Assignment 2 Cryptography File Sharing

Lise Johansen

29. February 2024

1 Introduction

This report outlines the design and implementation of a Python-based network file-transfer system. The primary objective is to ensure the confidentiality and integrity of the transmitted data. Guarding against key threats such as eavesdropping, man-in-the-middle and replay attacks, the system uses a symmetric AES encryption for data protection and asymmetric RSA encryption for keys exchange between a socket server and a socket client.

A simplification, the server stores a .txt or a .png file that the client requests. Libraries use are the cryptography and socket.

2 Background Theory

Encryption involves the transformation of plain-text into unintelligible ciphertext ensuring it's confidentiality.[5] The process of converting the ciphertext back into plaintext is known as **decryption**.

Eavesdropping occurs when an unauthorized entity intercepts network communications, enabling them to delete, modify, or gather data in transit between devices. [3] In the context of a network file-transfer system, this exposes confidential information to malicious actors.

In a **Man-in-the-Middle attack** an entity positioning themselves between two parties and relay and possibly alters the communications between sender and receiver. [1] The integrity and confidentiality of the file transfer is compromise leading to unauthorized access or manipulation of the transmitted message.

Replay attacks involves that the attacker eavesdrops on the communication and resend the message. [2]

To counter these threats, both **symmetric** and **asymmetric** encryption can be utilized. Symmetric encryption employs a single private key for both encryption and decryption. [5] While asymmetric encryption uses a public key to encrypt the ciphertext, and the corresponding private key to decrypt it. Specific encryption schemas that can be used are the **Advanced Encryption Standard** (aka AES) encryption and the **RSA encryption** schema. The symmetric encryption AES is fast and used for encryption and decryption of large amount of data. And to make the AES key exchange secure the asymmetric RSA encryption is used.[4]

3 Design and Implementation

3.1 Overall system design

The architecture of the system follows a modular design, with clear separation between the two primary components: the socket server-client, responsible for sending and receiving data, and the RSA and AES encryption mechanisms used to secure both the data and keys.

3.1.1 Server-client design

The figure bellow illustrates the socket server-client. The client sends a request and the server responds by sending a .txt or .png file. to the client. The socket server-client only communicates over a TCP socket.

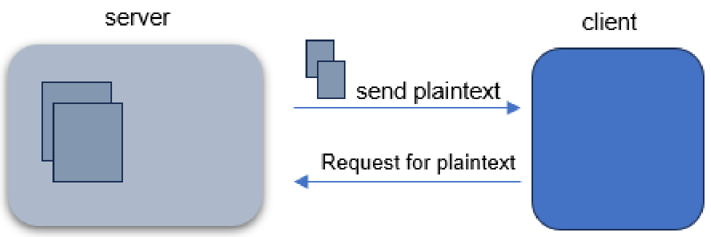


Figure 1: Simple illustration of the server-client file-sharing system.

The server side: Binds to a specified host and port. Then listens for incoming client connection. After accepting the client connection, a file request is received and a file is send to client as bytes. Then the client connection is closed.

The client side: After creating the client socket a file request is send to server as bytes. The client receives the file and saves it as plaintext.

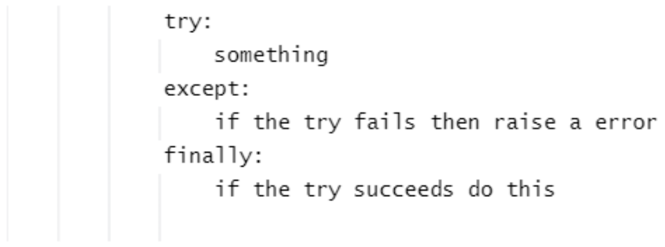
3.1.2 Encryption design

The following steps outlines the design of the encryption and decryption process.

Both the server and the client independently generate their RSA key pairs. The client transmits its public RSA key to the server. The server generates an AES key, encrypts it with the client's public RSA key, and sends the encrypted AES key to the client. The client utilizes its private RSA key to decrypt the received AES key. The server encrypts the file data with the AES key and transmits the encrypted data to the client. The client, after possessing the AES key, decrypt the received file.

