Toronto Metropolitan University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Network Security
Course Number:	COE817
Semester/Year (e.g.F2016)	W2025

Instructor:	Dr. Truman Yang
TA:	Aman Yadav

Assignment/Lab Number:	Course Project
Assignment/Lab Title:	Secure Banking System

Submission Date:	April 4, 2025
Due Date:	April 12, 2025

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Al-Shalabi	Mohammad	501026067	06	M.A.S
Javaheri	Hamid	501050134	06	H.J
Gill	Simrat	501100893	10	S.S.G
Ali	Syed Ammar	501034707	06	S.A.

Introduction

This project aims to design and implement a secure banking application that connects a secure banking server that should be able to accommodate multiple ATM clients. This project is intended to provide an application that supports the concepts of confidentiality, integrity, and authentication of financial transactions between the client and the bank. This application intends to store user information and defend against potential attacks safely.

To achieve the goals stated above, we utilized an authentication key distribution protocol containing shared keys later used to establish a new symmetric master key between the bank and ATM client. This shared master key is then later used to derive two more keys. One of those keys is used for encrypting the data, and another is used to generate the Message Authentication Codes to maintain the integrity of the server. This is crucial in ensuring that the procedures, such as depositing, withdrawing, and balance inquiries, are not tampered with during transmission. Furthermore, the system will also log all the custom interactions in an encrypted manner to ensure transparency and a trail for everything.

The purpose of this project is to understand and address the growing need for security within banking. With increasing threats to these systems, developing cryptographic algorithms is more important than ever. This project will teach us the principles practically taught in class. It combines concepts learned in both the classroom and labs, such as threading, authentication, and replay attack prevention. We utilize these concepts to offer a solution for secure transaction processing and multithreaded environments.

This project will focus on a couple of basic banking operations, including customer registration, depositing, withdrawal, and balance inquiry, all implemented in a secure manner between the server and ATM clients. This implementation should simulate the real-world banking environment while incorporating a plethora of security features. This, however, is not without certain limitations, such as the assumption that there is an initial shared key between the clients and server. This lab will address threats within the protocol without extending the project scope to include other security concerns, such as large-scale distributed attacks. Furthermore, this is enhanced by creating a GUI and developing multithreaded components to demonstrate how the security protocol works. Overall, the project will serve as a great learning experience for understanding the strengths and weaknesses of this security protocol and the importance of designing a secure system.

Design

Project Implementation Overview

This banking application's implementation uses a client-server architecture and contains various classes that all achieve a specific task.

<u>Client-Side (ATM Client)</u>

This class and package act as the main client controller. This means it is responsible for setting up the graphical user interface, establishing the connection to the bank server, performing the secure key exchange, and dispatching individual commands such as login, registration, and transactions to the server.

1. Important variables

- Server_Address/PORT: This defines the server location and the port number
- Socket, BufferedReader, PrintWriter, and Variables to manage the network over TCP/IP
- JFrame, JTextField, JPasswordField, JTextArea: These are GUI components to collect user information such as credentials and display the status messages
- SecretKey encryptionKey, macKey: Stores the session-specific keys that are established during the key exchange protocol

2. Important Methods

- Constructor (ATMClient()): The constructor initializes the GUI using the setupUI() and immediately attempts to connect to the server using the connectToServer() method
- setupUI(): Configures the main application window using Swing, this includes the input field such as the username password, buttons and helper methods for styling and other things to apply consistent styling across all the components
- performKeyExchange(): This method sends the "KEY_EXCHANGE" command to the server. It then waits for the server response, which should concatenate the encrypted master secret. It then performs Base64 validation and decoding, decrypts the master secret using the pre-shared master key, and calls the key derivation method to store the session's encryption and MAC keys
- loginUser(): This method reads the username and password from the GUI, and ensures that they are not empty, then concatenates and encrypts them using the encryption key. Then, the "LOGIN" command is sent with the encrypted user information. When the server response is received the MAC is verified before determining whether to open the transaction panel or to prompt an error message.

<u>Graphical User Interface</u>

The graphical user interface (GUI) is developed using Swing. It contains various screens for login, registration, account deletion, and transaction operations. This component contains classes such as the RegistrationPage, deleteAccountPage, and TransactionPage, which are all responsible for rendering.

1. RegistrationPage (Package: gui)

- Purpose: Provide the use interface for new account registration
- GUI Components: Text fields for username, password, confirm password, and initial deposit amount, buttons for register and back to login, status are to display the status of the registration
- Workflow: The function registerUser validates that all fields are filled and that the password and confirm passwords match. Then, the input data is concatenated into one big string. This string is encrypted using the session key and generates a MAC using the MAC key. It then sends the register command with the encrypted data to the server.

