



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

Course Title:	Network Security
Course Number:	COE 817
Semester/Year:	W2025
Instructor:	Dr. Truman Yang
Assignment Title:	Project Final Report - Secure Banking System
Submission Date:	April 9th, 2025
Due Date	April 12th, 2025

TA Name:	Safwan Hasan
Section	082

Student Last Name	Student First Name	Student Number	Email	Signature*
Baaj	Tala	500995802	tala.baaj@torontomu.ca	T.B.
Coskun	Munevver	500923319	munevver.coskun@torontomu.ca	M.C.

\*By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a “0” on the work, an “F” in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:

<https://www.torontomu.ca/senate/policies/academic-integrity-policy-60/>.

## Table of Contents

- 1. Introduction**
  - 2. System Design and Implementation**
    - 2.1 Project Architecture and Overview
    - 2.2 Authenticated Key Distribution Protocol
    - 2.3 Key Derivation and Security Protocols
  - 3. Testing & Results**
  - 4. Conclusion**
- 

## 1. Introduction

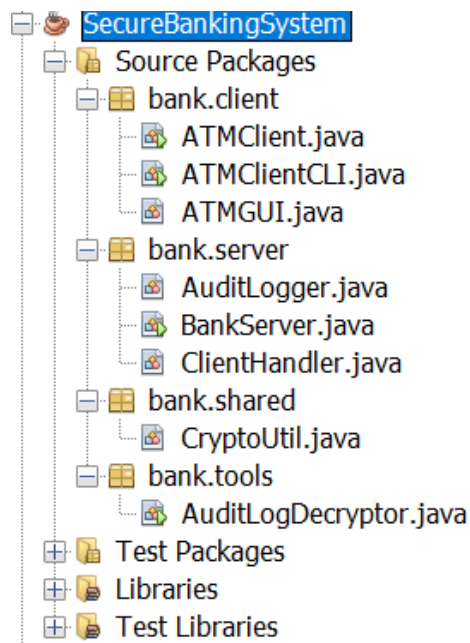
The purpose of this project is to develop a Secure Banking System to allow secure and encrypted transactions to take place between a central bank server and ATM clients. This system shows how ATM banking networks ensure security , confidentiality and authentication during the interactions with customers. The core functionalities of this project include:

Account (Registration and Login)
Details on balance, deposits and withdrawals
Secure exchange of data using AES encryption and HMAC verification
Handle multiple clients with concurrent secure sessions
Activity's encrypted audit logging

The scope of this project covers multiple ATM clients interacting with a central server (GUI and CLI), mutual authentication and encryption based on session, as well as thread-safe server that is capable of handling multiple users simultaneously. Some limitations this project faced included having no database so the data is stored in memory.

## 2. System Design and Implementation

### 2.1 Project Architecture and Overview



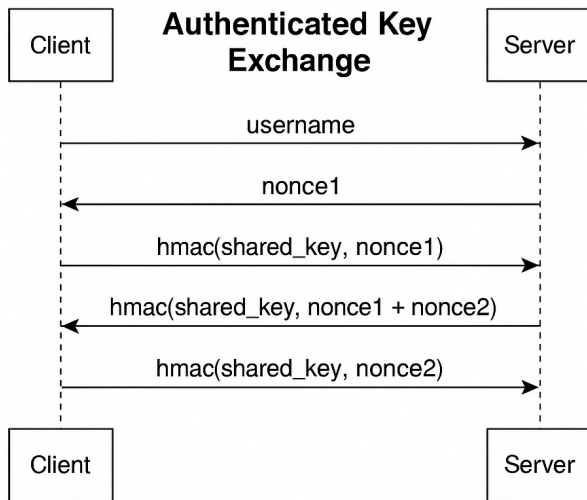
**Screenshot 1:** The Project structure shows design of the Secure Banking System. The system is composed of four java packages including bank.client, bank.server, bank.shared, and bank.tools.

The system's four java packages are structured to cover the following:

- Bank.server covers the main logic of the server (BankServer.java) and threading (ClientHandler.java). It handles authentication and processes transactions.
- Bank.client covers the CLI interface (ATMClientCLI.java) and the GUI interface (ATMClient.java). It handles multiple ATM Clients, which can be run in either GUI or CLI form.
- Bank.shared covers cryptographic functions (CryptoUtil.java). A Crypto Utility helps manage key derivation, encryption, decryption, and HMAC.
- Bank.tools has the log decryptor tool (AuditLogDecryptor.java). An Audit Logger creates encrypted logs of users actions.

The GUI interface is user-friendly and makes ATM operations more straightforward. Each client request is routed securely to the server and processed over an encrypted connection.

