INF-2310: Computer Security

Assignment 2: Cryptographic File Sharing

Amund Harneshaug Strøm

UiT id: ast283@uit.no

February 28, 2024

1 Introduction

This report describes the design and implementation of a secure network file-transfer system. The system consists of a server and a client that should be able to communicate and protect the confidentiality and integrity of the data while in transit. The system should protect against three types of attacks; Eavesdropping, Man-in-the-middle attacks, and Replay attacks.

2 Background Theory

These are some of the key topics encountered in this assignment and is important to understand before tackling the task at hand.

- **Symmetric ciphers** is a type of encryption algorithm that uses the same key for both the encryption and decryption of the data. The key is shared between the server and the client, and it must be kept secret to maintain the security of the communication. The primary goals of symmetric ciphers are confidentiality and integrity of the data. Symmetric ciphers are generally faster and more computationally efficient than asymmetric ciphers (discussed later), making them suitable for encrypting large amounts of data. However, the challenge lies in securely distributing and managing the secret keys between the server and client.

- **AES**, or Advanced Encryption Standard, is a symmetric encryption algorithm that is employed to secure sensitive data. It operates on fixed-size blocks of data and supports key lengths of 128, 192, or 256 bits. The encryption process of the algorithm consists of 4 basic steps; substitute bytes, shift rows, mix columns, and "add round key" which consists of XORing the output of the previous three steps with four words from the key schedule [1]. These operations are repeated multiple times in rounds, with the number of rounds depending on the key size. The decryption process also consists of four steps: inverse shift rows, inverse substitute bytes, add round key, and inverse column mix [1].

  The Initialization Vector (IV) is a random value that is used in combination with a key to provide uniqueness to the encryption process. The IV ensure that the same plaintext encrypted with the same key produces different ciphertexts each time, even if the plaintext remains constant. This means that when a server and client are communicating, they need the same IV to be able to encrypt and decrypt the message. A new IV is generally created between each message or session to increase security.

- **Asymmetric ciphers** is a type of encryption algorithm that uses a pair of keys for the encryption and decryption processes, a public key and a private key. The public key is freely distributed and used for encryption, while the private key is kept secret and used for decryption. The keys do share a mathematical relationship, but deriving the private key from the public key is unrealistic due to time complexity. This encryption method could be compared to a mailbox; anybody can put their mail into a public mailbox (encrypting the message), but only the owner of the mailbox can open and read the mail (decrypting the message). This makes asymmetric ciphers support confidentiality: If someone wants to send you an encrypted message, they can use your public key to encrypt it. Only you, with the corresponding private key, can decrypt and read the message.

- **RSA** is an asymmetric encryption algorithm that, involves the use of a pair of keys, a public key and a private key. RSA differ from other asymmetric encryption algorithm by the mathematical principles, key generation, and specific use cases. It is based on the mathematical difficulty of factoring the product of two large prime numbers, and is commonly used for secure communication, digital signatures, and key exchange.

3 Design and Implementation

The overarching structure of the communication between the server and client is divided into 6 steps, where each step consists of sending or receiving data. This section aims to go through these steps and show how this approach solved the problem of securely transmitting data over the internet.

Before communication begins between the server and client, certain preparations are necessary, including generating the asymmetric keys on the server-side. The server begins by creating a key pair, which can either be new or reused from a previous session. If the server is launched for the first time, it generates a completely new key pair and saves both the private and public keys into separate files. In subsequent executions, the server uses the existing keys by importing them from the previously created files. This process ensures that the server consistently uses the same key pair for each execution.

To start the communication we need to establish an connection between the server and client. This is done with the help of sockets which communicates over a TCP (Transmission Control Protocol) network. On the server side, a socket is created, bound to a specific address and port, and set to listen for incoming connections. This sets up the server to accept connections from clients. On the client side, a socket is created, and a connection is established to the server by specifying the server's hostname and port number. Now the server and client are connected and can begin the communication.

As mentioned above, the process of securely transmitting a file is divided into 6 steps. This figure visualizes the structure of the program, including the 6 different steps. It shows the order of the program and who preform each task.

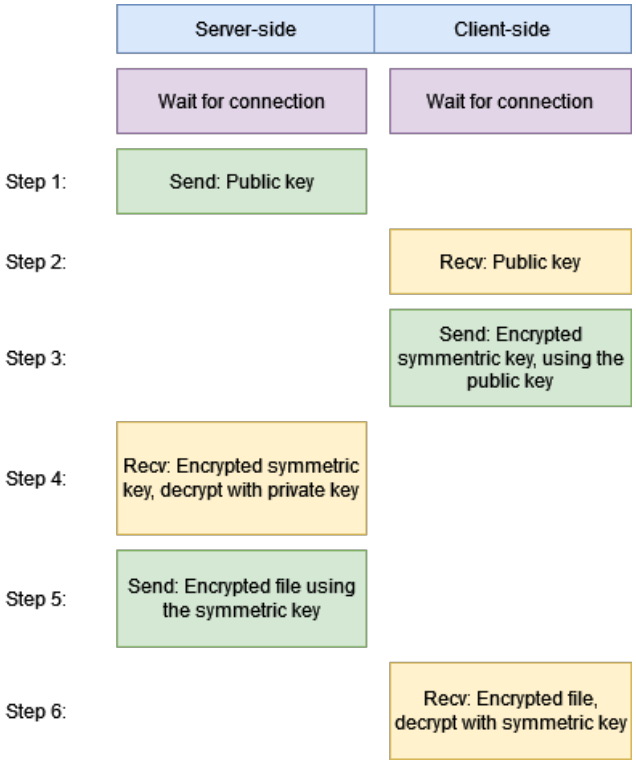

Figure 1: The steps taken when securely transmitting a file.

