COE817 Project Report: Secure Banking System

April 12th, 2025

Austin Wort, Jahmil Ally
Dr. Truman Yang
Safwan Hasan

Introduction

The purpose of this project is to design and develop a secure banking system that simulates the functionality of an ATM service. The emphasis of this project is on implementing the required security protocols necessary for protecting the sensitive information ATMs would handle, such as balance, account numbers, passwords, etc.

The functionality of the ATM service itself allows the user to make deposits, withdrawals, and balance inquiries from a dedicated bank server. The bank server is designed to listen for incoming connection requests via sockets. Upon receiving a connection request from an ATM client, the server initiates a dedicated thread to authenticate the client and process the customer's banking transactions securely. Additionally, each client is assigned a single, unique port to ensure the proper distribution of clients and no overlapping sockets.

Key Functionalities:

- Account registration: users can register with the ATM service with a username and password which is stored server-side
- ATM Login: each ATM prompts users for their credentials for secure login and authentication
- Secure Authentication and Key Exchange:
  - A key distribution protocol is employed once a verified user logs in to the ATM service
    - A symmetric key is shared between the client and server, and a session key named 'Master Secret' is additionally established
- Key Derivation: from the Master Secret key, two additional keys are derived
  - Encryption Key: ensures confidentiality of data; Used to encrypt messages between client and server
  - MAC Key (Message Authentication Code): ensures integrity of messages and authenticity of exchanged messages

Secure Transactions:

- The ATM system allows users to perform:
  - Deposits
  - Withdrawals
  - Balance Inquiries
- Each transaction is encrypted and authenticated using the derived keys
- A MAC is generated and verified for every transaction to prevent message tampering; If the server receives a MAC from the client that doesn't match the expected value, the message was tampered with
- The bank server logs the actions of all customers in an encrypted file called the audit log to provide a trail of activities
  - Each activity in the audit log has an attached timestamp

Graphical User Interface (GUI):
The system must include a user-friendly GUI for clients to interact with to access the ATM service. Additionally, the bank server must provide a UI that displays the actions and requests it is processing/has processed. The GUI

Design - Backend

To implement the Secure Banking ATM System we developed two Python scripts that handle the functionality of the bank server and the ATM client, named 'bank_server.py' and 'atm_client.py' respectively. The primary chosen libraries and softwares to implement the necessary security protocols and features of this system are as follows:

- HKDF (from cryptography.hazmat.primitives.kdf.hkdf)
  - Hash-based Key Derivation Function
- SHA256 (from cryptography.hazmat.primitives.hashes)
- HMAC
- Fernet (from cryptography.fernet)
  - Used for symmetric encryption
- Bcrypt
  - Used for password hashing
- Flask
  - Used for passing data between backend and frontend

Bank Server
- Listens for connections on a predefined port
- Handles signup, login, and secure transactions using threads
- Maintains encrypted and plaintext audit logs
- Uses Bcrypt for password hashing and Fernet for secure message exchange via symmetric encryption

Security Layer
- Passwords are hashed using Bcrypt
- Users are authenticated through key exchange using a pre-shared key and a randomly generated Master Secret key
- Keys are derived from Master Secret using HKDF
  - Encryption key (32 bytes)
  - MAC Key (32 bytes) used with HMAC-SHA256

Audit Logging

- All transactions (deposit, withdrawal, balance inquiry) are logged into both a plaintext file (for verification), and an encrypted file.
    - Plaintext Log: audit_log_unencrypted.txt for debugging
    - Encrypted Log: audit_log.enc using Fernet to ensure confidentiality

Design - Frontend

The other Python file, 'atm_client.py', handles all of the frontend page routing and contains all of the API endpoints for Flask to interact with the frontend. All of the user's actions, such as navigating to a different page, or completing a transaction with the ATM service is routed by the atm_client file and will call necessary functions from the bank server to execute any operations.

There are three pages that users can navigate between: action.html, login.html, and signup.html. Respectively, these pages are to handle any bank transactions, user login, and user signup. No special framework was selected for the frontend portion of this project as it would be unnecessarily complicated, and so the frontend consists of simple HTML pages styled using CSS.

