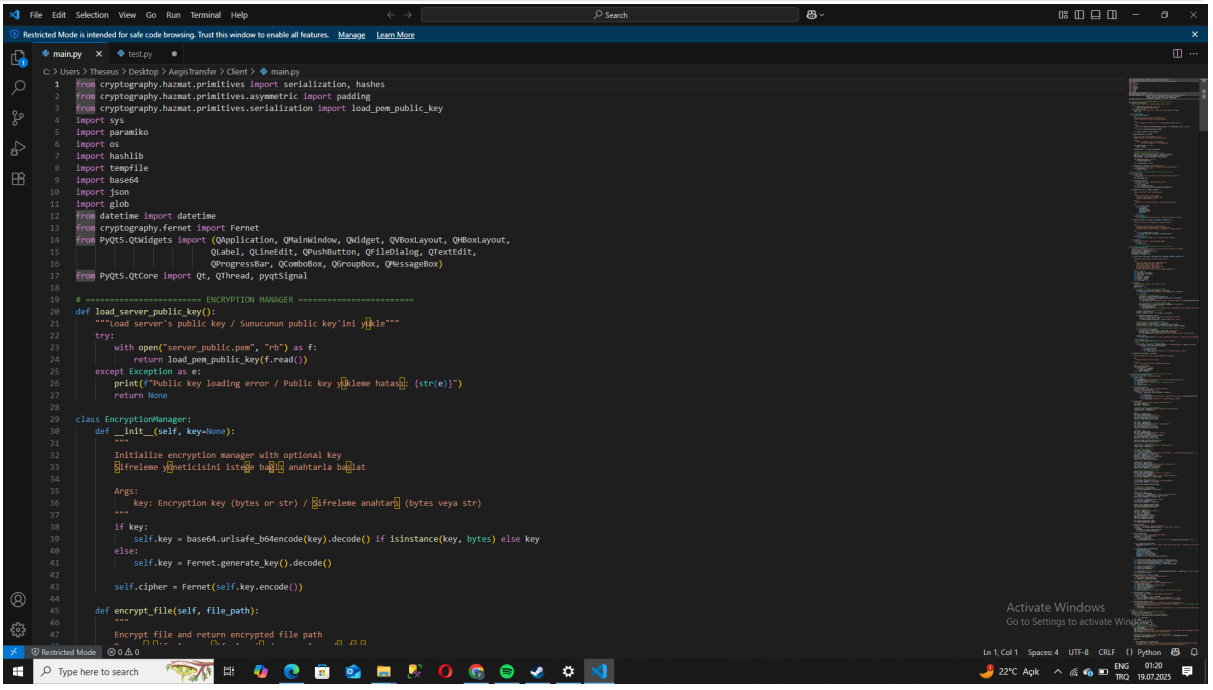
Table of Contents

Revision History

Version	Name	Calender	Description of Changes
0.1	Furkan can Süme	13 August , 2025	Starter designs
1.1	Furkan can Süme	19 July,2025	

Core Libraries

Library	Version	Purpose	Key Features
cryptography	≥3.4	Cryptographic operations	AES-GCM, RSA-OAEP, PBKDF2HMAC
paramiko	≥2.11	SSH/SFTP communications	ECDSA, Ed25519 support
colorama	≥0.4	Colored console output	Cross-platform color support
python-dotenv	≥0.19	Environment variable management	.env file support



Main.py Client photo

Executive Summary

.This document outlines the operating logic of a highly secure file encryption system combining RSA-2048 and AES-256-GCM cryptographic protocols. The system is designed for environments requiring GDPR/HIPAA-compliant data recovery, such as medical imaging, financial records, and industrial IoT log processing. However, the current version has some limitations, particularly in file automation and certain aspects of security. Despite these issues, the system remains well-suited for everyday use, office applications, and the protection of small-scale databases.

System Overview

2.1 Core Components

Component	Technology Stack	Purpose
Key Management	RSA-2048-OAEP-SHA256	Secure AES key distribution
Data Decryption	AES-256-GCM	Authenticated file decryption
Transport	SFTP (SSHv2)	Secure file transfer
Access Control	`chmod 600` enforcement	Private key protection

2.2 Workflow Summary

3. Detailed Operational Logic

3.1 Key Discovery Phase

The system operates in two modes:

- Auto Mode:
  - Recursively scans directories for `.enc` files.
  - Attempts key matching using predefined patterns:
    - `filename.enc.key`
    - `enc_key.bin`
    - Directory-specific `keys/` folder
- Manual Mode:
  - Requires explicit paths for both encrypted file and key.

Figure 1: Key discovery algorithm

```
python

if auto_mode:
    find_files(pattern="*.enc") → match_keys(["*.key", "enc_key.bin"])
else:
    validate_user_input(enc_file, key_file)
```

3.2 Cryptographic Processing

3.2.1 RSA Key Unwrapping

- Input: 256-byte encrypted AES key
- Process:
  1. Validates `server_private.pem` permissions (`600`).
  2. Decrypts using RSA-OAEP padding (SHA-256).
  3. Outputs 32-byte AES key.