## 2.2 Authenticated Key Distribution Protocol



To ensure the establishment of secure sessions, the client and server perform a custom mutual authentication exchange using a shared static key through the following steps:

- Client sends login information by inputting username and password
- Server replies with nonce as a challenge
- Client replies with its own nonce and an encrypted message with the shared key that contains both nonces.
- Both sides derive a master key session when validation passes.

This ensures that only confirmed users who know the shared key can connect and that each session uses new random inputs to avoid replay attacks.

## 2.3 Key Derivation and Security Protocols

The `CryptoUtil.deriveKey()` method derives two keys from the master session key. It derives ENC key which is used for AES messages encryption and MAC key which is used for HMAC-SHA256 message authentication.

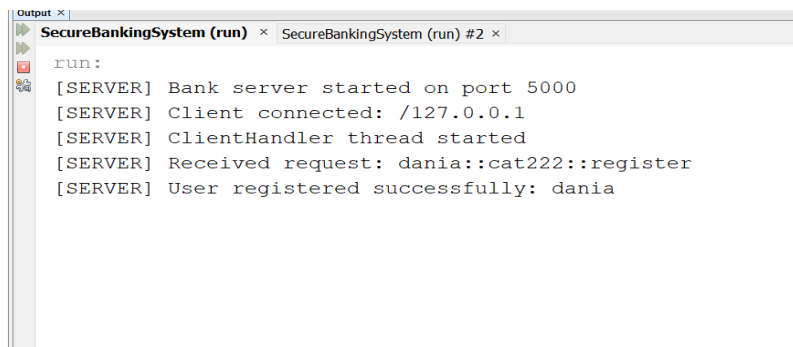
The client's commands such as deposit are encrypted using AES, signed with HMAC and sent to the server for verification and decryption. The server executes these commands, logs the results and returns encrypted responses. All the user's actions are saved in an encrypted audit log that can be viewed by running (`AuditLogDecryptor.java`).

Thus, the techniques used are symmetric cryptography(AES-128), HMAC for integrity, nonce-based authentications, thread-based session handling and file-based encrypted audit logging.

### 3. Testing & Results

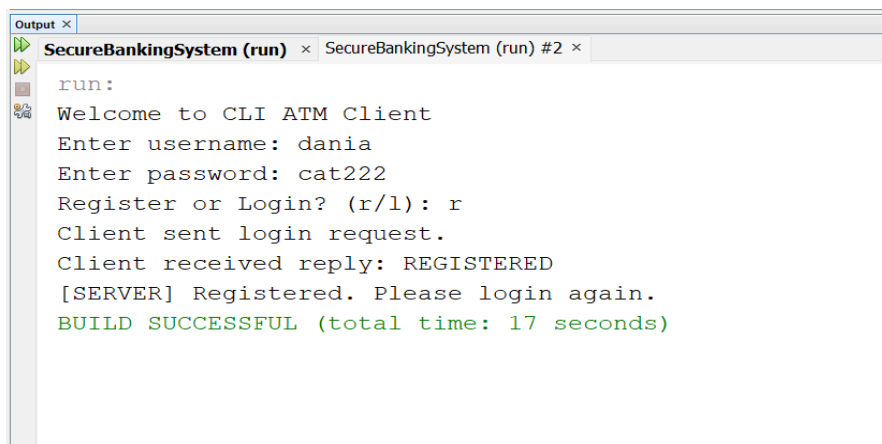
We tested the system using both CLI and GUI versions of the ATM client. Our goal is to demonstrate the core functionalities which are user registration, login, transaction security, audit logging and multiple client support. Additionally, we ensure these functionalities are demonstrated with encrypted communication and isolated sessions.

Step 1: Launching server and registering a new user (dania).



```
Output x
SecureBankingSystem (run) x SecureBankingSystem (run) #2 x
run:
[SERVER] Bank server started on port 5000
[SERVER] Client connected: /127.0.0.1
[SERVER] ClientHandler thread started
[SERVER] Received request: dania::cat222::register
[SERVER] User registered successfully: dania
```