ATM Client

- GUI for users to sign up, log in, and perform banking transactions
- Maintains persistent society connections to the server during sessions, and increments assigned port numbers for different users
- Handles encryption, MAC generations, and decryption of responses

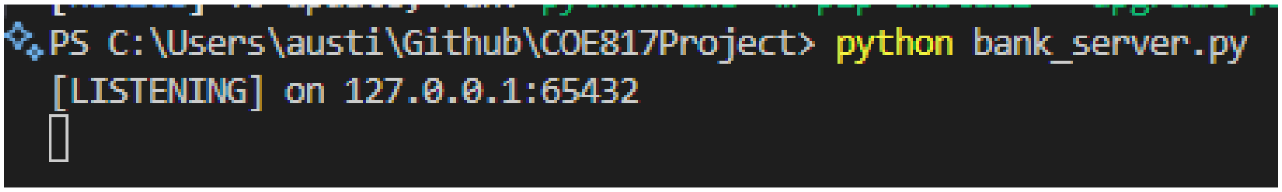
User Interaction Pipeline

- User either logs in to an existing account, or chooses to create a new account
- Upon a successful login, the server generates a random 32-byte Master Secret key
- The Master Secret is securely sent to the authenticated client
- The server and client both use HKDF with SHA-256 to derive 64 bytes of cryptographics material; The first 32 bytes are used as the encryption key, and the remaining 32 are used as the MAC key
- User completes some transaction, and sends the transaction as a message to the server
- Server checks to see that the message contains the correct MAC to ensure it's integrity, before executing the respective transaction and logging it in the audit logs
- GUI updates to show new the user's new balance or requested information

The above protocol ensures mutual authentication between the client and server through the use of verified credentials via the Master Secret key exchange. Additionally, it ensures secure derivation of sessions keys from Master Secret to avoid reusing keys for security.

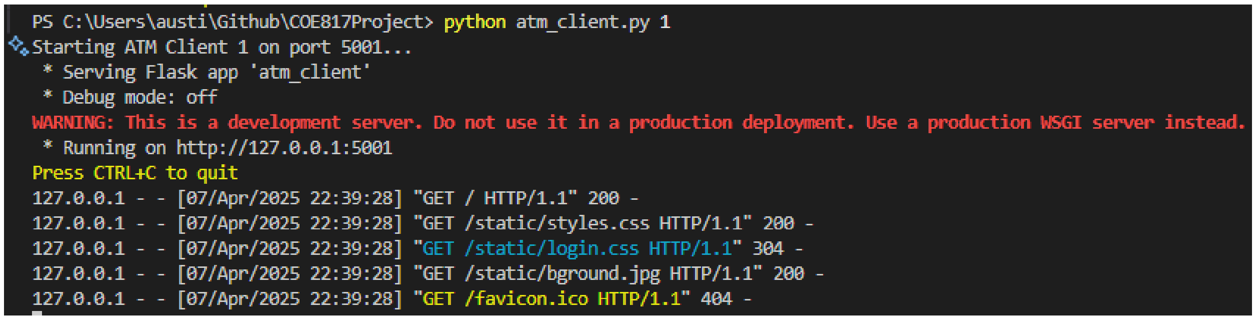
Results

The implementation of the Secure Banking ATM System was successful using the described approach, and ensured all of the necessary security and encryption protocols were executed to ensure the secure transfer of data between the backend and frontend.



```
PS C:\Users\austi\Github\COE817Project> python bank_server.py
[LISTENING] on 127.0.0.1:65432
```

Figure 1: Bank Server Startup and Listening on Port 65432



```
PS C:\Users\austi\Github\COE817Project> python atm_client.py 1
Starting ATM Client 1 on port 5001...
 * Serving Flask app 'atm_client'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5001
Press CTRL+C to quit
127.0.0.1 - - [07/Apr/2025 22:39:28] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [07/Apr/2025 22:39:28] "GET /static/styles.css HTTP/1.1" 200 -
127.0.0.1 - - [07/Apr/2025 22:39:28] "GET /static/login.css HTTP/1.1" 304 -
127.0.0.1 - - [07/Apr/2025 22:39:28] "GET /static/bground.jpg HTTP/1.1" 200 -
127.0.0.1 - - [07/Apr/2025 22:39:28] "GET /favicon.ico HTTP/1.1" 404 -
```