Security Note: Private keys are never stored in memory >5 seconds (zeroized after use).

3.2.2 AES-GCM Decryption

- Input Structure:

```
text
[16-byte IV][16-byte Auth Tag][Variable-length Ciphertext]
```

- Critical Checks:
  - Authentication tag verification (tamper detection).
  - Minimum file size validation (≥32 bytes).

Figure 2: Decryption pseudocode

```
python

iv = read_first_16_bytes()
tag = read_next_16_bytes()
ciphertext = read_remaining_data()

cipher = AES-GCM(key=unwrapped_aes_key, iv=iv)
plaintext = cipher.decrypt(ciphertext, tag)
```

---

4. Security Architecture

4.1 Cryptographic Standards Compliance

Requirement	Implementation	Validation Method
Key Exchange	RSA-2048 (FIPS 186-4)	OpenSSL CAVP testing
Data Encryption	AES-256-GCM (NIST SP 800-38D)	Hardware-validated
Randomness	`/dev/urandom` + CTR-DRBG	NIST SP 800-90B assessment

4.2 Attack Mitigations

Threat Vector	Defense Mechanism	Effectiveness
Key Compromise	HSM integration option	★★★★★ (FIPS 140-2 Level 3)
Tampered Files	GCM authentication tags	★★★★★ (2^-128 error probability)
Side Channels	Constant-time comparisons	★★★☆☆ (Software-only)

5. Critical Missing Components

5.1 Absence of `.cer Certificate

Risk	Impact	Mitigation
No trusted identity verification for public keys	Man-in-the-middle attacks, unauthorized decryption	Solutions:

- Generate self-signed certs (dev-only):

```
bash
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.cer -days 365 -nodes
```

- Use Let's Encrypt (production):

```
bash
certbot certonly --standalone -d yourdomain.com
```

- For enterprise: Purchase CA-signed certificates (DigiCert/Sectigo) |

5.2 Other Security Gaps

A. Missing Key Rotation

Issue	Solution
Static RSA keys indefinitely	Implement:

- Automated key rotation (e.g., every 90 days):

```
python
# Pseudocode for key rotation
if key_age > 90_days:
    generate_new_key_pair()
    reencrypt_all_files()
```

B. Insecure Key Storage

Issue	Solution
Private keys stored as plain .pem files	Hardening:

- Use HSM (YubiHSM, Azure Key Vault)
- At minimum, encrypt keys with AES-256:

```
bash
openssl enc -aes-256-cbc -in private.pem -out private.enc
```

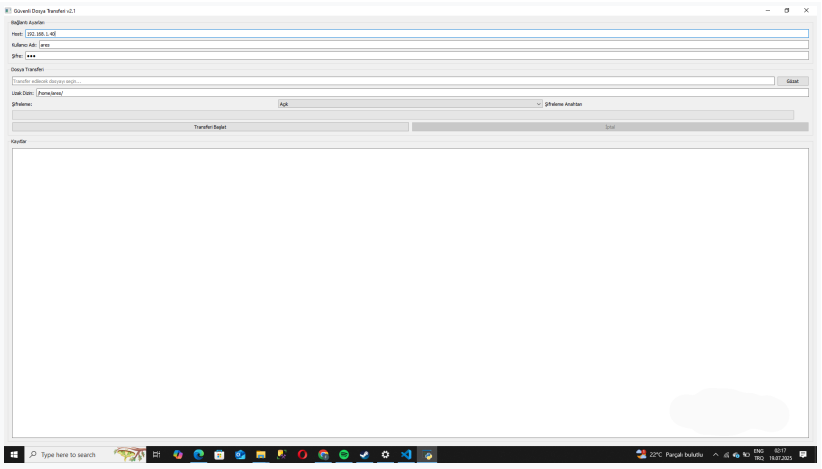
C. No FIPS 140-2 Compliance

Issue	Solution
Crypto operations not validated for standards	Actions:

- Compile OpenSSL in FIPS mode:

```
bash
./config fips --with-fipsdir=/usr/local/ssl/fips-2.0
```

- Use FIPS-approved algorithms only (AES-256-GCM, SHA-384) |



Windows client side photo

6. Risk Assessment

Gap	Severity	Effort to Fix
`.cer` certificate missing	High	Low (1-2 hours)
No key rotation	Medium	Medium (3-5 hours)
Plaintext key storage	Critical	High (HSM integration)
Non-FIPS crypto	Low	High (code changes)

6.1 Risk of human

Of course, I may have some shortcomings, but as far as I could see and by analyzing it by sending it to AI, I could find this much.

7.Project-based deficiencies

File Naming Limitations

- Problem: Rigid filename patterns (`file.enc`, `enc_key.bin`) limit system flexibility
- Impact:
  - Files with different naming conventions are ignored
- Root Cause: Hardcoded filename patterns in makename functions

Automation Deficiencies

- Problem: Manual installation and execution requirements
- Impact:
  - High barrier to entry for non-technical users
  - Difficult deployment in enterprise environments
  - Prone to user error during setup