2. DeleteAccountPage (Package: gui)
- Purpose: Provides the interface for users to delete their accounts
- GUI Components: This GUI class includes input fields for username and password, two buttons, one to trigger the deleteAccount method and another to go back to the previous menu
- Workflow: The deleteAccount() method validates the user input by combining the username and password, encrypting them and generating a MAC. The DELETE command is sent to the server with the encrypted credentials and MAC

3. TransactionPage (Package:gui)
- Purpose: The transaction facilitates the handling of financial transactions such as deposits, withdrawals, and balance inquiries
- GUI Components: This page contains buttons for Deposit, withdrawal, check balance and logout
- Workflow: For transactions that require specific amounts, a prompt launches an area to collect the value. The method itself combines the username transaction amount, encrypts it, and generates a MAC before sending the command with the payload. For balance inquiries, the response is supposed to be in a specific format and is verified by the client.

Server-Side Components

BankServer

This class's purpose is to serve as the central server that listens for connections, manages client sessions, and coordinates user-related operations such as login, registration, account deletion, and transactions.

1. Important Variables
- Database_File: File name where the user information is stored
- Port: A port number that the server listens on
- userDatabase: An in-memory hashmap that is loaded from the file to retrieve user information quickly
- SecretKey, encryptionKey, macKey: Keys which are used for secure communications; generated and stored at the server

2. Important Methods
- main(): Creates an instance of the BankServer and calls the startServer method
- startServer(): Loads the user database by reading the file line by line, then starts the server socket that will listen for client connections. For each connection, the ClientHandler thread is created and started
- generateEncryptionKeys(): This method ensures that the server's encryption and MAC keys are generated. This is done by creating a master secret and serving session keys
- loadUserDatabase(): This method reads the contents of the userDatabase file and populates the hashmap previously mentioned for access during login or registration

ClientHandler
There are two variations of this class, one implemented as a standalone and one as an inner class within the BankServer.

Variant 1 (Inner class in BankServer)
This class handles a single client connection within the context of the BankServer. It is integrated directly with the server instance to easily access the shared keys and user database.

1. Important Variables
- Socket, BufferedReader, PrintWriter: Handles communication with the client
- BankServer server: This is a reference to the main BankServer instantiation to be able to access shared keys and methods
2. Important Methods
- run(): continuously reads the input from the client. This method first checks for the "KEY_EXCHANGE" command and calls the handleKeyExchange() before going into the main loop to process the "LOGIN", "REGISTER", "DELETE_ACCOUNT" commands
- handleKeyExchange(): Checks whether the server keys are present; if not, it generates them. Then the master key is created, the encryption derived and MAC keys, Base64 encodes the master secret, encrypts it with the servers encryption key and send the result of this entire process to the client
- handleLogin(), handleRegistration(), handleAccountDeletion(): These methods work together to do the following, read the encrypted data from the client, decrypt this data using the session encryption key parse the credentials or command details, validate the date and generate a MAC protected response that is then sent back to the client

Variant 2 (Standalone Class)
This class accepts encryption and MAC keys using its constructor.

1. Important Variables
- Socket, BufferedReader, PrintWriter: Handles communication with the client
- BankServer server: This is a reference to the main BankServer instantiation to be able to access shared keys and methods

2. Important Methods
- run(): continuously reads the input from the client. This method first checks for the "KEY_EXCHANGE" command and calls the handleKeyExchange() before going into the main loop to process the "LOGIN", "REGISTER", "DELETE_ACCOUNT" commands
- handleKeyExchange(): Checks whether the server keys are present; if not, it generates them. Then the master key is created, the encryption derived and MAC keys, Base64 encodes the master secret, encrypts it with the severs encryption key and send the result of this entire process to the client

- handleLogin(), handleRegistration(), handleAccountDeletion(): These methods work together to do the following, read the encrypted data from the client, decrypt this data using the session encryption key parse the credentials or command details, validate the date and generate a MAC protected response that is then sent back to the client

<u>Utility Classes</u>

<u>KeyManager (Package: utils)</u>
The purpose of this class to manage key generation and provide helper method to convert key to and from strings

1. Important Variables
- PRE_SHARED_KEY: A string that acts as a pre-shared key for the first steps of key exchange
- masterSecretKey: A secret key used to store the master secret after it is generated

2. Important Methods
- encrypt(String data, String key), decrypt(String encryptedData, String key): These two methods use the implementation of AES discussed in class to encrypt and decrypt data from the given key string. It calls a helper function named generateKeyFromString() that either pads or trims the key to a 16-byte length suitable to be suitable for AES
- generateMasterSecret(): Uses Java's keyGenerator with AES to generate a 128-bit random key, which is then stored in the masterSecretKey
- getEncodedMasterSecret()/getMasterSecretKey()/ setMasterSecretKey(): These methods retrieve the master secret in a format that is compatible such as Base64 format, get the raw SecretKey, or set is based on the encoded string. Then the secure transfer and storage of the master key is done.

