PKI-Based Chat Application

Group 12

Trentyn Beekman: 70057533
Dylan Bonner: 61315101
Reggie Schriner: 02200719

Introduction

This project implements a secure chat application using Public Key Infrastructure or PKI. The program uses public-key cryptography to encrypt, sign, and verify messages exchanged between a server and a client. The server is also able to securely broadcast that message to multiple other clients. In addition to security, the project aims to provide a pleasing user experience with an easy to use command line interface and helpful error prompts to handle communication with the client when the system is down.

PKI Implementation

The first part of the PKI implementation involves setting up a server and client and making them connect to each other. When the server starts up, it generates a private and a public key using the RSA algorithm from the pycryptodome library. The client will then initiate a connection with the server using the server's IP address and generate its own private and public key using the same algorithm. These are 2048-bit key pairs. Upon successful connection, the server and client will each share their public keys over the secure socket that has been setup. Next, the client readies their plaintext message and the message is signed with their own private key using the PKCS1_15 and SHA256 algorithms, each from the pycryptodome library. After the client signs their message, Advanced Encryption Standard (AES) is used with MODE_EAX to encrypt the message and the signature. With each message that is sent, a new AES session key gets generated. These session keys are then encrypted with RSA using the PKCS1_OAEP algorithm and the server's public key to create a hybrid encryption method. Once the encryption is complete, the client sends the RSA encrypted session key and the AES encrypted signature and message to the server.

On the server side, after it receives the 4 bytes to determine the length of the message, the encrypted message is received. The AES key is then decrypted using PKCS1_OAEP and the message is decrypted using the AES algorithm with MODE_EAX. The signature is then verified using the PKCS1_15 and SHA256 algorithms with the client's public key. As will be explained in the next section, this message is then re-encrypted and broadcasted to other clients.

Socket Programming for Communication

For secure communication, the Transmission Control Protocol (TCP) was used for socket programming. This allows for networked applications to send and receive data over the TCP stream. The server uses the socket library to specify the IP address and TCP sockets are used to listen for incoming connections on port 8080. As an added option, a shutdown server feature was added for safe exit of the server.

To handle broadcasting and connections to and from multiple clients, multithreading is used. A new thread is generated to handle each new client that connects at the same time as another. This way, each client will get its own thread for sending and receiving messages. This is done by using the threading library and getting the name of each client as they connect so they can be differentiated.

Because TCP treats data as a continuous stream, the server needs to know the length of the message. To make this work, the client first sends the length of the encrypted message as a 4-byte integer and then sends the actual encrypted message after that. This way, the server can know the exact length of the message. As stated above, once the message is decrypted by the server, it is then re-encrypted with a new session key so that it can be broadcasted to other clients if they are connected. This session key is generated with AES and then encrypted with the RSA using the PKCS1_OAEP algorithm, the same process that the client uses. This encrypted message is then broadcasted to each client and they decrypt it using the same process the server uses. No signature verification is needed in this broadcast process because the message from the server to the other clients is not signed by the server. The signature from the original client has already been verified by the server and the broadcast clients are expected to trust the server's verification.

Challenges

The first challenge we encountered had to do with decoding the message on the server end. Our initial implementation attempted to decode the message received from the client without decrypting it first, causing an error. This was fixed by decrypting the data received from the client before having the server decode it.The next challenge we ran into was with the threading on the client end. Our threading wasn't working properly which led to only one client being valid at a time. Attempting to make a second client connect with a connection already existing would cause the original connection to be dropped. This was resolved by creating a thread for receiving messages as well as sending messages. By doing this, our application is responsive to input and output messages without any delays, and this correct implementation now allows for multiple clients to connect to the server at once. One last challenge we faced was errors when the server was disconnected while clients were in use. To mitigate this error we added a graceful shutdown feature that allows the server to be turned off even when clients are using it. This feature gives a message to the client that the server was shut down, removing the caused error and informing the client of the shutdown.

Conclusion

Overall, we feel that the project has been completed effectively. Our implementation not only provides a robust and secure method of communication for the users, but also gives a friendly interface to interact with, using names and multiple lines to easily differentiate between messages. Also, we included useful features like a graceful shutdown option for the server, an easy exit option for the user, and output messages for both the server and the clients when either disconnects or shuts down. The program has been crafted to provide a great user experience with tangible security measures, and we are very pleased with its performance.