Figure 2: ATM Client 1 Connecting to Bank Server

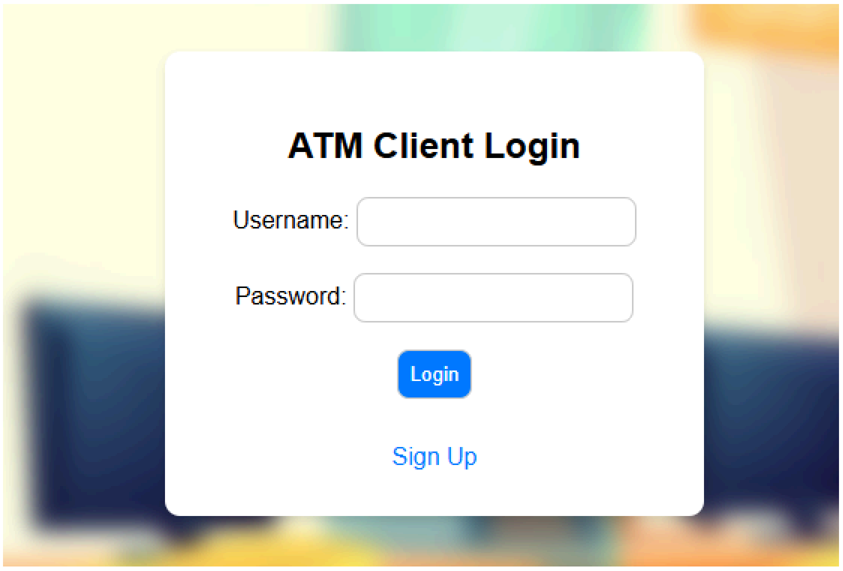
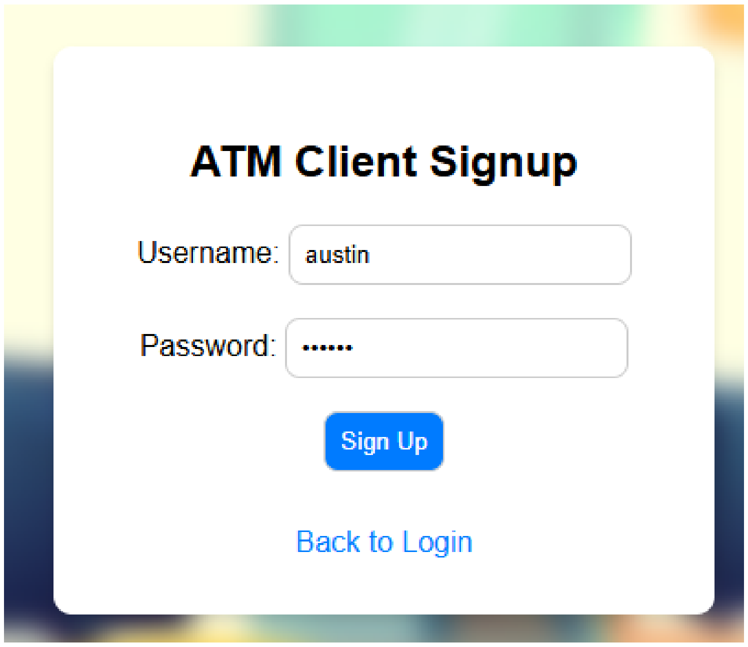