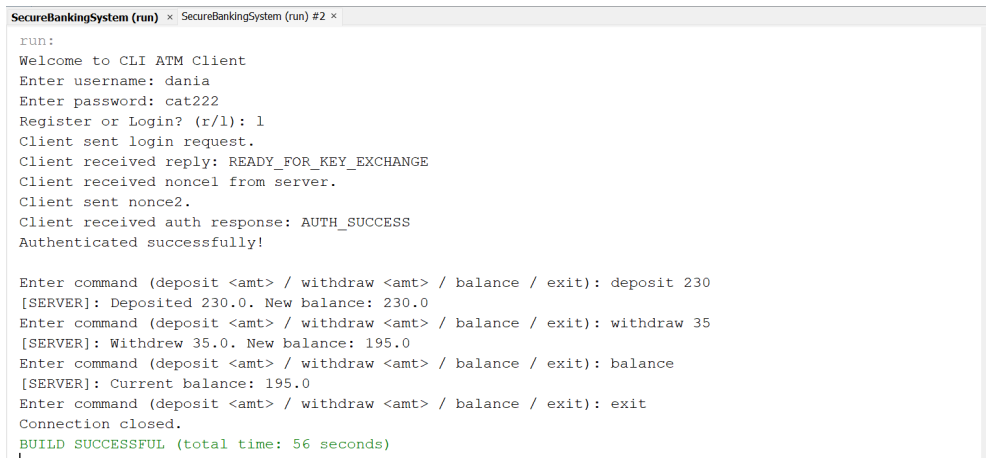
**Screenshot 2:** Server startup message showing the Bank Server is running and listening on port 5000 and is ready to accept incoming ATM client connection.



```
Output x
SecureBankingSystem (run) x SecureBankingSystem (run) #2 x
run:
Welcome to CLI ATM Client
Enter username: dania
Enter password: cat222
Register or Login? (r/l): r
Client sent login request.
Client received reply: REGISTERED
[SERVER] Registered. Please login again.
BUILD SUCCESSFUL (total time: 17 seconds)
```

**Screenshot 3:** CLI client dania connects and registers successfully with the server using username dania and password cat222. Server logs confirm the registration and connection.

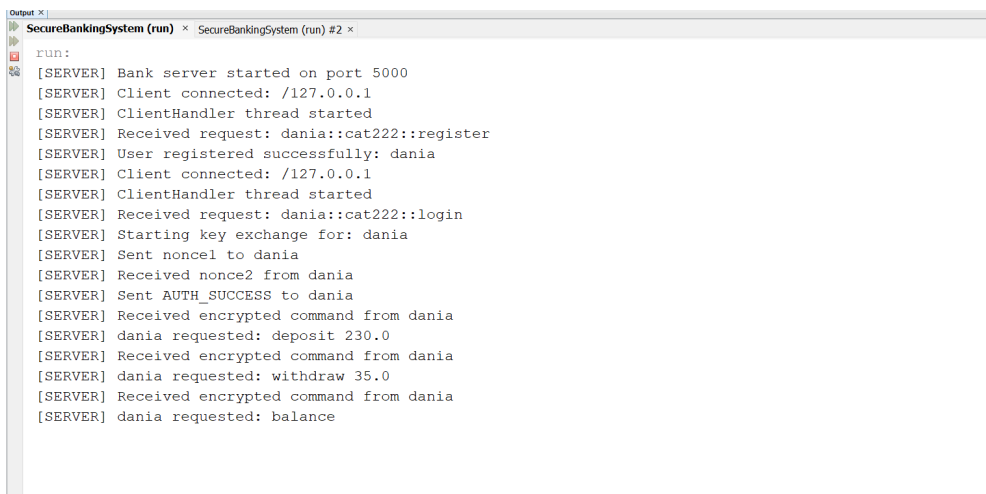
Step 2: Logging in with CLI and establishing a secure connection after registration.



```
SecureBankingSystem (run) x SecureBankingSystem (run) #2 x
run:
Welcome to CLI ATM Client
Enter username: dania
Enter password: cat222
Register or Login? (r/l): l
Client sent login request.
Client received reply: READY_FOR_KEY_EXCHANGE
Client received nonce1 from server.
Client sent nonce2.
Client received auth response: AUTH_SUCCESS
Authenticated successfully!

Enter command (deposit <amt> / withdraw <amt> / balance / exit): deposit 230
[SERVER]: Deposited 230.0. New balance: 230.0
Enter command (deposit <amt> / withdraw <amt> / balance / exit): withdraw 35
[SERVER]: Withdrew 35.0. New balance: 195.0
Enter command (deposit <amt> / withdraw <amt> / balance / exit): balance
[SERVER]: Current balance: 195.0
Enter command (deposit <amt> / withdraw <amt> / balance / exit): exit
Connection closed.
BUILD SUCCESSFUL (total time: 56 seconds)
```