<u>EncryptionUtils (package: utils)</u>
The purpose of this class is to provide a low-level cryptographic function for AES and RSA encryption

1. Important Methods
- encryptRSA(String data, PublicKey publicKey) / decryptRSA(String encryptedData, PrivateKey privateKey): This function uses RSA to encrypt and decrypt data
- encryptAES(String data, SecretKey secretKey) / decryptAES(String encryptedData, SecretKey secretKey): Uses AES in CBC mode while also ensuring that the output is Base64 encoded for safe transmission
- encryptAESKey(SecretKey secretKey, PublicKey publicKey) / decryptAESKey(String encryptedKey, PrivateKey privateKey): These two methods are in charge of wrapping or unwrapping a AES key using RSA, which is necessary when trying to transmit a symmetric key

<u>SecurityUtils (Package: Utils)</u>
This class acts as a wrapper that combines the functionality of the two classes outlined above

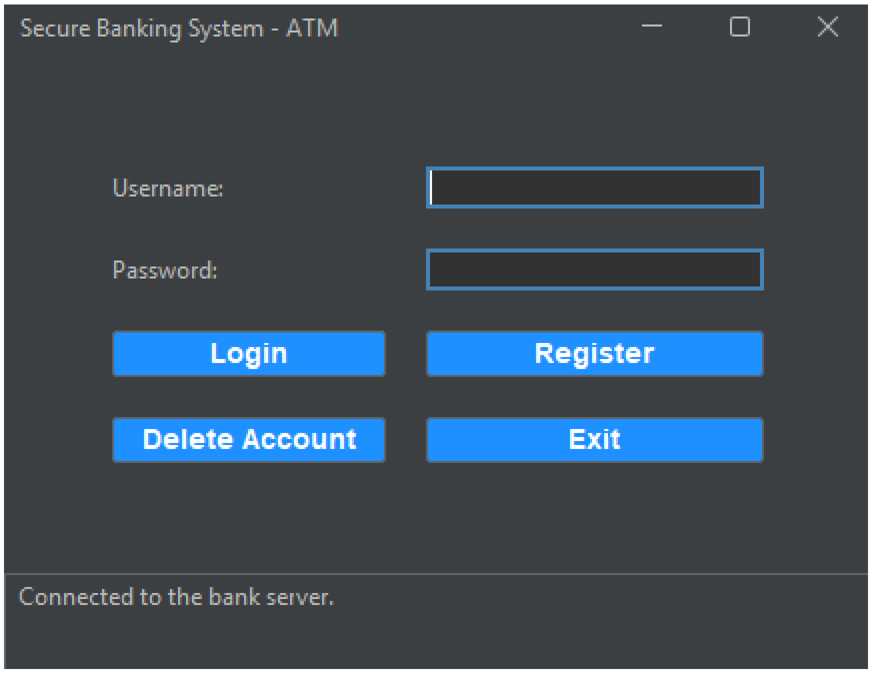
1. Important Methods

- generateAESKey() and deriveKeys(SecretKey masterSecret):this method generates a new AES key and uses the function to split the master secret into two separate keys: one for encryption and the other for MAC generation
- encrypt(String data, SecretKey key) / decrypt(String encryptedData, SecretKey key): Function outlined above
- decryptKey(byte[] decodedEncryptedMasterKey, SecretKey key):Decrypts the encrypted master secret received from the server.
- generateMAC(String data, SecretKey macKey) / verifyMAC(String data, String mac, SecretKey macKey): Implements MAC generation (likely using HMAC with a secure hash function such as SHA-256) and verification, ensuring data integrity for each message.

<u>Process of Events</u>

1. Connection and key exchange:
    a. The ATMclient connects and send the "KEY_EXCHANGE" request
    b. The server generates its encryption and MAC keys and produces a master secret, derives session keys, and sends an encrypted version of the master secret back to the client
    c. The client decrypts the master key and derives a session key which both the client and server now contains
2. Processing User Commands:
    a. For login, registration, deletion, and transaction the client encrypts the payload using the session encryption
    b. A MAC is generated over the encrypted data to ensure integrity
    c. The server decrypts the payload, verifies the MAC, processes the request and sends back a response which is also verified using MAC-protected
3. Responses Verification and Action:
    a. The client will receive a response and verify the MAC to confirm authenticity, and then proceeds with the corresponding UI update

<u>Results</u>



Connected to the bank server.

