Gabe Chavez and Hadi Fawad

Computer Security

CS-4173-995

Project Report: Secure Messaging Using AES-256

1.1 Introduction

Since the dawn of the information age, security has become an evolving field that safeguards data and its confidentiality. The reliance on digital messaging has been increasing and with that, privacy concerns have to. This Secure Messaging app aims to properly keep user information and messages confidential by encrypting the message when the message is sent then decrypting it when it is received by the other user.

1.2 Encryption algorithm

The main encryption/decryption mechanism is AES-256. It is a standard encryption algorithm used throughout the globe. The key length is 32 bytes or 256 bits. Each block of data is encrypted and decrypted in multiple rounds of manipulation such as shifting rows, substituting bytes, etc. so overall AES-256 is highly secure and will be not feasible to break into. It is computationally efficient allowing the messaging to still be fluid and not be delayed. AES-256 is paired with GCM mode to ensure the integrity of the data and add another layer of encryption.

1.3 Key derivation and management

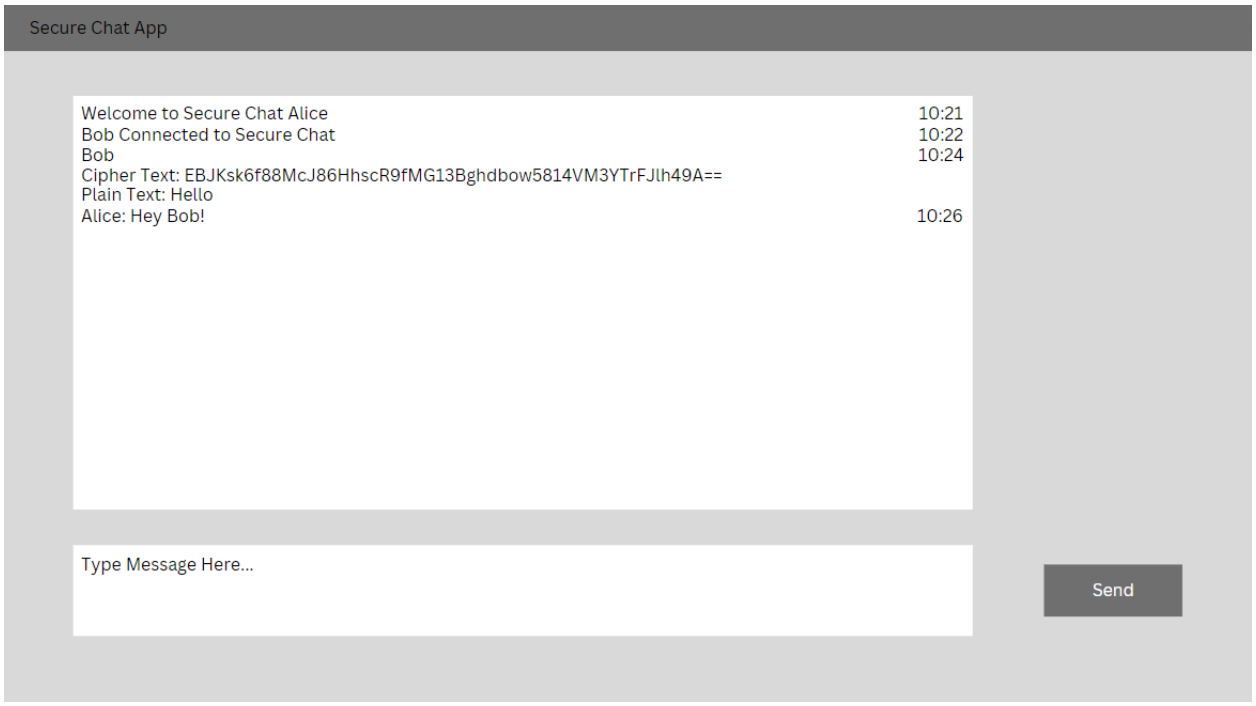
To derive the keys for Alice and Bob, we use PBKDF2 (Password-Based Key Derivation Function 2). PBKDF2 takes in multiple inputs such as the user's password, the salt and iterations or rounds the function will perform. This allows for customizable security where the creator can balance security and computational efficiency. PBKDF2 is an effective way to brute force attacks as it hashes multiple times which will make it not cost-effective to use trial-and-error methods. The key management system will create the keys, store them, and change them. The keys will change periodically after each hour or when the user logs in. This protocol will increase security greatly as an attacker has to solve for the key after every hour period to gain access to the messages which with the encryption layers in place would make this system infeasible to break into.

