Computer and Network Security

ATM-BANK COMMUNICATION PROTOCOL

Group 1

Heer Kubadia (22110096)
Jiya Desai (22110107)
Lavanya (22110130)
Nishi Shah (22110171)
Pratham Sharda (22110203)
Harshita Singh (22110299)

1. Introduction

The ATM Protocol project involves implementing two programs: an atm client and a bank server. The atm allows customers to perform operations like account creation, deposits, withdrawals, and balance checks. The bank server handles these operations securely, managing a single ATM at a time and maintaining data consistency. The primary goal is to build a secure system that adheres to the specified communication protocol and error-handling requirements.

2. System Architecture and Tools

The system follows a client-server architecture:

- ATM Client (atm): Acts as a client that interacts with users and sends commands to the bank server.
- Bank Server (bank): Acts as a server that processes requests from multiple atm clients, and handles transactions securely.
- Database/Storage file: Maintains user data and other account information (confidential).

Communication Flow

1. Atm connects to the bank server using TCP sockets.
2. Atm sends requests (account creation, deposit, withdrawal, balance check) to the bank.
3. Bank authenticates the request using the auth file and processes it.
4. Bank responds with a JSON-formatted result or error message.

5. Atm displays the response to the user.

Tentative Tools

The project utilizes C++ for implementation, with *make* as the build automation tool to compile and manage the atm and bank programs. Other languages and frameworks that might be used are Python, C, and MERN stack.

Libraries

- Networking: Sockets for communication between atm and bank, like BSD Sockets (<sys/socket.h>).
- JSON Parsing: Libraries like cJSON or custom parsers
- Cryptography: Libraries such as OpenSSL for authentication and secure communication

3. Modules

ATM Client (atm)

1. Command-Line Interface Module: Parses command-line arguments and handles input validation.
   - Functions: parse_arguments(), validate_input()
   - Error handling for incorrect formats and invalid commands.

2. Authentication Module: Reads and validates the auth and card files.
   - Functions: read_auth_file(), validate_auth(), read_card_file()
   - Ensures that files are correctly formatted and valid.

3. Network Communication Module: Establishes a connection to the bank server and sends/receives data.
   - Functions: connect_to_bank(), send_request(), receive_response()
   - Manages secure communication and handles timeouts.

4. Request Handler Module: Constructs and sends requests based on user operations.
   - Functions: create_account(), deposit(), withdraw(), check_balance()
   - Converts user commands into properly formatted requests.

Bank Server (bank)

1. Server Initialization Module: Initializes the server, sets up sockets, and generates the auth file.

- Functions: initialize_server(), generate_auth_file()
- Handles SIGTERM for graceful shutdown.

2. Connection Manager Module (*under consideration): Manages multiple concurrent connections from atm clients.

- Functions: accept_connections(), handle_client()
- Uses threading or asynchronous I/O for concurrent processing.

3. Authentication Module: Authenticates incoming connections using the auth file.

- Functions: authenticate_atm()
- Ensures that only legitimate ATMs can communicate with the server.

4. Transaction Processing Module: Processes transactions (account creation, deposit, withdrawal, balance check).

- Functions: process_create_account(), process_deposit(), process_withdraw(), process_check_balance()
- Ensures atomicity, consistency, isolation, and durability (ACID properties) for transactions.

5. Error Handling Module: Manages errors and ensures protocol compliance.

- Functions: handle_protocol_error(), rollback_transaction()
- Ensures that the server responds with appropriate error messages.

Additional Features under Consideration

Potential enhancements for future versions include two-factor authentication (2FA) for added security, advanced transaction history with search and filtering options, setting a limit on the amount of money a user can withdraw in a day or week, temporarily locking an account after a specified number of failed login attempts to prevent brute force attacks, notifying users when their account balance drops below a certain threshold, weekly spending tracker and real-time currency conversion. We are also considering an enhanced UI/UX design to make the interface more intuitive and visually appealing. Together, these features aim to improve the security, usability, and versatility of the system.

4. Security Considerations

- Authentication Model: Both atm and bank use the auth file for mutual authentication. This file must be transferred securely via a trusted channel.
- Secure Communication: Considering using TLS for secure communication between atm and bank to prevent eavesdropping and man-in-the-middle attacks.
- Data Validation: Ensure all inputs are validated to prevent buffer overflow attacks or invalid data entries.