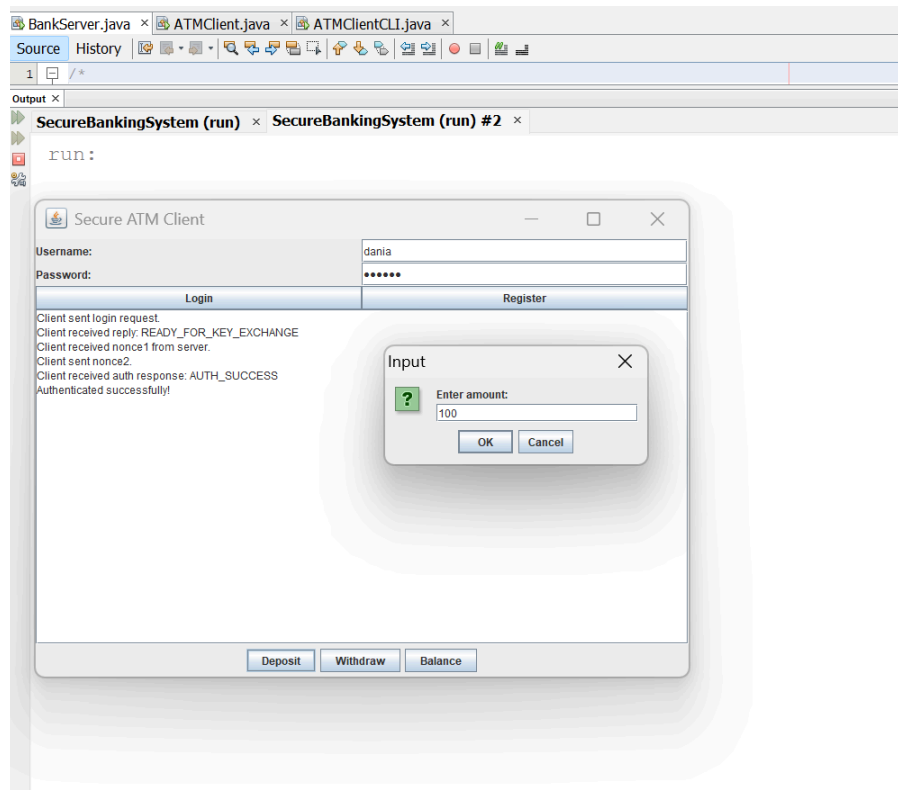
**Screenshot 4:** User dania logs in from the CLI using valid credentials. User proceeds to deposit 230 into her account and then withdraws 35 showing the current balance to be 195.



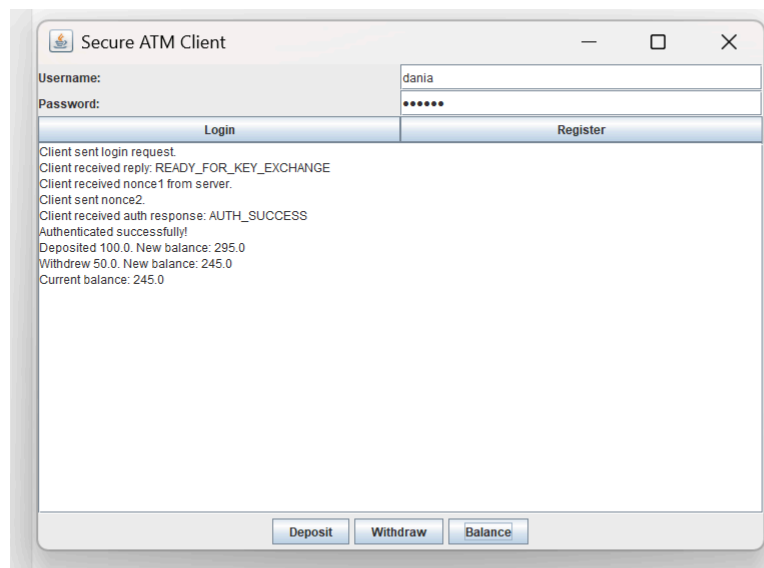
```
Output x
SecureBankingSystem (run) x SecureBankingSystem (run) #2 x
run:
[SERVER] Bank server started on port 5000
[SERVER] Client connected: /127.0.0.1
[SERVER] ClientHandler thread started
[SERVER] Received request: dania::cat222::register
[SERVER] User registered successfully: dania
[SERVER] Client connected: /127.0.0.1
[SERVER] ClientHandler thread started
[SERVER] Received request: dania::cat222::login
[SERVER] Starting key exchange for: dania
[SERVER] Sent nonce1 to dania
[SERVER] Received nonce2 from dania
[SERVER] Sent AUTH_SUCCESS to dania
[SERVER] Received encrypted command from dania
[SERVER] dania requested: deposit 230.0
[SERVER] Received encrypted command from dania
[SERVER] dania requested: withdraw 35.0
[SERVER] Received encrypted command from dania
[SERVER] dania requested: balance
```