1.4 Language and technology stack

This app is written in python. With python, it is much simpler to set up an effective app as configuration to run an application is limited. For our cryptography needs like hashing, we used a pycryptodome. The app itself is a python app. If the app were to be upgraded, it would be made with either next.js or react.js.

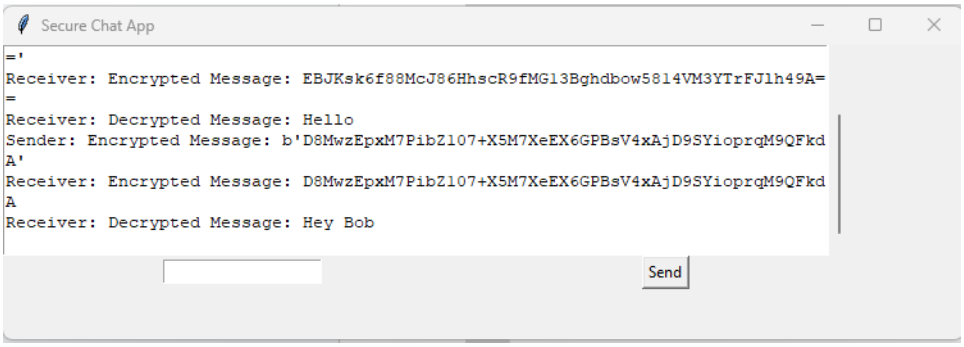
1.5 Application mock-up

The application is a messaging app with one server where two clients can connect. The application will be light in terms of graphics but simple and usable as the functionality will be much more complex.



2.1 First prototype

The first iteration of the application was not complete but it had the key functionalities. It would not have the capabilities for multiple users but it had the encryption algorithm and key management system in place. It would display both the ciphertext and plaintext message when a message was sent.



As shown, any message sent would show the sender's ciphertext, then the receiver ciphertext and plaintext. The key would change every hour or when the user logged in. If the user sent in a phrase such as "hello" twice, the ciphertext would be different for both even though the plaintext is the same. This is important for security as it will make it

harder for an attacker to find patterns within a hashing system which could possibly make brute forcing more effective.

2.2 Code features

Key management class from key_managment.py

```python
# Constants
KEY_UPDATE_INTERVAL = 3600  # Key update interval in seconds (e.g., 3600 seconds = 1 hour)
SALT_SIZE = 16  # Size of the salt


class KeyManager:
    def __init__(self, initial_password):
        self.password = initial_password
        self.key = derive_key(self.password)  # derive_key now only takes the password

    def update_key(self, new_password=None):
        """Update the encryption key."""
        if new_password:
            self.password = new_password
        self.key = derive_key(self.password)
        self.last_update_time = time.time()
        print("Key updated.")

    def check_key_update(self):
        """Check if it's time to update the key."""
        current_time = time.time()
        if current_time - self.last_update_time > KEY_UPDATE_INTERVAL:
            self.update_key()
```

The key manager has a simple initialize, update and check structure to effectively manage the keys. It will update the key every 3600 seconds (1 hour) or update it if a new password is input like if a user were to log in or log back in.