Figure 1: GUI Startup

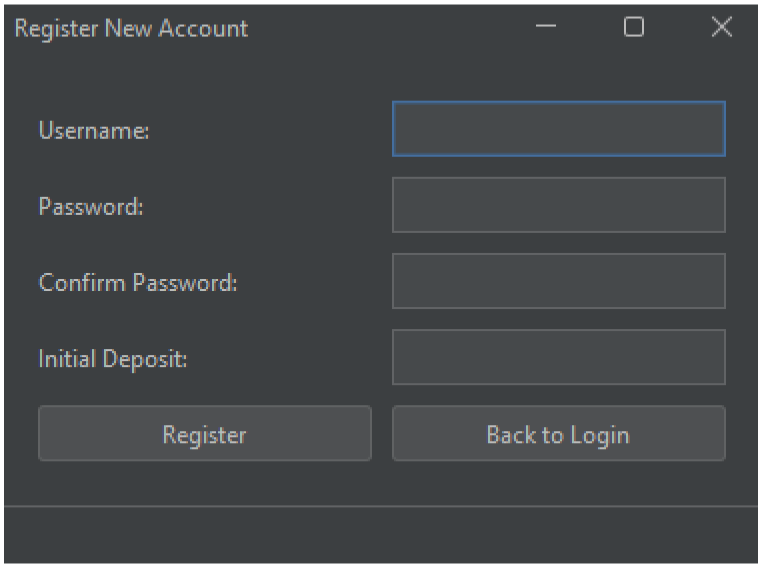
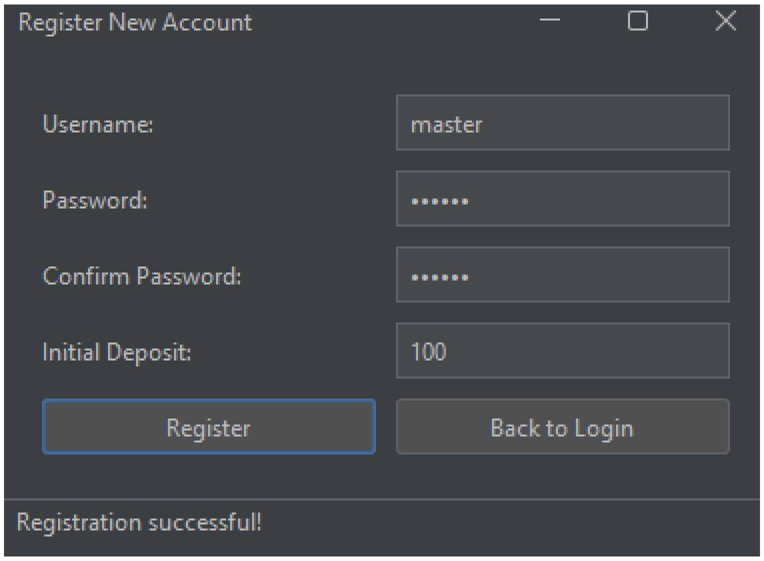


Figure 2: Account Registration


Register New Account

Username: master

Password: ••••••

Confirm Password: ••••••

Initial Deposit: 100

Registration successful!