Before going more into depth of each step, its important to know the logic and algorithm used for the methods to send and receive data.

The data that is **sent** must be in binary form to ensure that everything will be handled equally and there will be no errors when receiving the data. This also makes it possible to create an *header* at the start of the transmitted data, which tells the receiver how large the message is in bytes.

When **receiving** data, the data can be transmitted in an unpredictable amounts of bytes and can not be guaranteed to be received in a single transmission. To ensure the program that it received the full message, it must read the *header* on the first transmission which includes the size of the entire message in bytes. This size is saved and on each subsequent transmission it checks if the requested amounts of bytes has been received. This ensures that the program will always receive the full message and no data will be lost.

Now that sending and receiving data is clear, we can start going more into depth of structure of the communication between the server and client.

Step 1: The server sends its public key to the client. This step does not violate any securities, since the public key is only used for encryption. It is therefore not vital information and can be seen by everyone without causing any issues.

Step 2: The client receives the server's public key and stores it for later use.

Step 3: The client creates a new symmetric key on runtime and encrypts the key using the public key given by the server. The encrypted symmetric key is then sent by the client to the server. The data sent by the client is encrypted and the RSA algorithm ensure that it is impossible for anyone without the private key to extract any information from the encrypted message. Therefore, the system protects the confidentiality and integrity of the data in transit.

Step 4: The server receives the encrypted symmetric key from the client and decrypts it using the private key. Now both the server and client have the same symmetric key and can start sending information efficient and secure.

Step 5: The server opens the file its supposed to send and reads it in *binary mode*, to ensure that each file is treated as a stream of bytes. An encryption cipher is then created using the symmetric key and the file is then encrypted using the cipher. As mentioned in the "Background Theory" section, whenever a symmetric key is used to encrypt a message, it also employs an initialization vector (IV) to give the encryption process a uniqueness. This same initialization vector must also be employed in the decryption process to get the original message, and must therefore be sent together with the encrypted file. The IV is not required to be encrypted and is only inserted at the start of the message. The message is then sent containing the IV and encrypted file.

The system still protects the confidentiality and integrity of the data even though the IV remains in plaintext. Since the file containing the crucial data is encrypted using the AES algorithm which ensure that it is impossible to decrypt an message without the symmetric key, which is kept protected within the server and client.

Step 6: The client begins by opening a file in *write binary mode* to store the data received from the server as a stream of bytes. The client then receives the complete message, which includes both the initialization vector (IV) and the encrypted file. The IV is combined with the symmetric key to create a cipher, and this cipher is then used to decrypt the file. The resulting decrypted file is then written to a new file, indicating the successful transmission of the data.

4 Discussion

The approach described in the above section protects the confidentiality and integrity of data while in transit and effectively solves the given problem. However, there are some issues and choices made in the implementation that could be improve or change the behavior of the application, and this section aims look at these design choices made in the implementation.

To prove that the application works and solves the problem, we could print the different messages the server and client sends and receives during the communication. This will show that the two parts sends and receive the same information, and that it is encrypted and impossible to recognize.

Figure 2: Messages sent/received on the server-side.

Figure 3: Messages sent/received on the client-side.

The figures above shows the messages sent between the server and client during transit, and follows the same order as shown in 1. We see that the firs exchange involves sharing of the public key, which is not required to be encrypted. The second exchange is the encrypted symmetric key. This message itself does not prove much, since the actual symmetric key and the encrypted symmetric key would both look like garbage. The third and last exchange shows the encrypted file, which is clearly not readable. The final message on the client-side shows the decrypted message and proves that the encryption and decryption works. By employing these strategies it will effectively protect the data in transit from attacks such as; eavesdropping, man-in-the-middle attacks, and replay attacks.

In the current implementation the server contains the private and public key, but this is not required. It would be possible to change who sends their public key to the other for encryption. The assumption I made was that it made more sense for the server to send its public key to whoever wants to communicate with it. But in hindsight it makes equally amount of sense for the client to share its public key to whoever it wants to share information with. This can for example be seen when a consumer must share their public key with GitHub to be able to securely communicate between your local machine and the GitHub servers.

Something that could be improved from the current implementation is the handling of larger files. Currently whenever the program reads or writes to a file, it does it all in a single operation, which is not optimal if the

content of the operation is large. It should however, divide the content into smaller more tolerable sizes and preform more read/write operations.

5 Conclusion

This report has given a high-level view of how to implement a network file-transfer system that consist of a server and client. It has also proven that the system is secure and protects the confidentiality and integrity of data while in transit. Some of the issues and decisions made in the implementation has been discussed and given a suggestion to how one could improve it.

References

[1] Avinash Kak. Lecture Notes on "Computer and Network Security" Retrieved 18:45, April 29, 2019, from `https://en.wikipedia.org/w/index.php?title=Page_replacement_algorithm&oldid=886600514`