Snippet from networking.py

```python
def send_message_to_client(client_socket, message):
    try:
        # Encrypt the message before sending
        encrypted_message = encryption.encrypt_message(message, server_key)
        client_socket.sendall(encrypted_message)
    except Exception as e:
        print(f"An error occurred while sending a message: {e}")
```

This is an example of one of the systems working in the server system for the users. It uses socket programming to connect clients to the server and to send data to each client. The snippet is the function to send a message to the other client. It will encrypt the message using the key and the message itself then send it to the client where it will decipher it to display the plaintext along with the ciphertext.

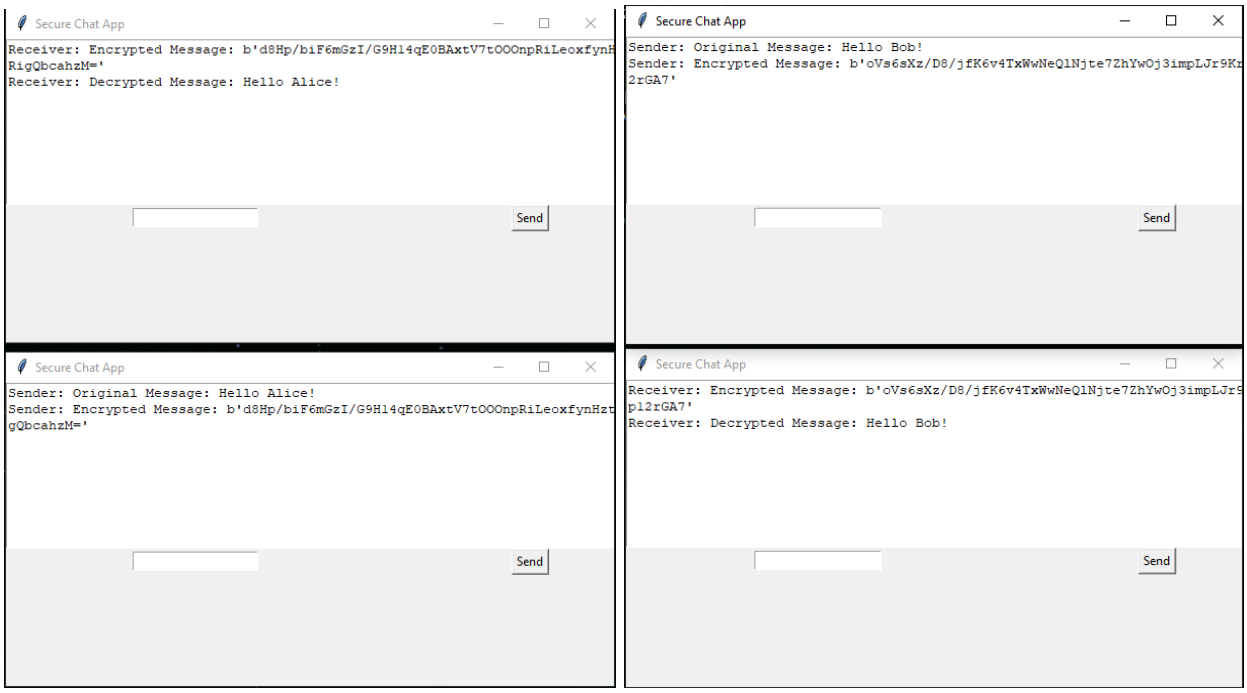
App GUI from gui.py

```
class ChatApp(tk.Tk):
    def __init__(self, host, port, key_manager):
        super().__init__()
        self.title("Secure Chat App")
        self.geometry("600x300")

        self.host = host
        self.port = port
        self.key_manager = key_manager

        # Message display area
        self.messages = scrolledtext.ScrolledText(self, state='disabled', height=10)
        self.messages.grid(row=0, column=0, columnspan=2)

        # Text input
        self.input_user = tk.StringVar()
        self.input_field = tk.Entry(self, text=self.input_user)
        self.input_field.grid(row=1, column=0)

        # Send button
        self.send_button = tk.Button(self, text="Send", command=self.send_message)
        self.send_button.grid(row=1, column=1)

        # Start a thread for the server to listen to incoming messages
        self.server_thread = Thread(target=self.run_server)
        self.server_thread.start()

        self.client_socket = networking.start_client(host, port, self.on_message_received)
```

This is a majority of the GUI design, its implementation is simple and it gets the job done effectively and is easy to understand. Python has an easier to configure and utilize GUI system at the cost of design capabilities in a modern GUI library like react.js. This made testing and reconfiguring a fast process as it was not a conventionally long process to change the GUI. It has direct access to the back-end and is not like a webpage with viewable html which is more secure for our purposes.