Figure 3: Successful Registration

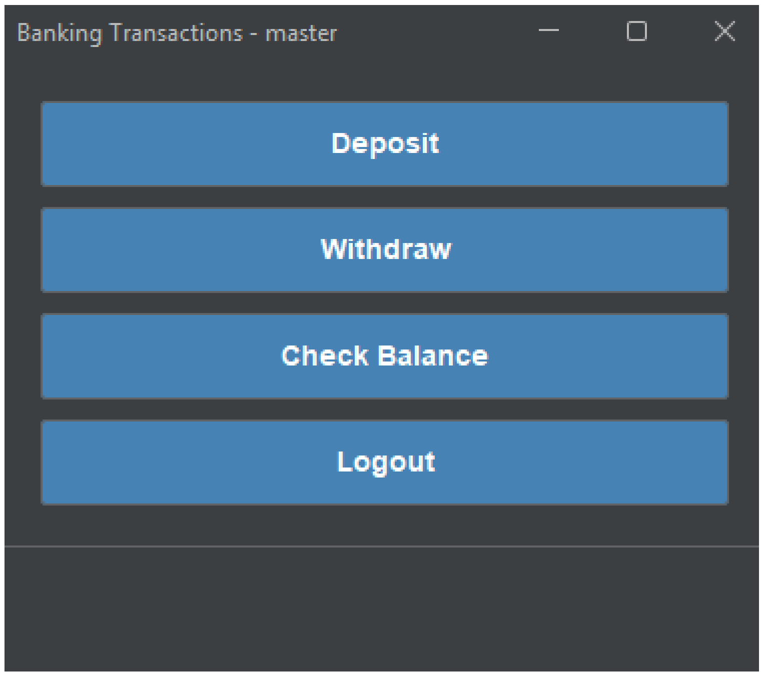
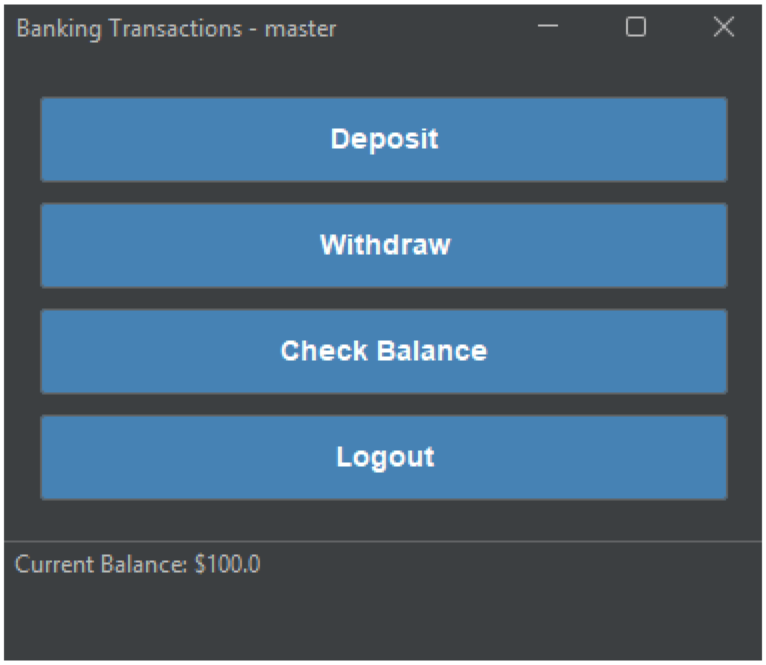
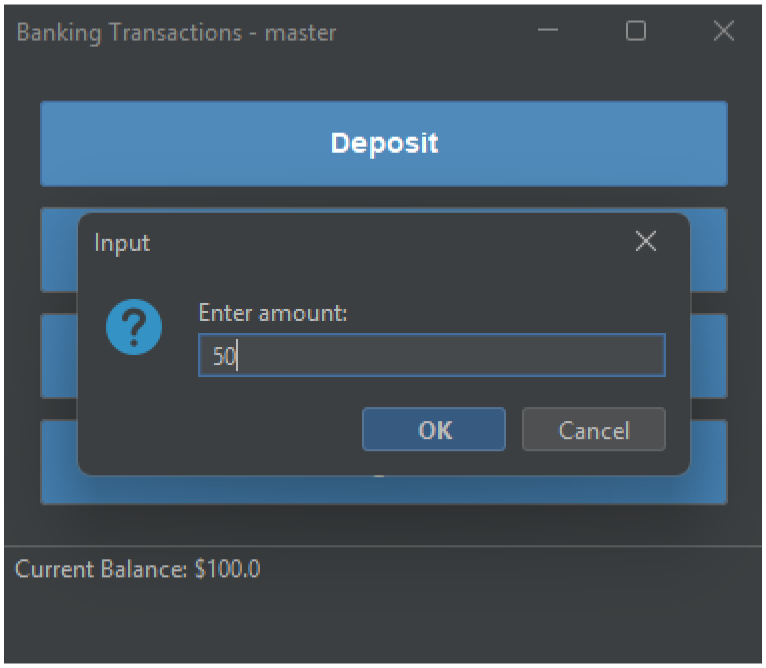