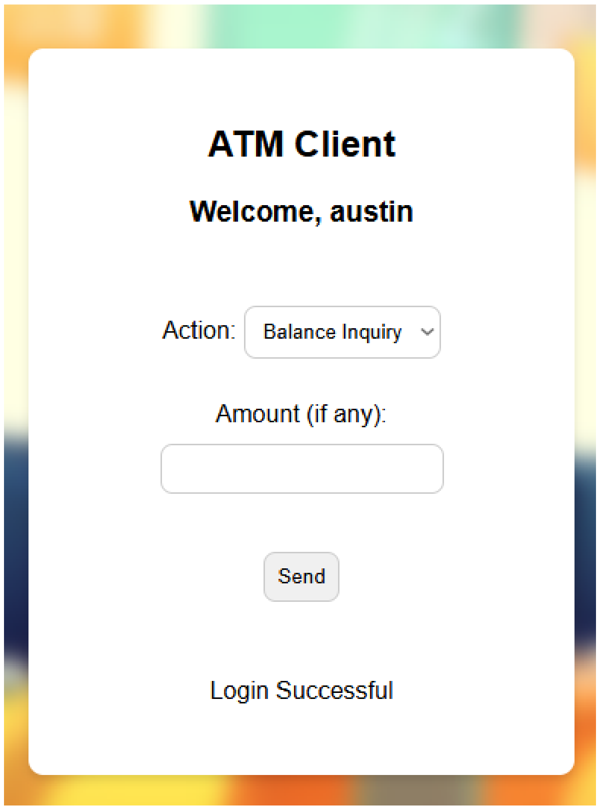
Figure 3: User Login/Landing Page


ATM Client Signup

Username: austin

Password: ••••••

Figure 4: Client Signup Page


ATM Client

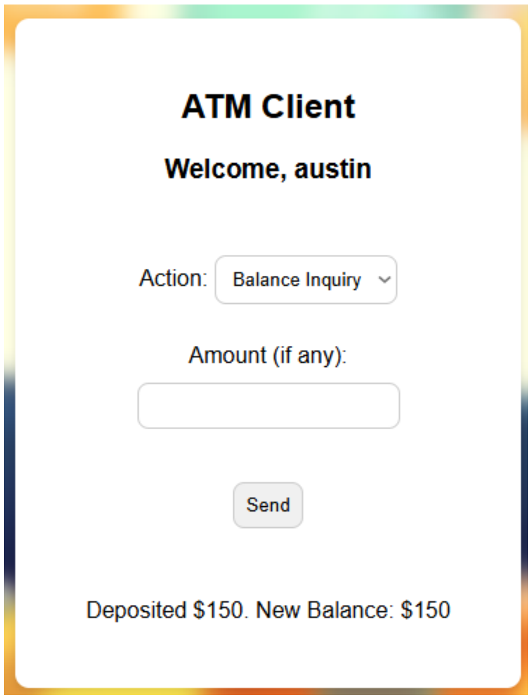
Welcome, austin

Action: Balance Inquiry

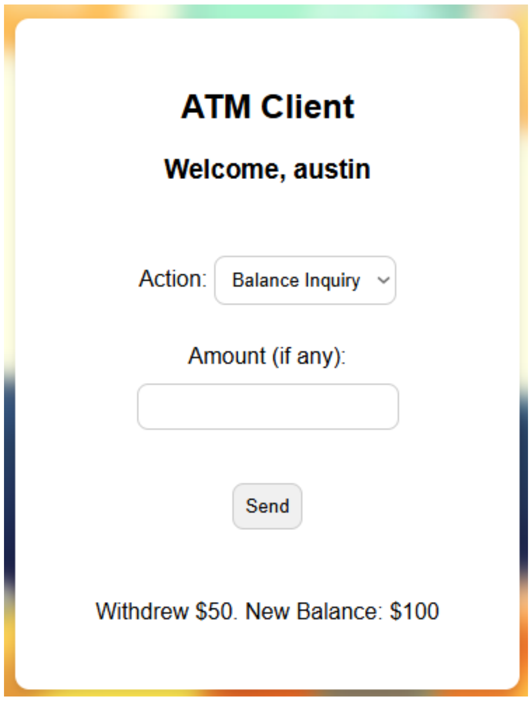
Amount (if any):

