Implementation of Remote File Storage System

Bakr Alkaanbari
North Central college
Balkanbari@noctrl.edu
Richard Roa
North Central College
rroa@noctlr.rdu

Abstract— For this assignment, we will be looking into using a Raspberry PI and VMware virtual machine to set-up a Bluetooth connection. This system will focus on ensuring we can send data from one location to another while maintaining security for the sender and receiver. The security aspect will ensure that the authenticity and confidentiality of the file will stay safe when the files are transferred. Since python has helped us in previous assignments, we will use it for this assignment because it is fast and efficient to run while also having most of the libraries available to use.

I. Introduction

Last month we looked into a peer-to-peer(P2P) system that used LinkedList, however after some reconsideration we moved on from that system design and focused on a simpler one. For this assignment, we used a Raspberry pi 3 and used VMware as our virtual machine to run the program. We had one set up as a sender and the other as a receiver, both connected via Bluetooth. For this assignment we also focus on the keeping the confidentially and the integrity of the files that were going to be send back and forward

II. System Design

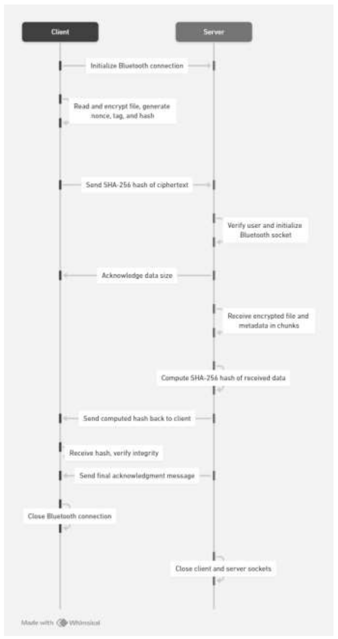

Client
Server
Initialize Bluetooth connection
Read and encrypt file, generate nonce, tag, and hash
Send SHA-256 hash of ciphertext
Verify user and initialize Bluetooth socket
Acknowledge data size
Receive encrypted file and metadata in chunks
Compute SHA-256 hash of received data
Send computed hash back to client
Receive hash, verify integrity
Send final acknowledgment message
Close Bluetooth connection
Close client and server sockets
Made with Whimsical

Figure 1.

This system design utilized two components which are the sender(server) and the receiver. They each have respected roles like what to send and verifying if the data was sent and ensuring that the data was correctly sent back.

A. Client Role

Securely send an encrypted file to a server via Bluetooth, ensuring the data's integrity can be verified by the server without decryption.

Key Functions:

Encryption: Encrypts files using AES in EAX mode which provides confidentiality and a way to check the integrity of the data (via a tag).

Hashing: Generates an SHA-256 hash of the encrypted file to allow integrity verification by the server.

Transmission: Sends the encrypted file, its AES nonce, tag, and the hash to the server over a Bluetooth connection.

Confirmation: Receives a hash from the server for integrity confirmation and an additional acknowledgment that the transfer was successful or not.

B. Server Role

Receive encrypted files over Bluetooth, verify integrity using a hash, and provide acknowledgment back to the client.

Key Functions:

Connection Handling: Waits for a connection from a client on a specified Bluetooth port and accepts incoming connections.

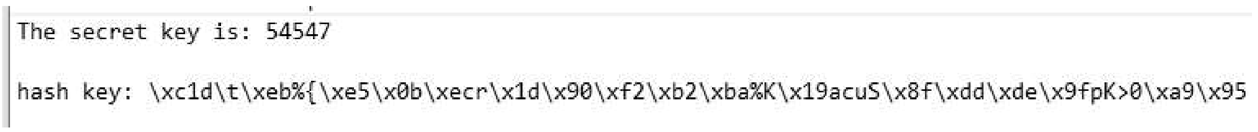
Data Reception: Receives the encrypted file and its associated metadata (nonce, tag) in chunks, ensuring the complete file is received.

Integrity Verification: Calculates a SHA-256 hash of the received encrypted file to verify against the hash sent by the client.

Acknowledgment: Sends the calculated hash back to the client for integrity verification and sends additional confirmation regarding the success of the data transfer.

C. Security measures

For another security measure, we went ahead and implemented a hashing and generated 5 letter passcode code that will write to a secret_key.txt file with the file (Modified_alice.txt) hash key and the code that the users will need to access the program. The secret_key.txt file will also be in a USB file so that any body who wants to get the hash or passcode for the program will need to have access to it if not then they will not be able to run the program.



```
The secret key is: 54547

hash key: \xc1d\t\xeb%{\xe5\x0b\xecr\x1d\x90\xf2\xb2\xba%K\x19acuS\x8f\xdd\xde\x9fpK>0\xa9\x95
```

Figure 1.