**Screenshot 5:** The server confirms successful login, initiates key exchange, and establishes a secure session.

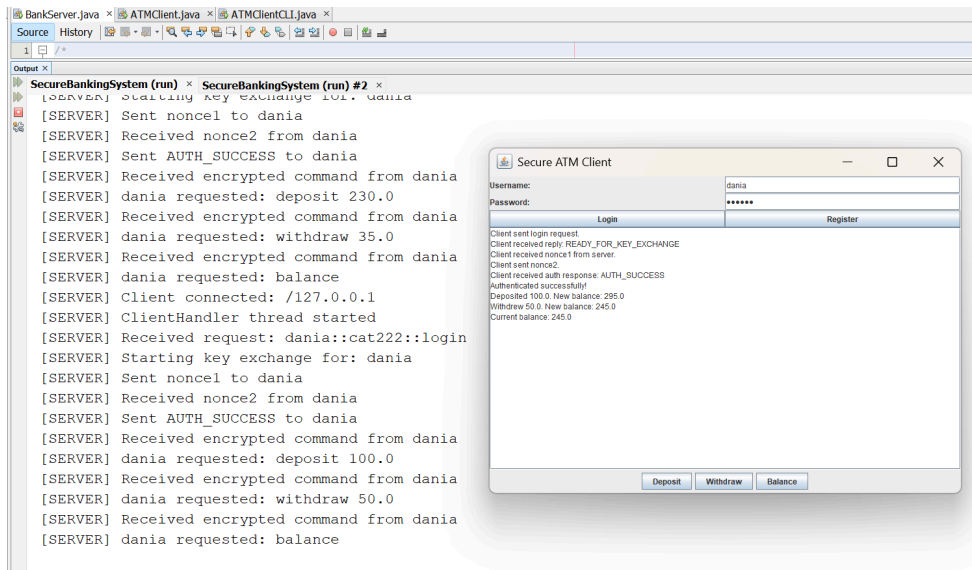
Step 3: Logging in and performing transactions via GUI



**Screenshot 6:** User dania selects the Deposit action from the GUI and enters an amount of 100.0. The transaction is transmitted securely to the server.

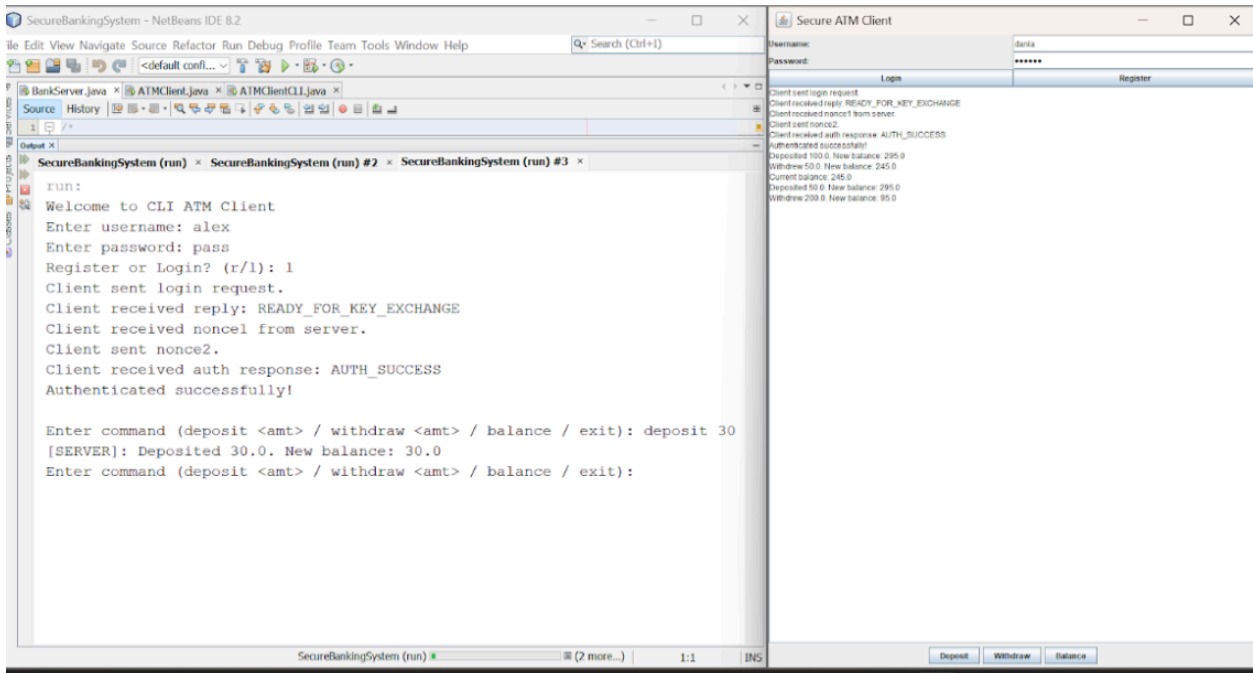


**Screenshot 7:** The server processes the deposit request for 100.0. This amount is added to her previous balance of 195, updating the balance to 295. Then she withdrew 50 making the new balance 245.