5. Testing Plan

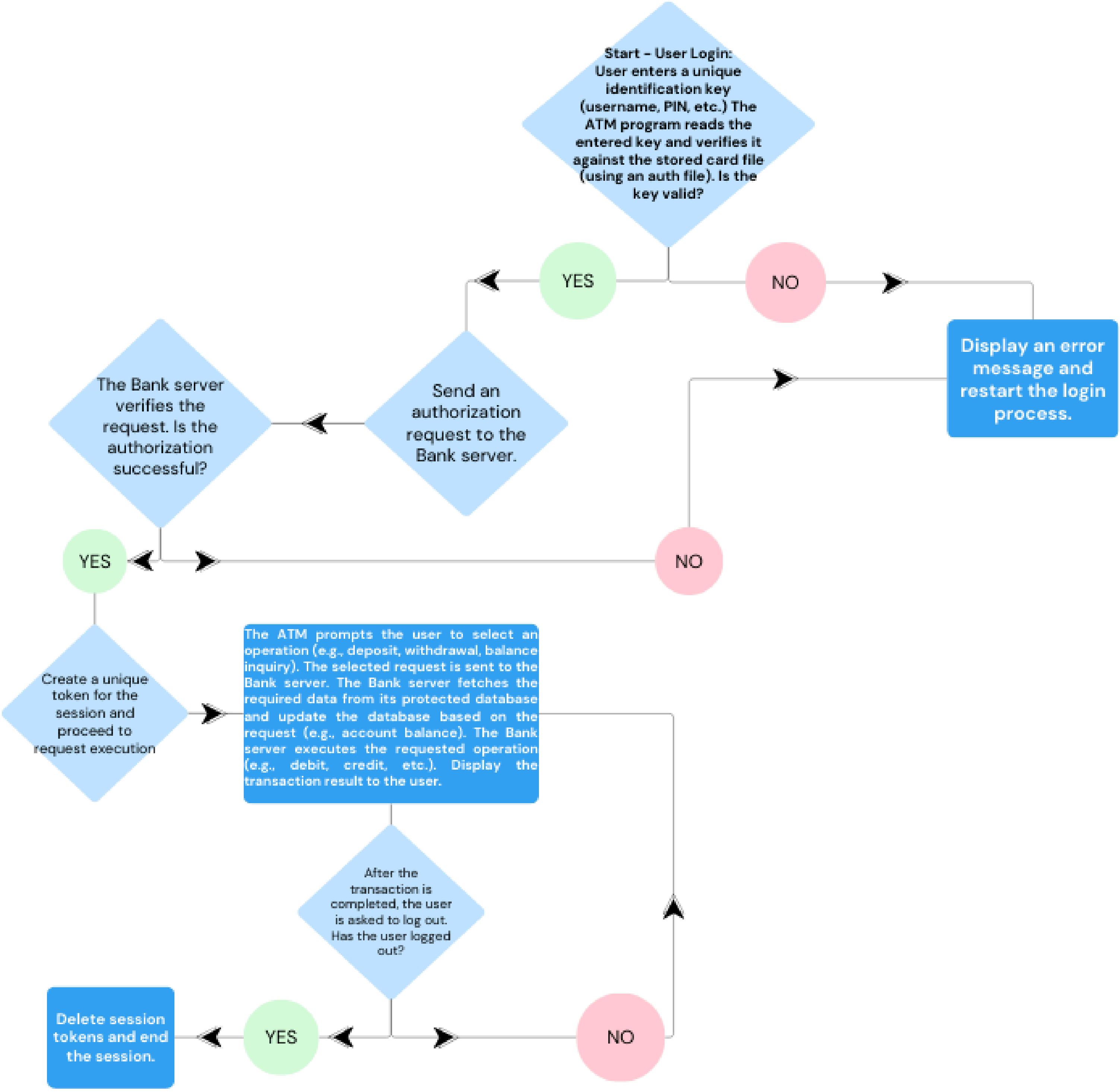
An essential part of the project will be the final testing stage which will ensure the reliability of the system and includes tests such as:

- Unit Tests: Test individual functions and modules for correct functionality.
- Integration Tests: Test communication between atm and bank to ensure correct protocol implementation.
- Edge Case Tests: Test invalid inputs, protocol errors, and system boundaries to ensure robustness.

6. Work Distribution

Name	Work
Harshita Singh	Database building, generating tests
Heer Kubadia	Working on the atm program (command-line interface module), UI/UX (optionally, as an additional feature)
Jiya Desai	Working on the atm program (network communication module, request handler module)
Lavanya Sharma	Working on the bank program (server initialization module, error handling module)
Nishi Shah	Working on the bank program (transaction processing module)
Pratham Sharda	Security and encryption (auth file)

The work allocation is dynamic and will change as per the complexity of tasks and extra tasks undertaken.



Start – User Login: User enters a unique identification key (username, PIN, etc.) The ATM program reads the entered key and verifies it against the stored card file (using an auth file). Is the key valid?

Display an error message and restart the login process.

Send an authorization request to the Bank server.

The Bank server verifies the request. Is the authorization successful?

Create a unique token for the session and proceed to request execution

The ATM prompts the user to select an operation (e.g., deposit, withdrawal, balance inquiry). The selected request is sent to the Bank server. The Bank server fetches the required data from its protected database and update the database based on the request (e.g., account balance). The Bank server executes the requested operation (e.g., debit, credit, etc.). Display the transaction result to the user.

After the transaction is completed, the user is asked to log out. Has the user logged out?

Delete session tokens and end the session.

Flow chart of the envisioned design

7. Conclusion

This design document provides a comprehensive overview of the ATM Protocol project, including the architecture, module breakdown, task allocation, and security considerations. The next steps are to implement the modules as described, test thoroughly, and integrate them to deliver a robust and secure ATM Protocol system.

8. References

[1] https://www.geeksforgeeks.org/socket-programming-in-cpp/
[2] https://github.com/IITGN-CS431/problems/tree/main/atm