III. Implementation

We implemented this system design in python because it was the best and most efficient way to write the program and because we have used this in previous assignments. We used my computer as our virtual machine since it was faster and already had most of the files installed and then used the Raspberry pi as the receiver

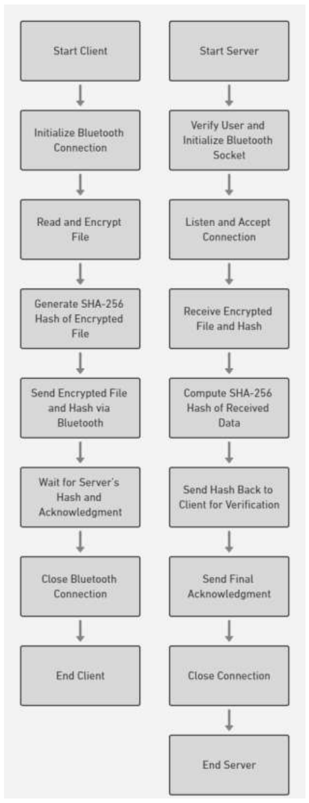

Start Client
Initialize Bluetooth Connection
Read and Encrypt File
Generate SHA-256 Hash of Encrypted File
Send Encrypted File and Hash via Bluetooth
Wait for Server's Hash and Acknowledgment
Close Bluetooth Connection
End Client
Start Server
Verify User and Initialize Bluetooth Socket
Listen and Accept Connection
Receive Encrypted File and Hash
Compute SHA-256 Hash of Received Data
Send Hash Back to Client for Verification
Send Final Acknowledgment
Close Connection
End Server

Figure 2.

A. Set up

Client:

-**Set Up**: Initialize the Bluetooth connection to the server.
-**File Preparation**: Read the file to be sent. Encrypt the file using AES in EAX mode, generating ciphertext, a tag for integrity verification, and a nonce for decryption. Generate a SHA-256 hash of the ciphertext.
-**File Sending**: Send the nonce, tag, and ciphertext over the Bluetooth connection. Send the SHA-256 hash of the ciphertext.
-**Confirmation**: Wait to receive the hash of the ciphertext from the server to verify that the server received the same data as sent. Receive final acknowledgment message from the server about the transfer status.
-**Cleanup**: Close the Bluetooth connection.

Server:

-**Set Up**: Verify the user via a secret key input to start the server. Initialize the server's Bluetooth socket, bind to a port, and listen for connections.
-**Connection Handling**: Accept an incoming Bluetooth connection from the client.
-**Data Reception and Processing**: Receive the size of the incoming data and acknowledge it. Receive the encrypted file in chunks, ensuring all data is received. Compute the SHA-256 hash of the received data.
-**Integrity Verification**: Send the computed hash back to the client.
-**Final Acknowledgment**: Send an additional confirmation message based on the success of the integrity check.
-**Cleanup**: Close the client and server sockets.

B. Authenticity and Confidentiality

To ensure we have authenticity and confidentiality when sending over the files, we first decided to hash the alice.txt file. Once we have the hash, we then send the file over to the receiver where it will hash the file and test if the hash is stored within the secret_key.txt file

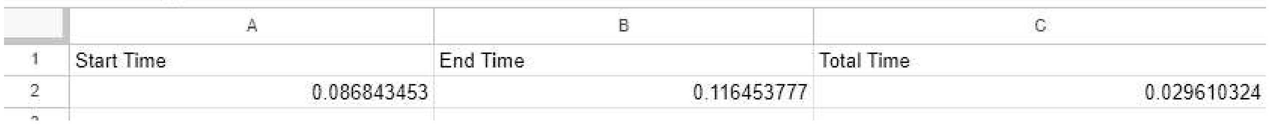
C. Run the Code

To ensure we had consistency within the program we re-ran the code multiple times to see if the code was working. We also switched the sender and receiver to ensure that both the Raspberry PI and VMware did not have any connection issues.

IV. Testing approach

D. Timing

During the testing, we ensure to include the processing time to see how long it will take the program to run and the difference between the shorter files and the longer files. The time was not affected as much because we were able to send the data with different types of bytes which made sending each block quicker. After some testing, we began to have technical issues such as “Data Recv Failure: [Errno] 14 Bad Address”. So really could not run enough tests to see the different time. For the timing we used the performance counter because it helps us accurately measure how long it took to the run the code



	A	B	C
1	Start Time	End Time	Total Time
2	0.086843453	0.116453777	0.029610324

Figure 3.

V. Results

After multiple runs, we managed to successfully connect both devices via Bluetooth, however when it came time to transfer the alice.txt file we ran into problems with where the files would be sent but after a few seconds we would get a runtime error. Since this issue kept persisting, we went ahead and modified the alice.txt file that will hold just the first chapter. Once we managed to get the code running, we were able to run the code properly.

A. connectivity

For figure 2, we can see how the devices connected with one another efficiently. This was important because with the Raspberry PI being a bit slow when it came to connecting and receiving data we manage to send and receive the data back.

B. Hashing

The data that was sent was getting hash, however we noticed that every time we were sending the file back and forth, we were getting different hash key even if the file was not edited. This prove to be a problem because if the sender would like to send a correct file, then the code will think it is wrong and affect the receiver by informing them inaccurate information

VI. Summary

This system establishes a secure channel for file transfer between a client and a server using Bluetooth. It leverages AES encryption for data confidentiality and integrity, and SHA-256 hashing for additional integrity verification. The client ensures the file is sent securely and receives confirmation of the file's integrity, while the server ensures that it only accepts data that matches the integrity checks, providing a robust method for secure file transfers.

VII. Conclusion

In Conclusion the connectivity between devices were working correctly and since most of the time the devices were nearby, we experience little to no connectivity issues. Security proves to be in the right direction but since that part of the code was not hashing correctly, the receiver was not able to ensure that the file was the correct one. For the sending file being a bit long we were having issues sending the full file so in the future we can increase the sending bytes by 2048 or 4096 instead of the base 1024. Our remote file storage system has shown an instance of growth but because of some small issues it can have a fatal flaw that can affect both receiver and sender.