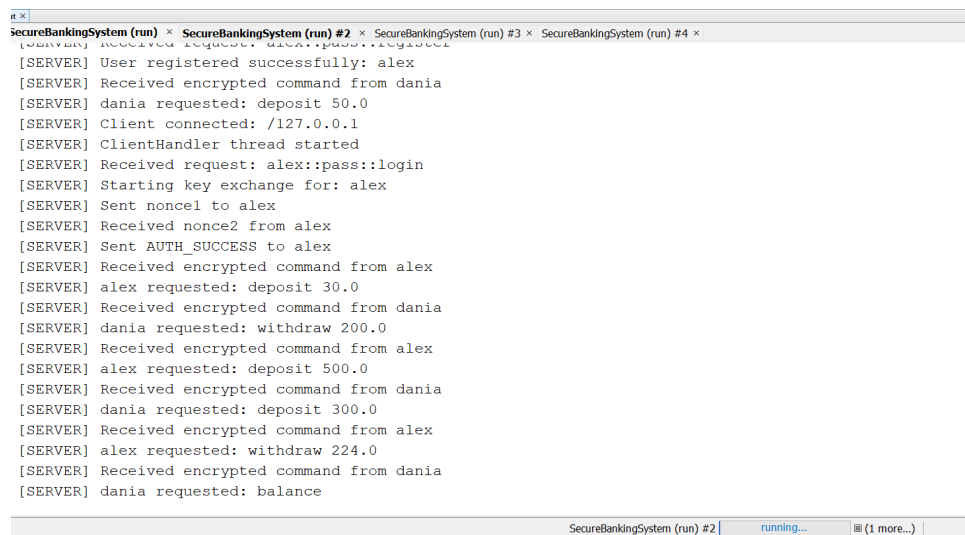
**Screenshot 8:** User dania performs a balance inquiry through the GUI interface. The server retrieves and returns the current account balance after all previous transactions.

Step 4: Handling multiple clients simultaneously and session isolation.





**Screenshot 9:** Terminal logs show concurrent ATM sessions for alex and dania, confirming successful multi-client handling, thread-safe processing, and secure session isolation.