Login Successful

Figure 5: User Transaction Page

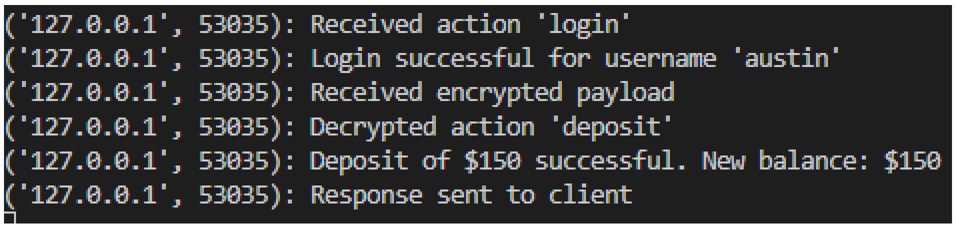


Deposited $150. New Balance: $150



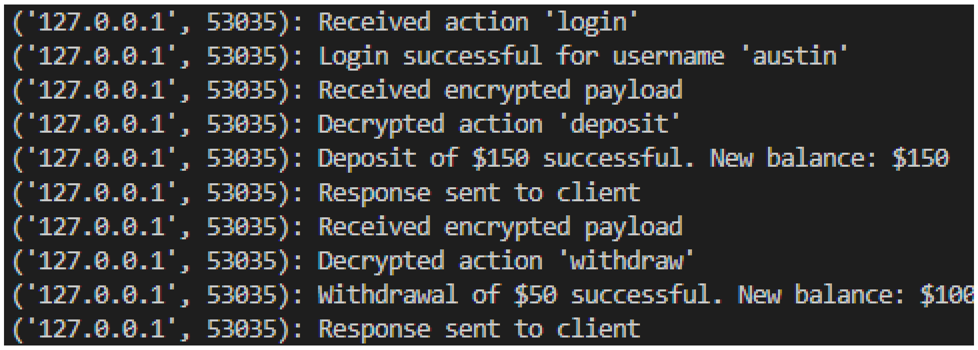
Withdrew $50. New Balance: $100

Figure 6-7: Successful Deposit of $150 and Withdrawal of $50



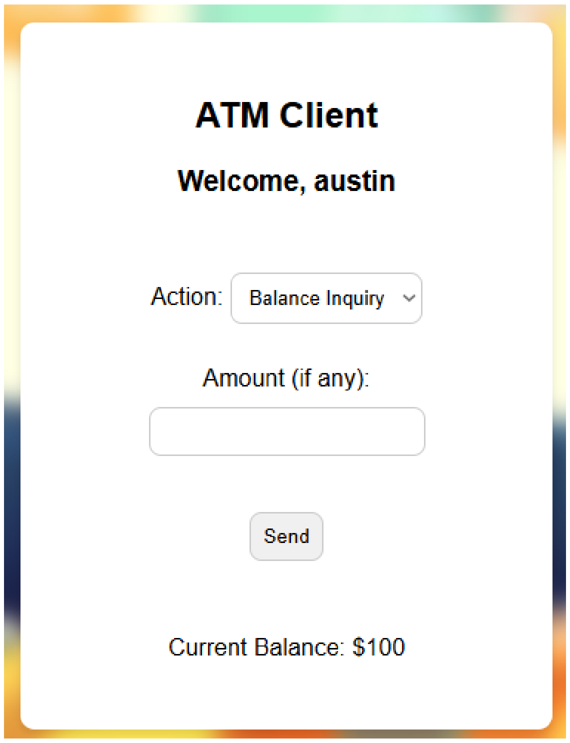
```
('127.0.0.1', 53035): Received action 'login'
('127.0.0.1', 53035): Login successful for username 'austin'
('127.0.0.1', 53035): Received encrypted payload
('127.0.0.1', 53035): Decrypted action 'deposit'
('127.0.0.1', 53035): Deposit of $150 successful. New balance: $150
('127.0.0.1', 53035): Response sent to client
```

Figure 7: Server Terminal Logs of Deposit Transaction



```
('127.0.0.1', 53035): Received action 'login'
('127.0.0.1', 53035): Login successful for username 'austin'
('127.0.0.1', 53035): Received encrypted payload
('127.0.0.1', 53035): Decrypted action 'deposit'
('127.0.0.1', 53035): Deposit of $150 successful. New balance: $150
('127.0.0.1', 53035): Response sent to client
('127.0.0.1', 53035): Received encrypted payload
('127.0.0.1', 53035): Decrypted action 'withdraw'
('127.0.0.1', 53035): Withdrawal of $50 successful. New balance: $100
('127.0.0.1', 53035): Response sent to client
```

Figure 8: Server Terminal Logs of Withdrawal Transaction



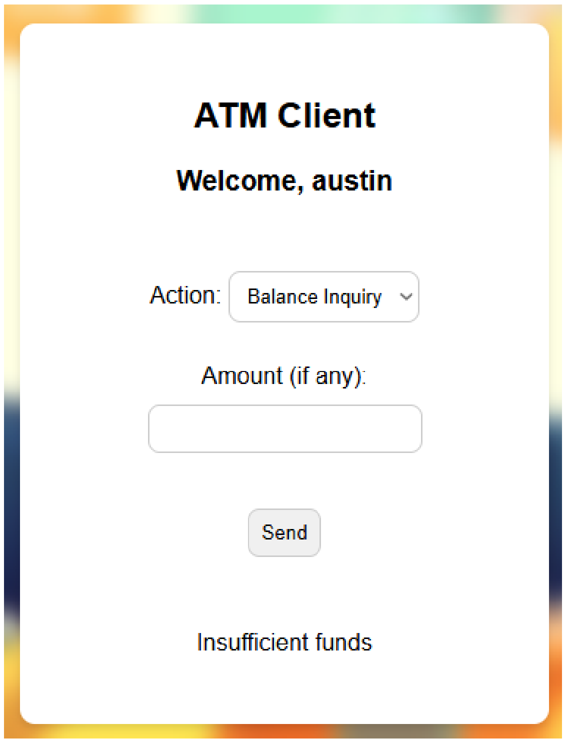
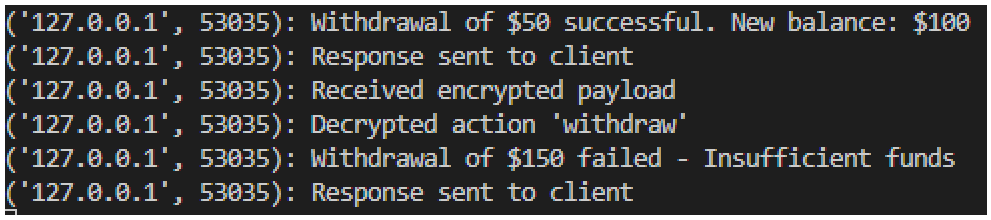
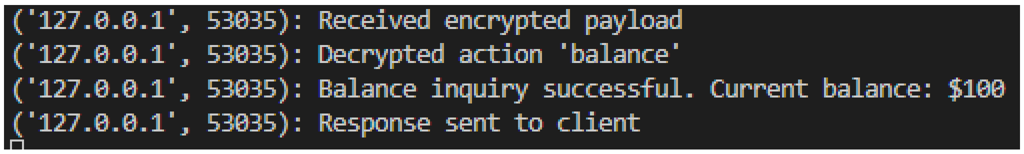