3.2 Error Handling design

The server-client is set up using try-expects, as shown in the pseudocode below. The try block test the code, and the excerpt block handles unexpected behaviors or errors.



```
try:
    something
except:
    if the try fails then raise a error
finally:
    if the try succeeds do this
```

Figure 2

4 Implementation

4.1 Server-client implementation

Server: Set host and port values. Create and bind the server using a socket. Generate a private and public key. Accept the connection from the client and retrieve its address.

Client: Create the client using socket, try to connect client to the host and port. The clients public key is sent to the server.

Server: Inside a try: Receive the pubic key from the client, deserialize the received public key. And send this key along with a random 16 bytes to the client, encrypted.

Client: Receive the AES key from the server and decrypt it. Then encode the file name and send it to the server.

Server: Decode the received file name, in a inner try, open the file and read it. Encrypt the file content using the deserialize public key and then send the encrypted file to the client. For the inner try an except is used to check if the requested file is not found. The first try has an expect to catch general errors.

Client: Receives the file content from the server in chunks. If a file is successfully revived, save and decrypt it using the client's private key. Open the file to write it to memory. Then at the end an except is there to catch general errors.

Server: Finally the socket is closed.

4.2 Encryption Implementation

The process begins by generating a unique RSA key pair, including both private and public keys, with a key size of 2048 bits. Then the client transmits the public key to the server. Upon receiving the public key, the server deserializes it, preparing the key for encryption. The server then generates a random 16-byte AES key, encrypting it using the client's public RSA key.

For each file transfer, a new random 16-bit AES key is generated. This AES key is encrypted using the client's public key with RSA OAEP padding. A random initialization vector (IV) is generated for AES in CFB mode. The file data is encrypted using this generated AES key and the IV. The resulting AES key, IV, and ciphertext are sent to the client.

On the client side, the received AES key is decrypted using its private RSA key. With the AES key and IV, the ciphertext is decrypted.

4.3 Improvement

Incorporating HMAC tags is going to improve system security. HMAC tags utilize a cryptographic hash function and a secret key for data verification. When the client generates an HMAC key that matches the received HMAC key from the server, data integrity is verified, and the ciphertext can be decrypted.

The system transfer .txt and .png files. To showcase that the system's is not restricted to only one file type. To enhance versatility and usability, additional file types can be added. And a more dynamic method for clients to request files would further improve the system's flexibility.

5 Evaluation

5.1 Experiment Design

To test if the system works, execute the code and verify whether the observed outcomes is the expected results. Print statements in the code serve as "check-points," allowing for continuous monitoring of the system's behavior. The last expected outcome from the server are the message "Encrypted file sent to client successfully," and the last expected message from client are "File downloaded and decrypted successfully.". If this expected messages is produced the system behave as expected.

5.2 Results

Expected result are produced when running the system.

5.2.1 Size

Optimization of space was not a primary focus, as the assignment prioritized security over space efficiency. Possible since this is a small and well defined file-transfer system. But in a real word scenario further considerations is needed.

5.3 Overall Assessment

The results section demonstrates the successful execution of the file-transferring system, with no errors encountered during code execution. This affirms the encryption logic and validates that the system works as expected.

6 Conclusions

A successful file-transfer system was implemented. Using RSA and AES encryption to ensure a secure key exchange and data transfer. The system guards against man-in-the-middle, eavesdropping and replay attacks.

References

[1] Fortinet. *Man-in-the-middle-attack*. 9 February 2024. February 2024. URL: `https://en.wikipedia.org/wiki/Man-in-the-middle_attack`.

[2] Fortinet. *What Is a Replay Attack?* 9 February 2024. 2024. URL: `https://www.kaspersky.com/resource-center/definitions/replay-attack`.

[3] Fortinet. *What Is Eavesdropping?* 9 February 2024. September 2021. URL: `https://www.fortinet.com/resources/cyberglossary/eavesdropping`.

[4] Nicolas Poggi. *ypes of Encryption: Symmetric or Asymmetric? RSA or AES?* 11 February 2024. 15 June 2021. URL: `https://preyproject.com/blog/types-of-encryption-symmetric-or-asymmetric-rsa-or-aes`.

[5] et al. Sun. Xingming. *Symmetric Encryption*. 9 February 2024. 2008. URL: `https://www.sciencedirect.com/topics/computer-science/symmetric-encryption`.