Figure 4: Transactions Page



Current Balance: $100.0

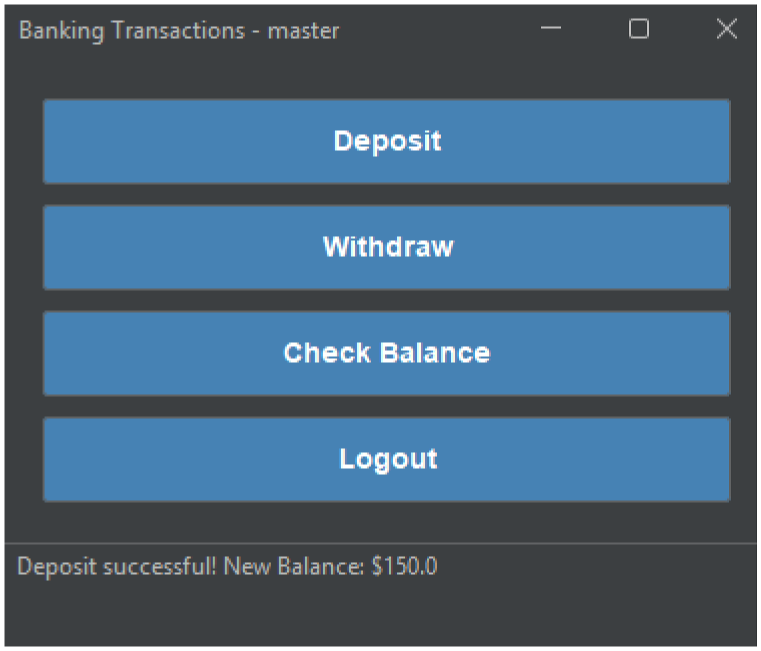
Figure 5: Balance Check Transaction



Enter amount: 50

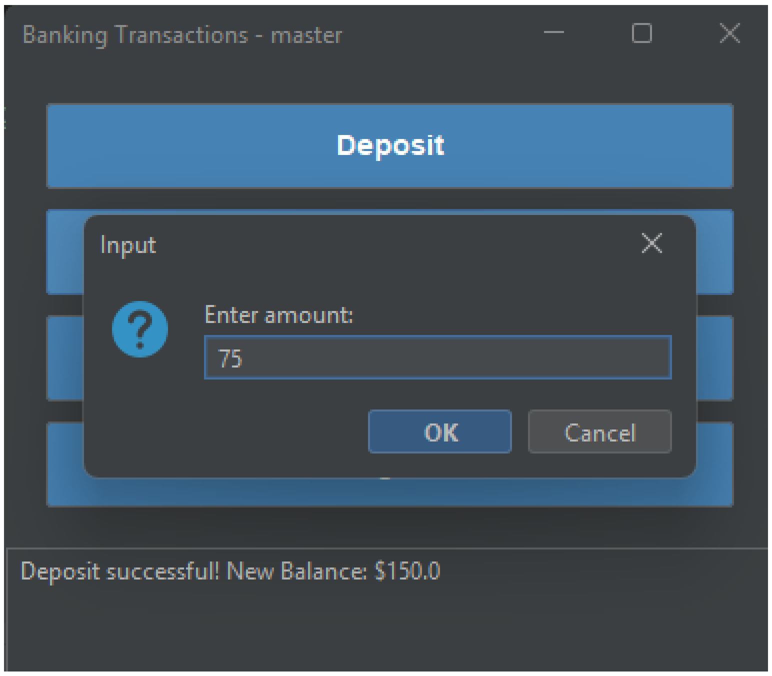
Current Balance: $100.0

Figure 6: Deposit Transaction



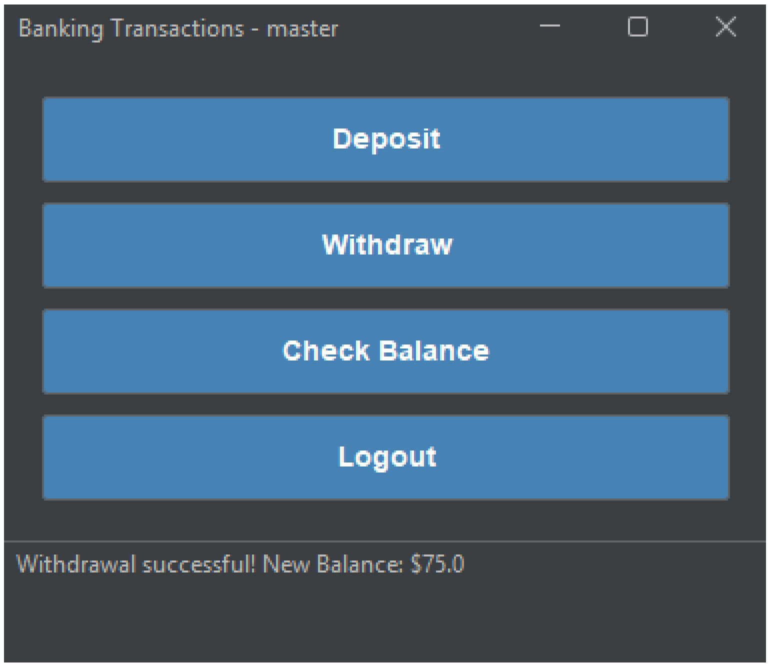
Deposit successful! New Balance: $150.0