Figure 9-10: Balance Inquiry and Attempted Overdraft of User Account



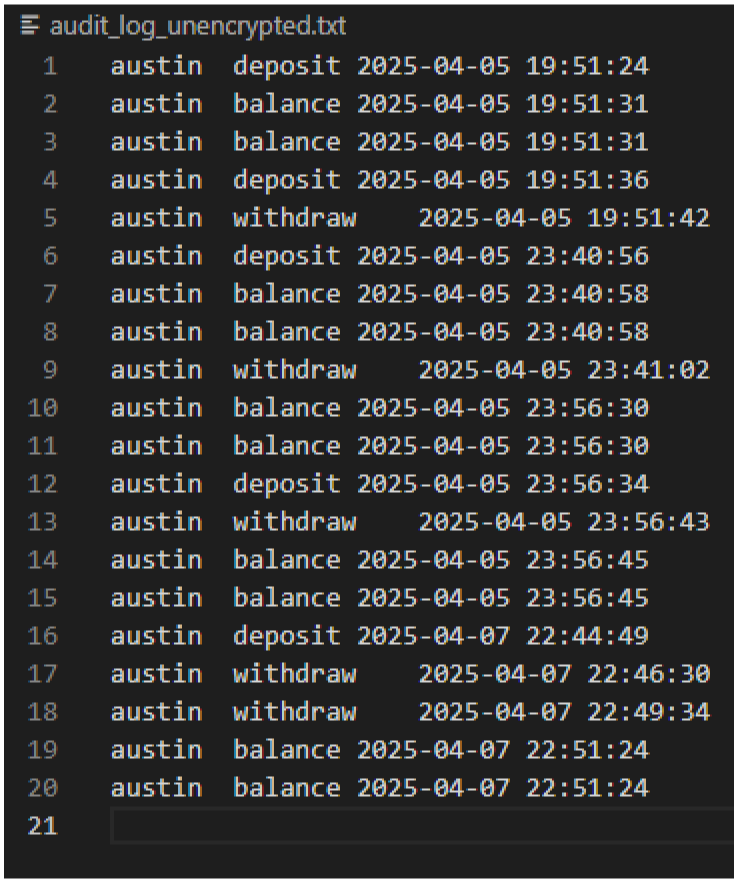
```
('127.0.0.1', 53035): Withdrawal of $50 successful. New balance: $100
('127.0.0.1', 53035): Response sent to client
('127.0.0.1', 53035): Received encrypted payload
('127.0.0.1', 53035): Decrypted action 'withdraw'
('127.0.0.1', 53035): Withdrawal of $150 failed - Insufficient funds
('127.0.0.1', 53035): Response sent to client
```

Figure 11: Server Terminal Logs of Overdraft Transaction



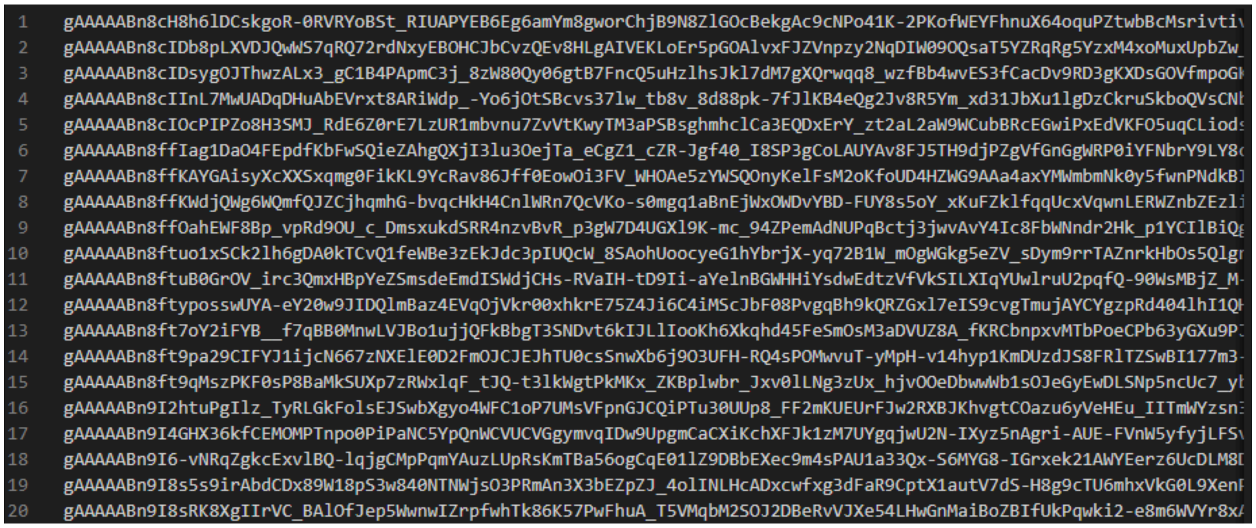
```
('127.0.0.1', 53035): Received encrypted payload
('127.0.0.1', 53035): Decrypted action 'balance'
('127.0.0.1', 53035): Balance inquiry successful. Current balance: $100
('127.0.0.1', 53035): Response sent to client
```

