Secure File Processing System Using Hybrid Cryptography

Computer Security and Cryptography
Assignment

July 5, 2025

1 Problem Overview

Design and implement a secure file processing system that combines symmetric and asymmetric cryptography to provide confidentiality, integrity, and secure key management. The system should be capable of handling files of arbitrary size by implementing a chunked encryption approach.

2 Problem Statement

You are tasked with developing a comprehensive cryptographic file processing system that implements the following security pipeline:

2.1 Encryption Pipeline

Step 1: **File Segmentation**: Split the input file into manageable chunks of fixed size (e.g., 1MB each)

Step 2: **Symmetric Encryption**: Encrypt each chunk using AES-256 with unique initialization vectors

Step 3: **Integrity Protection**: Generate cryptographic checksums (SHA-256) for each chunk

Step 4: **Key Management**: Encrypt the AES keys using RSA or ECC public key cryptography

Step 5: **Metadata Storage**: Store all metadata including checksums, chunk information, and encrypted keys

2.2 Decryption Pipeline

Step 1: **Metadata Validation**: Verify the integrity of metadata and encrypted chunks

Step 2: **Key Recovery**: Decrypt AES keys using RSA or ECC private key

Step 3: **Chunk Decryption**: Decrypt each chunk using recovered AES keys

Step 4: **Integrity Verification**: Verify checksums of decrypted chunks

Step 5: **File Reconstruction**: Combine decrypted chunks to reconstruct the original file

3 Technical Requirements

3.1 Cryptographic Specifications

- **Symmetric Encryption**: AES-256
- **Asymmetric Encryption**: RSA-4096 or ECC P-384
- **Hash Function**: SHA-256 for integrity verification
- **Key Generation**: Cryptographically secure random number generation

3.2 Implementation Requirements

- Support for files of arbitrary size (up to 512 MB)
- Chunk size configurability (default: 1MB)
- Secure key generation and storage
- Error handling and recovery mechanisms
- Memory-efficient processing for large files
- Cross-platform compatibility

4 System Architecture

4.1 File Structure

The system should generate the following file structure:

Listing 1: Output File Structure

```
output_directory/
          chunks/
                chunk_001.enc
                chunk_002.enc
                ...
          metadata.json
          checksums.txt
          encrypted_keys.bin
```

4.2 Metadata Format

Listing 2: Metadata JSON Structure

```
{
    "file_info": {
        "original_name": "document.pdf",
        "original_size": 15728640,
        "chunk_size": 1048576,
        "total_chunks": 15,
        "encryption_algorithm": "AES-256-GCM",
        "key_encryption": "RSA-4096"
    },
    "chunks": [
        {
            "chunk_id": 1,
            "encrypted_filename": "chunk_001.enc",
            "iv": "base64_encoded_iv",
            "checksum": "sha256_hash",
            "size": 1048576
        }
    ],
    "keys": {
        "encrypted_master_key": "base64_encoded_encrypted_key",
        "public_key_fingerprint": "key_fingerprint"
    }
}
```

5 Implementation Tasks

5.1 Core Functions to Implement

5.1.1 Encryption Script

Listing 3: Encryption Function Signature

```python
def encrypt_file(input_file_path, output_directory,
                public_key_path, chunk_size=1048576):
    """
    Encrypt a file using hybrid cryptography

    Args:
        input_file_path: Path to input file
        output_directory: Directory to store encrypted chunks
        public_key_path: Path to RSA/ECC public key
        chunk_size: Size of each chunk in bytes

    Returns:
        dict: Encryption metadata
    """
    pass
```

5.1.2 Decryption Script

Listing 4: Decryption Function Signature

```python
def decrypt_file(encrypted_directory, output_file_path,
                private_key_path):
    """
    Decrypt and reconstruct file from encrypted chunks

    Args:
        encrypted_directory: Directory containing encrypted chunks
        output_file_path: Path for reconstructed file
        private_key_path: Path to RSA/ECC private key

    Returns:
        bool: Success status
    """
    pass
```

5.2 Security Considerations

1. **Key Management**: Implement secure key generation, storage, and destruction
2. **IV Generation**: Use cryptographically secure random IVs for each chunk

3. **Integrity Verification**: Implement comprehensive checksum validation
4. **Error Handling**: Graceful handling of corruption and tampering
5. **Side-Channel Resistance**: Consider timing and memory access patterns

6 Deliverables

1. Complete source code with encryption and decryption scripts
2. Comprehensive documentation
3. Demonstration of the system working with files of different sizes
4. Analysis of security properties and potential vulnerabilities

7 Submission Guidelines

- Submit all code in a well-organized repository
- Include a detailed README.md file
- Provide test files and demonstration scripts
- Document any external dependencies
- Include performance analysis and security assessment

Something to Keep in Mind : This PS is meant for you to experiment, and learn by doing! Feel free to team up with others, tweak things as you go, and make it your own. What matters most is your effort and interest to learn more and have fun. If you have any ideas to make something better? Speak up! We're all ears and happy to chat about improvements. I wont give this note in all the docs again and again, so just keep it in mind for future tasks.