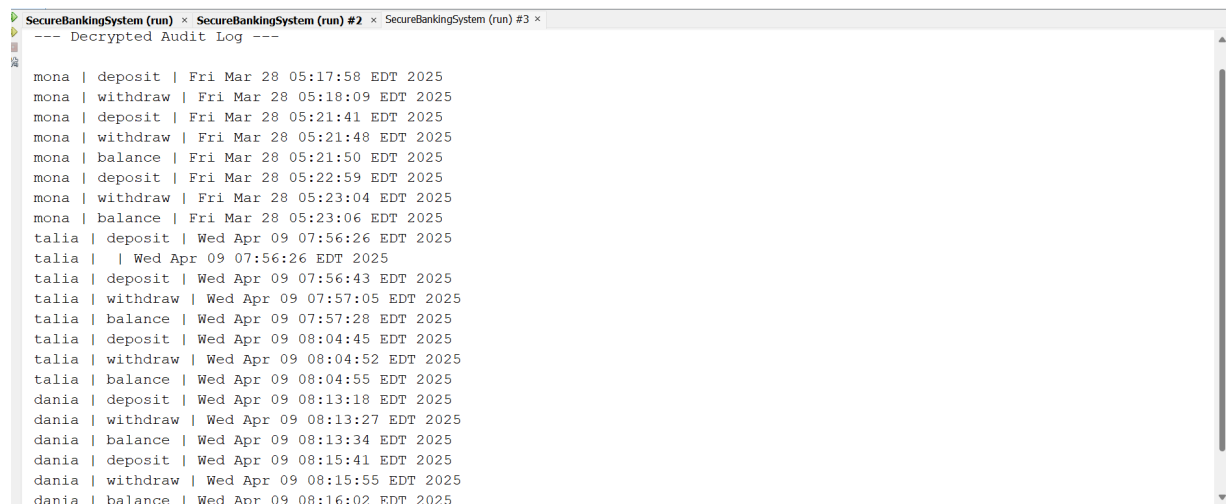


The screenshot shows a terminal window with four tabs, all titled "SecureBankingSystem (run)". The active tab displays a series of log messages from the server. The logs indicate that the server is handling multiple clients simultaneously. For example, it shows "User registered successfully: alex", "Received encrypted command from dania", "danial requested: deposit 50.0", "Client connected: /127.0.0.1", "ClientHandler thread started", "Received request: alex::pass::login", "Starting key exchange for: alex", "Sent nonce to alex", "Received nonce2 from alex", "Sent AUTH\_SUCCESS to alex", "Received encrypted command from alex", "alex requested: deposit 30.0", "Received encrypted command from dania", "danial requested: withdraw 200.0", "Received encrypted command from alex", "alex requested: deposit 500.0", "Received encrypted command from dania", "danial requested: deposit 300.0", "Received encrypted command from alex", "alex requested: withdraw 224.0", "Received encrypted command from dania", and "danial requested: balance". The status bar at the bottom shows "SecureBankingSystem (run) #2" and "running...".

```
# x |
SecureBankingSystem (run) × SecureBankingSystem (run) #2 × SecureBankingSystem (run) #3 × SecureBankingSystem (run) #4 ×
[SERVER] Received request: alex::pass::login
[SERVER] User registered successfully: alex
[SERVER] Received encrypted command from dania
[SERVER] danial requested: deposit 50.0
[SERVER] Client connected: /127.0.0.1
[SERVER] ClientHandler thread started
[SERVER] Received request: alex::pass::login
[SERVER] Starting key exchange for: alex
[SERVER] Sent nonce to alex
[SERVER] Received nonce2 from alex
[SERVER] Sent AUTH_SUCCESS to alex
[SERVER] Received encrypted command from alex
[SERVER] alex requested: deposit 30.0
[SERVER] Received encrypted command from dania
[SERVER] danial requested: withdraw 200.0
[SERVER] Received encrypted command from alex
[SERVER] alex requested: deposit 500.0
[SERVER] Received encrypted command from dania
[SERVER] danial requested: deposit 300.0
[SERVER] Received encrypted command from alex
[SERVER] alex requested: withdraw 224.0
[SERVER] Received encrypted command from dania
[SERVER] danial requested: balance
SecureBankingSystem (run) #2 | running... | (1 more...) | 1
```

**Screenshot 10:** Displays the server being able to process requests of both users at once, confirming that GUI access is capable of handling multiple users.

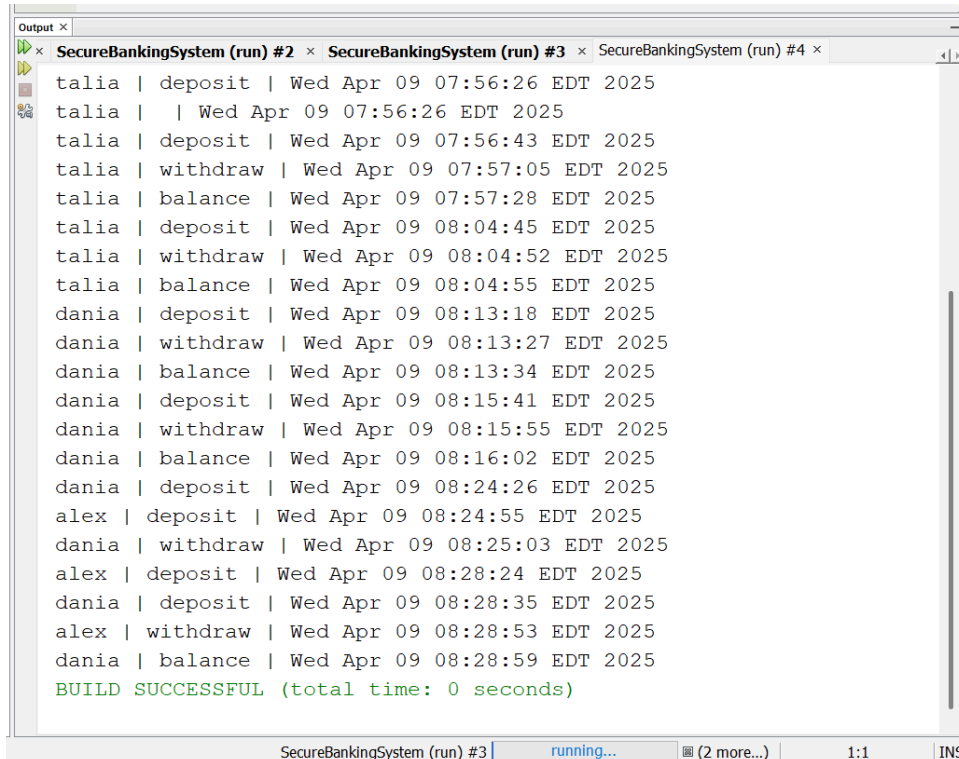
**Step 5:** Viewing the encrypted audit log including multiple clients.