Figure 12: Server Terminal Logs of Balance Inquiry



```
audit_log_unencrypted.txt
austin  deposit 2025-04-05 19:51:24
austin  balance 2025-04-05 19:51:31
austin  balance 2025-04-05 19:51:31
austin  deposit 2025-04-05 19:51:36
austin  withdraw    2025-04-05 19:51:42
austin  deposit 2025-04-05 23:40:56
austin  balance 2025-04-05 23:40:58
austin  balance 2025-04-05 23:40:58
austin  withdraw    2025-04-05 23:41:02
austin  balance 2025-04-05 23:56:30
austin  balance 2025-04-05 23:56:30
austin  deposit 2025-04-05 23:56:34
austin  withdraw    2025-04-05 23:56:43
austin  balance 2025-04-05 23:56:45
austin  balance 2025-04-05 23:56:45
austin  deposit 2025-04-07 22:44:49
austin  withdraw    2025-04-07 22:46:30
austin  withdraw    2025-04-07 22:49:34
austin  balance 2025-04-07 22:51:24
austin  balance 2025-04-07 22:51:24
```

Figure 13: Plaintext Audit Logs of Transactions



```
gAAAAABn8cH8h6lDCskgoR-0RVRYoBSt_RIUAPYEB6Eg6amYm8gworChjB9N8ZlGOcBekgAc9cNPo41K-2PKofWEYFhnuX64oquPZtwbBcMsrivti
gAAAAABn8cIDb8pLXVDJQwWS7qRQ72rdNxyEBOHCJbCvzQEv8HLgAIVEKLoEr5pGOAlvxFJZVnpzy2NqDIW09OQsaT5YZRqRg5YzxM4xoMuxUpbZw_
gAAAAABn8cIDsygOJThwzALx3_gC1B4PApmC3j_8zW80Qy06gtB7FncQ5uHzlhsJkl7dM7gXQrwqq8_wzfBb4wvES3fCacDv9RD3gKXDsGOVfmpoG
gAAAAABn8cIInL7MwUADqDHuAbEVrxt8ARiWdp_-Yo6jOtSBcvs37lw_tb8v_8d88pk-7fJlKB4eQg2Jv8R5Ym_xd31JbXu1lgDzCkruSkboQVsCN
gAAAAABn8cIOcPIPZo8H3SMJ_RdE6Z0rE7LzUR1mbvnu7ZvVtKwyTM3aPSBsghmhclCa3EQDxErY_zt2aL2aW9WCubBRcEGwiPxEdVKFO5uqCLiod
gAAAAABn8ffIag1DaO4FEpdfKbFwSQieZAhgQXjI3lu3OejTa_eCgZ1_cZR-Jgf40_I8SP3gCoLAUYAv8FJ5TH9djPZgVfGnGgWRP0iYFNbrY9LY8
gAAAAABn8ffKAYGAisyXcXXSxqmg0FikKL9YcRav86Jff0EowOi3FV_WHOAe5zYWSQOnyKelFsM2oKfoUD4HZWG9AAa4axYMWmbmNk0y5fwnPNdkB
gAAAAABn8ffKWdjQWg6WQmfQJZCjhqmhG-bvqcHkH4CnlWRn7QcVKo-s0mgq1aBnEjWxOWDvYBD-FUY8s5oY_xKuFZklfqqUcxVqwnLERWZnbZEzl
gAAAAABn8ffOahEWF8Bp_vpRd9OU_c_DmsxukdSRR4nzvBvR_p3gW7D4UGXl9K-mc_94ZPemAdNUPqBctj3jwvAvY4Ic8FbWNndr2Hk_p1YCIlBiQ
gAAAAABn8ftuo1xSCk2lh6gDA0kTCvQ1feWBe3zEkJdc3pIUQcW_8SAohUoocyeG1hYbrjX-yq72B1W_mOgWGkg5eZV_sDym9rrTAZnrkHbOs5Qlg
gAAAAABn8ftuB0GrOV_irc3QmxHBpYeZSmsdeEmdISWdjCHs-RVaIH-tD9Ii-aYelnBGWHHiYsdwEdtzVfVkSILXIqYUwlruU2pqfQ-90WsMBjZ_M-
gAAAAABn8ftyposswUYA-eY20w9JIDQlmBaz4EVqOjVkr00xhkrE75Z4Ji6C4iMScJbF08PvgqBh9kQRZGxl7eIS9cvgTmujAYCYgzpRd404lhI1Q
gAAAAABn8ft7oY2iFYB__f7qBB0MnwLVJBo1ujjQFkBbgT3SNDvt6kIJLlIooKh6Xkqhd45FeSmOsM3aDVUZ8A_fKRCbnpxvMTbPoeCPb63yGXu9P
gAAAAABn8ft9pa29CIFYJ1ijcN667zNXElE0D2FmOJCJEJhTU0csSnwXb6j9O3UFH-RQ4sPOMwvuT-yMpH-v14hyp1KmDUzdJS8FRlTZSwBI177m3-
gAAAAABn8ft9qMszPKF0sP8BaMkSUXp7zRWxlqF_tJQ-t3lkWgtPkMKx_ZKBplwbr_Jxv0lLNg3zUx_hjvOOeDbwwWb1sOJeGyEwDLSNp5ncUc7_y
gAAAAABn9I2htuPgIlz_TyRLGkFolsEJSwbXgyo4WFC1oP7UMsVFpnGJCQiPTu30UUp8_FF2mKUEUrFJw2RXBJKhvgtCOazu6yVeHEu_IITmWYzsn
gAAAAABn9I4GHX36kfCEMOMPTnpo0PiPaNC5YpQnWCVUCVGgymvqIDw9UpgmCaCXiKchXFJk1zM7UYgqjwU2N-IXyz5nAgri-AUE-FVnW5yfyjLFS
gAAAAABn9I6-vNRqZgkcExvlBQ-lqjgCMpPqmYAuzLUpRsKmTBa56ogCqE01lZ9DBbEXec9m4sPAU1a33Qx-S6MYG8-IGrxek21AWYEerz6UcDLM8
gAAAAABn9I8s5s9irAbdCDx89W18pS3w840NTNWjsO3PRmAn3X3bEZpZJ_4olINLHcADxcwfxg3dFaR9CptX1autV7dS-H8g9cTU6mhxVkG0L9Xen
gAAAAABn9I8sRK8XgIIrVC_BAlOfJep5WwnwIZrpfwhTk86K57PwFhuA_T5VMqbM2SOJ2DBeRvVJXe54LHwGnMaiBoZBIfUkPqwki2-e8m6WVYr8x
```

