Lab 01: Hybrid Encryption

1 Introduction

In this lab, you will implement a simple hybrid encryption system for secure file exchange. The system's overview can be described via Figure 1

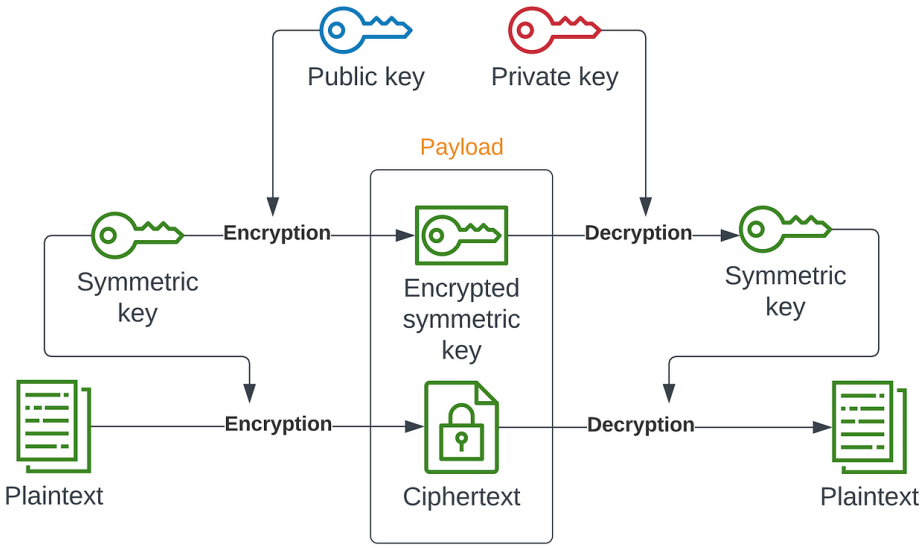

Figure 1: Demonstration of a hybrid encryption scheme (Source)

Given a file exchange scenario involving a sender and a receiver, the exchange flow can be described as follows:

1. The sender will generate a symmetric key and use that key to encrypt the file. Note that you must ensure the symmetric encryption algorithm would break the pattern so that the encrypted file doesn't have any patterns mapped to the original file.
2. The sender will use the receiver's public key to encrypt the symmetric key. This encrypted key will be sent with the encrypted file to the receiver.
3. The receiver will use his own private key to decrypt the symmetric key, and use the decrypted key to decrypt the file.

You will have to generate a pair of (public, private) keys to demonstrate your work.

2 Lab description

Requirements

Your system must fullfill these requirements:

- You will have an `encryptor` and `decryptor`, where
  - `encryptor` will take the receiver's public key and a file as input. It will output the encrypted symmetric key and the encrypted file. The symmetric key will be generated randomly by the encryptor and encrypted using the receiver's public key.

    Example command-line parameters for the `encryptor`:

    ```
    python encryptor.py --receiver_pub_key=receiver_pub_key.pub
                        --input_file=file_to_encrypt.txt
                        --output_encrypted_file=output_encrypted_file.txt
                        --output_encrypted_symmetric_key=encrypted_key.key
    ```

  - `decryptor` will take the receiver's private key, the encrypted file, and the encrypted symmetric key as input. It will decrypt and return the decrypted file.

    Example command-line parameters for the `decryptor`:

    ```
    python decryptor.py --receiver_private_key=receiver_private_key.key
                        --encrypted_key=encrypted_key.key
                        --input_file=encrypted_file.txt
                        --output_decrypted_file=output_decrypted_file.txt
    ```

- Your encryptor and decryptor should be able to work with text (`.txt`) and images (`.png`, `.jpg`, `.jpeg`) files. If it can work with more types of files, that would be a bonus.
- All components should be implemented as a command-line application using Python. Here are some resources that can help you with your implementation:
  - `pycryptodome`: cryptography library for Python
  - `argparse`: parser module for command-line applications.

Good-to-have

After decryption, can the receiver determine if the file is corrupted during the exchange phase, or just verify if the file originally came from the sender? To verify the originality & integrity of the file, an additional step is added to the process:

1. The sender will hash the file using a cryptographic hash function (MD5, SHA256, ..etc.), then encrypt the hash with the sender's private key. This means your implementation now includes an extra step: generating a (public, private) key pair for the sender. The encrypted hash will be sent along with the encrypted file and symmetric key.
2. The receiver, after decrypting the file, will hash it using the same cryptographic hash function. The hash will be compared to the decrypted hash (using the sender's public key) to see if they match.

This way, the receiver can verify that (1) the file is intact during the exchange phase and (2) the file originally came from the sender. In your implementation, print out the verification status (e.g., print "FILE IS VALID" or "FILE IS INVALID" depending on the result).

Rubrics

The evaluation rubrics for this lab are considered as follows (Table 1):

Content	Percentage
Keygen (symmetric & asymmetric)	20%
Encryption & Decryption (symmetric & asymmetric)	40%
Application (command-line)	10%
Demo video	10%
Report	20%
Multiple filetype support (bonus)	5%
File integrity verification (bonus)	5%

Table 1: Rubrics for Lab 01

3 Submission

- Students are required to submit one .zip folder named *SID.zip* (where SID is your student ID). This folder should contain the following materials:
  - `src/`: Contains all implemented source code.
    - **Precaution:** Please delete all `__pycache__`, `.ipynb-checkpoints`, `.git`, `.vscode`, `..etc.` folders before submitting your work. This will help ensure that your submission is not too large for the Moodle system.
  - `install.md`: Detailed instructions to install and run the code.
  - `report.pdf`: A **technical report**, in which you should (1) describe your system, (2) mention special parameters in your settings (for example, the size of the RSA key, the size of the AES key, the hash algorithm used, the file types supported, etc.).
  - `demo.txt`: This file should contain the link to your publicly shared demo video. You can upload your video to YouTube, Google Drive, MS OneDrive, or any other platform. Your demo video should show how all the components of your system work, including any bonus sections.
- You will submit your work through Moodle. We will not accept any late submissions for any reason.
- All cheating behavior (copying source code, content of reports, or any other materials from other students or the Internet without proper citing) will result in a 0 for **the whole course**.