Figure 7: Successful Deposit Transaction



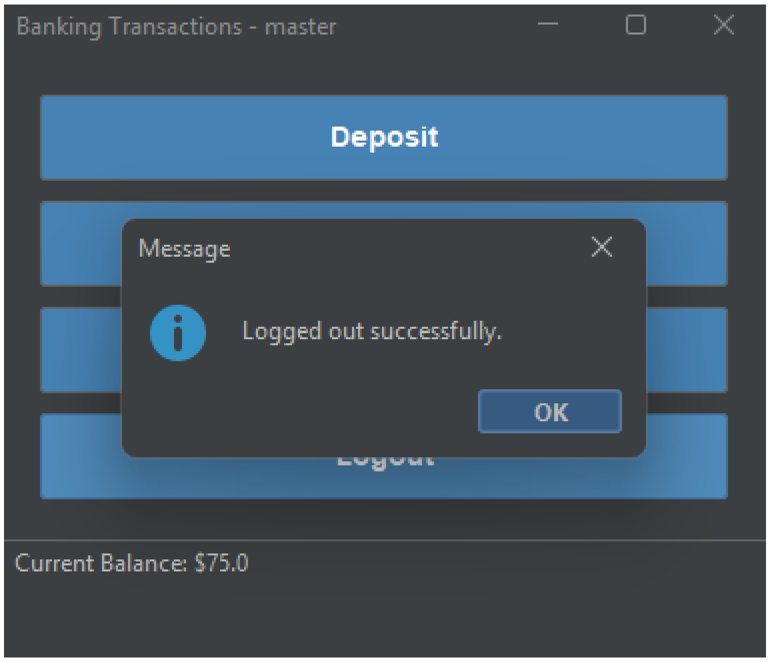
Deposit successful! New Balance: $150.0

Figure 8: Withdrawal Transaction



Withdrawal successful! New Balance: $75.0

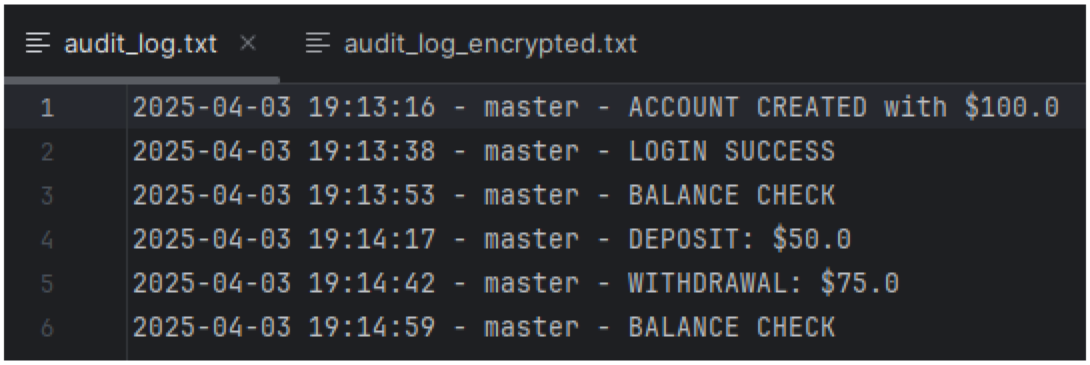
Figure 9: Successful Withdrawal Transaction



Logged out successfully.