Figure 14: Encrypted Audit Logs of Transactions

Conclusion

The Secure Banking ATM System project successfully implemented a secure client-server architecture to simulate the functionality of an ATM service while emphasizing network security principles. The system consisted of two main components: the ATM client (atm_client.py) and the bank server (bank_server.py). The ATM client provided a user-friendly GUI for users to interact with, allowing them to sign up, log in, and perform secure banking transactions such as deposits, withdrawals, and balance inquiries. The bank server was a multi-threaded application that listened for incoming client connections, authenticated users, and processed transactions securely.

The project employed an authenticated key distribution protocol to ensure mutual authentication between the client and server. Upon successful login, the server generated a unique 32-byte Master Secret key, which was securely shared with the client. Both the client and server used this Master Secret to derive two session keys using the HMAC-based Key Derivation Function (HKDF): an encryption key for ensuring data confidentiality and a MAC key for verifying data integrity. All transaction data was encrypted using the encryption key with the Fernet symmetric encryption scheme, and a Message Authentication Code (MAC) was generated using the MAC key to prevent tampering. The server verified the MAC and decrypted the payload before processing the transaction, ensuring that the data remained secure throughout the communication.

The system supported secure transactions, including deposits, withdrawals, and balance inquiries. Each transaction was logged in both plaintext and encrypted formats, providing a comprehensive audit trail for debugging and verification. The server used threading to handle multiple clients concurrently, ensuring scalability and efficient resource utilization. Each client was assigned a unique port to prevent socket conflicts, and the server logged all actions with the client's address for easy tracking. The project integrated several advanced cryptographic libraries, including bcrypt for password hashing, Fernet for encryption, and HKDF for key derivation, ensuring a high level of security. The GUI, implemented using Flask, provided a seamless user experience, allowing users to navigate between pages for login, signup, and transactions.

The project successfully demonstrated the implementation of secure communication protocols, ensuring confidentiality, integrity, and authenticity of sensitive financial data. It served as a practical application of network security concepts, showcasing the importance of encryption, authentication, and integrity in real-world financial systems. This project not only met its objectives but also provided a strong foundation for further exploration and innovation in the field of cybersecurity.