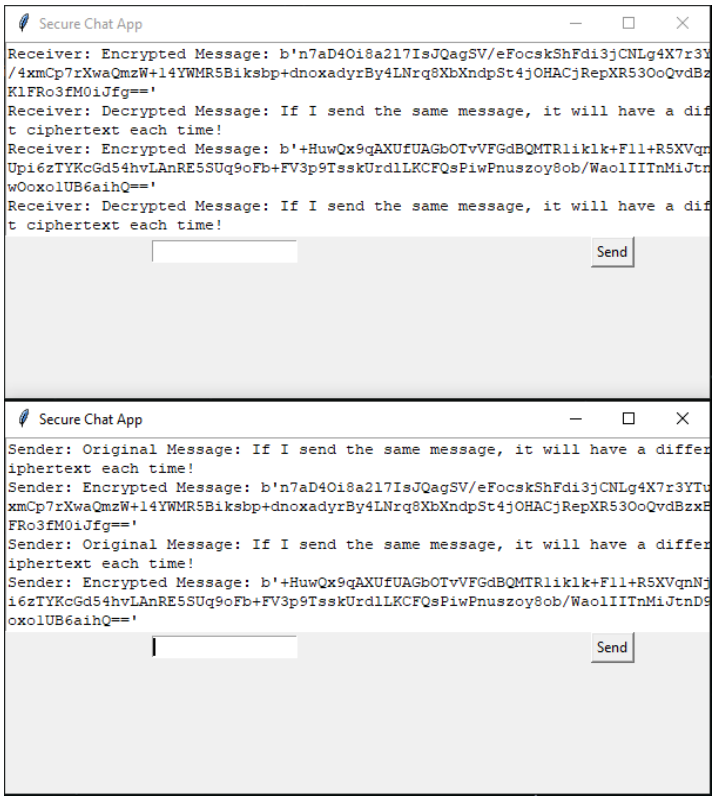
2.3 Secure Chat Demo

Basic Functionality



This snippet shows the basic functionality of our app from both clients' sides. The left and right images are two separate instances of our program running. The sender will send a message such as “Hello Alice!” and then will send their original sent message along with the encrypted message that the encryption algorithm creates. The receiver will receive the encrypted message and the decrypted message. The separate images are to show that the send and receive functionalities work for each user.

Snippet of the same message being sent twice



This is an example of the ciphertext changing if the same message was sent multiple times. The ciphertext changing adds security measures which resist brute force attacks as it makes it harder for an attacker to identify any patterns.

2.4 Conclusion

This project aimed to make a secure messaging app for two users to send messages to each other using encryption with a GUI. The results show we have achieved these main points. The program's security measures used AES-256, GCM Mode and PBKDF2 to properly encrypt and decrypt each user's messages. The key management system employs a key changing feature to change the key every hour or when a user logs in to mitigate attacks. The program uses socket programming to connect both clients to the server. We were limited by resources and time but we allocated them properly to meet all objectives. Next steps for this project would be to make a more appealing GUI and a more complex messaging app where you can have group chats and message different users with multiple sessions running at once. Overall it is necessary to add security features when manipulating, moving and storing user data. With the access of libraries and the internet, it is easier to add them into your application, project, etc. and add varying complexity. This project shows how important it is to have security measures and a variant of them. There are numerous methods for any application. It is critical to combine theory and practical implementation to be more proactive in our strides to better protect data and information from threats.