The screenshot shows a terminal window with three tabs, all titled "SecureBankingSystem (run)". The active tab displays a "Decrypted Audit Log" with a list of transactions. The log entries are as follows:

Client	Action	Date	Time	Timezone
mona	deposit	Fri Mar 28	05:17:58	EDT 2025
mona	withdraw	Fri Mar 28	05:18:09	EDT 2025
mona	deposit	Fri Mar 28	05:21:41	EDT 2025
mona	withdraw	Fri Mar 28	05:21:48	EDT 2025
mona	balance	Fri Mar 28	05:21:50	EDT 2025
mona	deposit	Fri Mar 28	05:22:59	EDT 2025
mona	withdraw	Fri Mar 28	05:23:04	EDT 2025
mona	balance	Fri Mar 28	05:23:06	EDT 2025
talial	deposit	Wed Apr 09	07:56:26	EDT 2025
talial		Wed Apr 09	07:56:26	EDT 2025
talial	deposit	Wed Apr 09	07:56:43	EDT 2025
talial	withdraw	Wed Apr 09	07:57:05	EDT 2025
talial	balance	Wed Apr 09	07:57:28	EDT 2025
talial	deposit	Wed Apr 09	08:04:45	EDT 2025
talial	withdraw	Wed Apr 09	08:04:52	EDT 2025
talial	balance	Wed Apr 09	08:04:55	EDT 2025
danial	deposit	Wed Apr 09	08:13:18	EDT 2025
danial	withdraw	Wed Apr 09	08:13:27	EDT 2025
danial	balance	Wed Apr 09	08:13:34	EDT 2025
danial	deposit	Wed Apr 09	08:15:41	EDT 2025
danial	withdraw	Wed Apr 09	08:15:55	EDT 2025
danial	balance	Wed Apr 09	08:16:02	EDT 2025

**Screenshot 11:** Output from AuditLogDecryptor displays encrypted log entries for mona,talia and dania confirming secure recording of multiple actions.



```
Output x
SecureBankingSystem (run) #2 x SecureBankingSystem (run) #3 x SecureBankingSystem (run) #4 x
talia | deposit | Wed Apr 09 07:56:26 EDT 2025
talia | | Wed Apr 09 07:56:26 EDT 2025
talia | deposit | Wed Apr 09 07:56:43 EDT 2025
talia | withdraw | Wed Apr 09 07:57:05 EDT 2025
talia | balance | Wed Apr 09 07:57:28 EDT 2025
talia | deposit | Wed Apr 09 08:04:45 EDT 2025
talia | withdraw | Wed Apr 09 08:04:52 EDT 2025
talia | balance | Wed Apr 09 08:04:55 EDT 2025
dania | deposit | Wed Apr 09 08:13:18 EDT 2025
dania | withdraw | Wed Apr 09 08:13:27 EDT 2025
dania | balance | Wed Apr 09 08:13:34 EDT 2025
dania | deposit | Wed Apr 09 08:15:41 EDT 2025
dania | withdraw | Wed Apr 09 08:15:55 EDT 2025
dania | balance | Wed Apr 09 08:16:02 EDT 2025
dania | deposit | Wed Apr 09 08:24:26 EDT 2025
alex | deposit | Wed Apr 09 08:24:55 EDT 2025
dania | withdraw | Wed Apr 09 08:25:03 EDT 2025
alex | deposit | Wed Apr 09 08:28:24 EDT 2025
dania | deposit | Wed Apr 09 08:28:35 EDT 2025
alex | withdraw | Wed Apr 09 08:28:53 EDT 2025
dania | balance | Wed Apr 09 08:28:59 EDT 2025
BUILD SUCCESSFUL (total time: 0 seconds)
SecureBankingSystem (run) #3 | running... | (2 more...) | 1:1 | INS
```

**Screenshot 12:** Continued audit log output shows activity for users dania and alex, including the simultaneous activity of both users which shows actions are logged correctly even when running at the same time.

## 4. Conclusion

The Secure Banking System project provided us with knowledge on designing and building a secure network-based application that is multi-threaded. By being able to simulate an ATM banking environment, we studied key security concepts such as session-based encryption, authenticated key exchange and message integrity with the use of HMAC.

During the project we utilized java sockets to implement secure communication, applied AES for symmetric encryption and HMAC for message verification, developed CLI and GUI to simulate user interaction, managed concurrent client sessions using multi-thread design and ensured transactions were encrypted and logged for audit.

## Team Contributions

Student 1 (Munevver): Responsible for backend socket logic, implementation of key exchange protocol, encryption module and CLI setup.

Student 2 (Tala): Responsible for GUI design, development in frontend, testing and documentation.