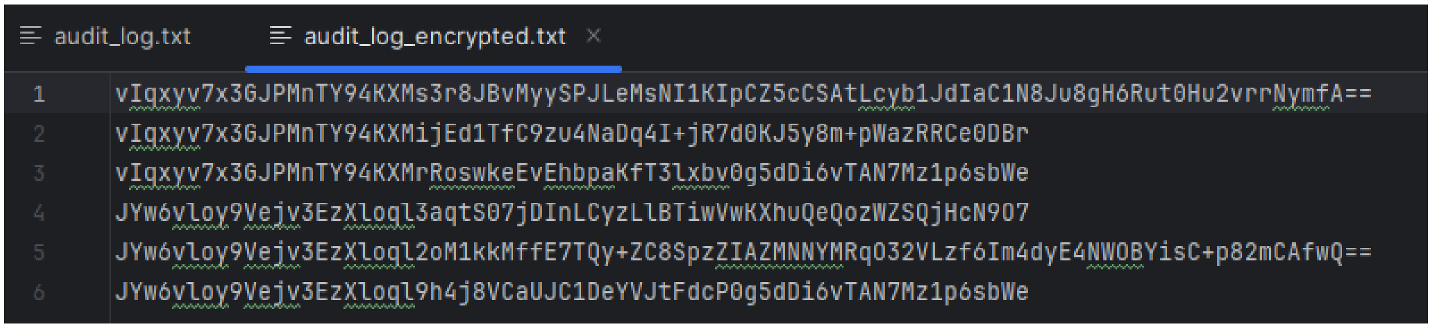
Current Balance: $75.0

Figure 10: Logout



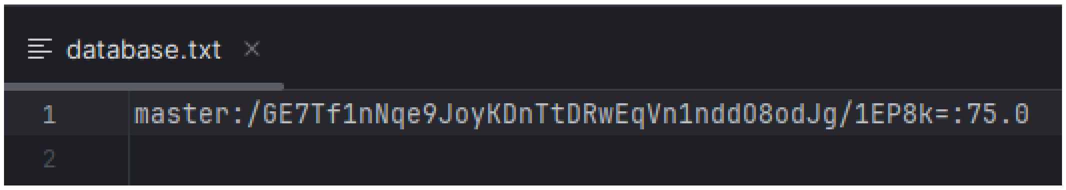
```
2025-04-03 19:13:16 - master - ACCOUNT CREATED with $100.0
2025-04-03 19:13:38 - master - LOGIN SUCCESS
2025-04-03 19:13:53 - master - BALANCE CHECK
2025-04-03 19:14:17 - master - DEPOSIT: $50.0
2025-04-03 19:14:42 - master - WITHDRAWAL: $75.0
2025-04-03 19:14:59 - master - BALANCE CHECK
```

Figure 11: Unencrypted Audit Log After All Transactions for Viewing Purposes



```
vIqxyv7x3GJPMnTY94KXMs3r8JBvMyySPJLeMsNI1KIpCZ5cCSAtLcyb1JdIaC1N8Ju8gH6Rut0Hu2vrrNymfA==
vIqxyv7x3GJPMnTY94KXMijEd1TfC9zu4NaDq4I+jR7d0KJ5y8m+pWazRRCe0DBr
vIqxyv7x3GJPMnTY94KXMrRoswkeEvEhbpaKfT3lxbv0g5dDi6vTAN7Mz1p6sbWe
JYw6vloy9Vejv3EzXloql3aqtS07jDInLCyzLlBTiwVwKXhuQeQozWZSQjHcN907
JYw6vloy9Vejv3EzXloql2oM1kkMffE7TQy+ZC8SpzZIAZMNNYMRq032VLzf6Im4dyE4NWOBYisC+p82mCAfwQ==
JYw6vloy9Vejv3EzXloql9h4j8VCaUJC1DeYVJtFdcP0g5dDi6vTAN7Mz1p6sbWe
```

Figure 12: Encrypted Audit Log After All Transactions



```
master:/GE7Tf1nNqe9JoyKDnTtDRwEqVn1ndd08odJg/1EP8k=:75.0

```

Figure 13: Database After All Transactions

Conclusion

This project has been an important learning experience that has allowed us, the students, to more deeply understand secure system design and implementation by using an easily understood example. Throughout the project, we have explored the different complexities associated with establishing secure communication by using the concepts learned in this course. We learned how key distribution protocols function in actual code while also understanding how to derive and manage encryption and MAC keys effectively. Additionally, having to implement cryptographic techniques, handle errors, and multi-threaded client servers has made us understand the key concepts of this course.

The nature of this project has also allowed us to develop our personal leadership and teamwork skills, which are ultimately invaluable. Each team member participated in this project and played a key role while leveraging each person's strengths to ensure its success.

<u>Contributions:</u>

Hamid:

- Built BankServer.java and ATMClient.java
- Designed and implemented the full client-server architecture
- Developed BankServer.java to:
  - Handle secure user login, registration, deletion
  - Manage account balances (deposit, withdrawals, checking balance, etc.)
- Developed ATMClient.java to:
  - Allow user interaction with the banking server
  - Send encrypted commands and receive secure responses
- Structured clean communication flow between server and client
- Laid the foundation for encryption, GUI integration, and logging functionality

Mohammad:

- Set up the AES Encryption to ensure the protection of data
- Set up the log files to output the encrypted data using AES encryption to an audit log that can be used to justify actions
- Added an unencrypted log file for demo/viewing purposes
- Completed the Introduction and Design portions of the report

Simrat:

- Designed security protocols for deposits, withdrawals, and balance checks to preserve the integrity and confidentiality of data
- Set up code to generate and verify MAC (message authentication codes) to ensure data integrity
- Ensured the bank server and ATM client have a proper connection with the master key to ensure the MAC key, encryption key, and secret key are shared to ensure proper encryption and decryption of messages
- Ensured proper verification of encryption keys and MAC keys to avoid false verification
- Completed Results section of the report

Ammar:

- Worked on developing the graphical user interface (GUI) for the ATM client using Java Swing
- Helped create the Login, Registration, Delete Account, and Transaction pages
- Focused on designing a simple and user-friendly interface
- Ensured proper integration between the GUI, backend logic, and security features
- Completed Conclusion section of the report