Secure File Transfer System

420-A12-AS C2_INFORMATION SECURITY

Final Project Report

APRIL 17, 2024

Manoj Sharma (2330575)

Omid Moridnejad (2331293)

Thuvaarakkesh Ramanathan (2331425)

Table of Contents:

1) Introduction:

The Secure File Transfer System is a Python-based application designed to facilitate secure file transfers while ensuring confidentiality and integrity using RSA encryption and hashing algorithms. This report provides an overview of the system's design, implementation details, testing strategy, and additional considerations.

2) Design Overview:

RSA Key Generation: Implemented using the cryptography library, RSA key pairs are generated with a key size of 2048 bits and a public exponent of 65537. Keys are serialized to PEM format for portability and compatibility.

File Encryption and Decryption: Files are encrypted using RSA encryption with OAEP padding and SHA256 hashing for enhanced security. The recipient's public key is utilized for encryption, while decryption is performed using the corresponding private key. This ensures confidentiality and integrity of transferred data, safeguarding against unauthorized access and tampering.

Hashing: SHA-256 hashing is used to generate unique hashes for files, ensuring data integrity during transit and providing resistance against tampering.

Integrity Verification: The system verifies the integrity of received files by comparing their computed hash with the original hash. If the hashes match, the file integrity is confirmed, indicating that the file has not been tampered with during transfer. This mechanism provides assurance of data integrity and helps detect any unauthorized modifications or corruption.

User Interface: A simple command-line interface is provided for user interaction, offering a menu-driven approach for executing key operations. Users are guided through the process with clear prompts and instructions, ensuring ease of use and accessibility.

3) Key Decisions:

File Handling Approach:

During the development of the Secure File Transfer system, key decisions were made to ensure its effectiveness, security, and reliability, echoing the design principles evident in the codebase. The selection of the RSA encryption algorithm, in conjunction with OAEP padding and SHA-256 hashing, mirrors the cryptographic choices made in the generate_key_pair() and encrypt_file() functions. These decisions were driven by RSA's robust security properties and its compatibility with the cryptography library, reflecting a commitment to leveraging established encryption standards for secure file transfer.

Error Handling Approach:

The Secure File Transfer system incorporates robust error handling mechanisms to enhance reliability and user experience, aligning closely with the codebase's design philosophy. Error handling is implemented throughout the system to detect and manage potential issues such as invalid inputs, file not found errors, and exceptions raised during cryptographic operations, ensuring smooth operation under various conditions.

User Interface Design:

The Secure File Transfer system features a streamlined user interface designed for simplicity, clarity, and ease of use. The interface provides intuitive access to the system's functionalities, guiding users through key operations seamlessly.

Additionally, the interface incorporates error handling mechanisms to validate user inputs and provide informative feedback in case of invalid choices or errors. By guiding users through the interaction process and offering clear prompts and error messages, the interface ensures a smooth and frustration-free user experience.

4) Testing Strategy and Coverage:

Comprehensive unit tests are implemented using the **unittest** framework to validate each functionality's correctness and robustness. Test cases cover key generation, file encryption/decryption, hashing, and integrity verification, ensuring thorough testing coverage across various scenarios.

- test_generate_rsa_key_pair: Verifies that RSA key pair generation returns non-None private and public keys.
- test_file_encryption_decryption: Tests file encryption and decryption operations, ensuring that the decrypted file matches the original data.
- test_generate_file_hash: Checks the correctness of file hashing by comparing the generated hash with the expected value.
- test_verify_integrity: Validates the integrity verification process by comparing the hash of the original file with the generated hash.
- setUp: Prepares the test environment by creating a test file and generating its hash.
- test_integrity_verification: Tests integrity verification for a file with unaltered content.
- test_integrity_verification_with_tampered_file: Verifies integrity verification behavior when the file content has been tampered with.
- test_integrity_verification_with_nonexistent_file: Ensures that integrity verification raises a FileNotFoundError when attempting to verify a nonexistent file.
- tearDown: Cleans up the test environment by removing the test file.

5) Additional Considerations:

RSA Padding Schemes: The system employs RSA OAEP padding, providing security against chosen-plaintext attacks and ensuring data confidentiality.

Performance Optimization: Performance optimizations can be explored to enhance system efficiency, particularly for large files and high-volume transfers, without compromising security.

Key Management: Robust key management features, such as key rotation and storage, can be incorporated to enhance security and scalability.

6) Conclusion:

The Secure File Transfer System provides a reliable mechanism for secure file transfers, addressing the critical requirements of confidentiality and integrity. By leveraging RSA encryption and hashing algorithms, the system ensures data security during transit, mitigating risks associated with unauthorized access and tampering. Continuous refinement and integration of advanced security measures will further enhance the system's resilience to evolving cyber threats.

7) Implementation of Code:

Below are the key components of the Secure File Transfer System implementation:

- Key Generation (function.py)
- File Encryption and Decryption (function.py)
- Hashing (function.py)
- Integrity Verification (function.py)
- User Interface (function.py)
- Unit Tests (uttest.py)

8) Code Repository:

- https://github.com/RT-Rakesh/RSACode_Final_Project/tree/main