VoteGuard: A Secure and Encrypted Voting System

1. Abstract

The Secure Voting System project aims to develop a reliable and secure electronic voting platform using a hybrid encryption model combining RSA and AES algorithms. The primary objective is to ensure vote confidentiality, data integrity, and secure key exchange between clients and the server. The system leverages a client-server architecture, where clients securely cast votes via a GUI interface developed with Tkinter, and the server stores encrypted votes in a MySQL database. The system supports multi-client handling using POSIX threads and uses RSA for key exchange and AES for vote encryption, achieving a robust and secure voting process.

2. Introduction

Project Background and Relevance:

In today's digital age, secure and reliable voting systems are essential for maintaining the integrity of democratic processes. Traditional voting methods, such as paper ballots, are prone to human errors, tampering, and logistical challenges, which can compromise the accuracy and fairness of the voting outcome. To address these issues, electronic voting systems have emerged as a viable alternative. However, the adoption of such systems introduces new challenges, particularly concerning data security, voter authentication, and vote integrity. Cyber-attacks, data breaches, and unauthorized access are significant threats that can undermine the credibility of electronic voting.

Given these challenges, the primary goal of this project is to develop a Secure Voting System that combines robust encryption techniques, secure data transmission, and user-friendly interfaces to ensure the safety and reliability of voting processes. The project leverages the hybrid encryption model, utilizing both RSA (Rivest–Shamir–Adleman) and AES (Advanced Encryption Standard) to ensure data confidentiality and integrity during transmission and storage. The RSA algorithm, an asymmetric encryption method, secures the transmission of the AES session key, while AES, a symmetric encryption technique, encrypts the actual vote data. This combination is essential to maintain both speed and security, as RSA alone can be computationally expensive for large data volumes.

Objectives

The primary objective of this project is to develop a secure voting system that ensures the integrity and confidentiality of votes while allowing efficient vote collection and processing. The system is designed to achieve the following:

Secure Vote Transmission: Implement end-to-end encryption using a hybrid model of RSA and AES to protect votes during transmission between the client and server.

Efficient Client-Server Communication: Utilize TCP sockets for reliable data transfer, ensuring that votes are transmitted without loss or corruption. By leveraging Edge Computing principles, the system reduces the risk of latency issues that could compromise voting accuracy.

User Authentication and Access Control: Integrate secure user authentication using bcrypt hashing to safeguard user credentials. The system distinguishes between admin and user roles, allowing only authorized personnel to view and analyze voting results.

Multi-Client Handling: Employ POSIX threads to enable the server to handle multiple voting clients simultaneously without performance degradation. This feature ensures scalability, allowing the system to function effectively even during high voter turnout.

Data Integrity and Storage: Store encrypted votes and user credentials securely in a MySQL database. The use of symmetric and asymmetric encryption guarantees that votes remain confidential and tamper-proof.

Real-Time Vote Decryption for Admins: Develop a user-friendly admin panel using Tkinter, where authorized personnel can view both encrypted and decrypted votes in a secure environment. This feature enhances the transparency and accountability of the voting process.

Comprehensive GUI for Voters: Create an intuitive graphical interface using Tkinter for users to register, log in, and cast their votes. The client application will guide users through the voting process and confirm successful vote submission.

System Robustness and Error Handling: Implement rigorous error handling and encryption validation to prevent unauthorized access or vote manipulation.

By accomplishing these objectives, the Secure Voting System aims to bridge the gap between digital innovation and electoral security. The project not only demonstrates the practical application of cryptographic techniques in safeguarding votes but also highlights the potential of IoT and Edge Computing in enhancing the reliability and efficiency of modern voting systems. Through this project, we aim to showcase a secure, scalable, and user-friendly voting platform that can serve as a foundation for future advancements in electronic voting technologies.

3. System Overview

System Architecture:

The Secure Voting System follows a client-server architecture that ensures secure data transmission, vote encryption, and multi-client handling. The system is designed with a focus on maintaining data confidentiality, integrity, and availability while allowing multiple users to cast votes simultaneously. The primary components of the system include the Client Application, Server Application, Database, Encryption and Decryption Modules, and Admin Panel. Below is a detailed outline of the system architecture and the data flow between these components.

Key Components of the System:

Client Application (User Interface)

The client application, developed using Python and Tkinter, serves as the primary interface for users to interact with the system. Users can perform the following tasks:

- Register with a username and password.
- Log in to cast their votes.
- Select their preferred programming language (Python, Java, C, JavaScript) and submit their vote securely.

The client application is responsible for:

- User authentication and password hashing using bcrypt.
- Secure vote encryption using the AES algorithm.
- Generating a random session key for each voting session.
- Sending the encrypted vote and session key to the server.

Encryption and Decryption Modules:

The encryption module is implemented on the client side, while the decryption module resides on the server side. The system utilizes a hybrid encryption model that combines AES and RSA:

AES Encryption (Symmetric):

- Used for encrypting the vote before transmission.
- Fast and efficient for short data (like votes).

RSA Encryption (Asymmetric):

- Used to securely transmit the AES session key from client to server.
- Ensures that only the server, with its private key, can decrypt the session key.

Data Encryption Flow:

- The client generates a random AES session key for each voting session.
- The session key is encrypted using the server's public RSA key.
- The actual vote is encrypted using AES with the generated session key.
- Both the encrypted session key and the encrypted vote are transmitted to the server.

Server Application (Vote Handling and Decryption)

The server application, developed in C using POSIX threads, handles incoming client connections and stores votes securely. Key functionalities include:

- Accepting client connections via TCP sockets.
- Handling multiple clients concurrently using threading.
- Decrypting the received session key using the server's RSA private key.
- Decrypting the vote using the AES session key.
- Storing encrypted votes and corresponding user data in the database.

Data Decryption Flow:

- The server receives the encrypted AES session key and encrypted vote.
- The RSA private key decrypts the session key.
- The decrypted session key is used to decrypt the vote.
- Decrypted votes are compared against valid options (Python, Java, C, JavaScript) and counted.

Database (MySQL):

The database is responsible for securely storing user credentials, encrypted votes, and session keys. It contains the following tables:

- Users2 Table: Stores user credentials (username, password hash, and admin status).
- Votes3 Table: Stores encrypted votes along with the user ID and session key.

Database Operations:

- Insertion of new user data during registration.
- Storing votes upon successful decryption.
- Retrieving vote data for the admin panel.

Admin Panel (Results Display):

The admin panel, built using Tkinter, allows authorized personnel to:

- Log in securely using a predefined password.
- View encrypted and decrypted votes.
- Display vote counts for each language.
- Handle decryption errors and display invalid votes.

The admin panel retrieves the stored encrypted votes from the database, decrypts them using the session key, and displays both the encrypted and decrypted votes for verification.

Data Flow:

- User registers or logs in through the client application.
- The client application generates a random session key and encrypts the vote using AES.
- The session key is encrypted using the server's public RSA key.
- The client sends both the encrypted vote and session key to the server.
- The server decrypts the session key using its private RSA key.
- The decrypted session key is used to decrypt the vote.
- The server stores the encrypted vote in the MySQL database.
- Admin accesses the panel to view decrypted votes and vote counts.

Security Measures:

- Hybrid encryption (AES + RSA) ensures both secure key exchange and data confidentiality.
- Password hashing with bcrypt prevents plaintext password storage.
- Multi-client support using POSIX threading ensures that one client does not block others.
- Base64 encoding allows binary data to be stored and transmitted as text.

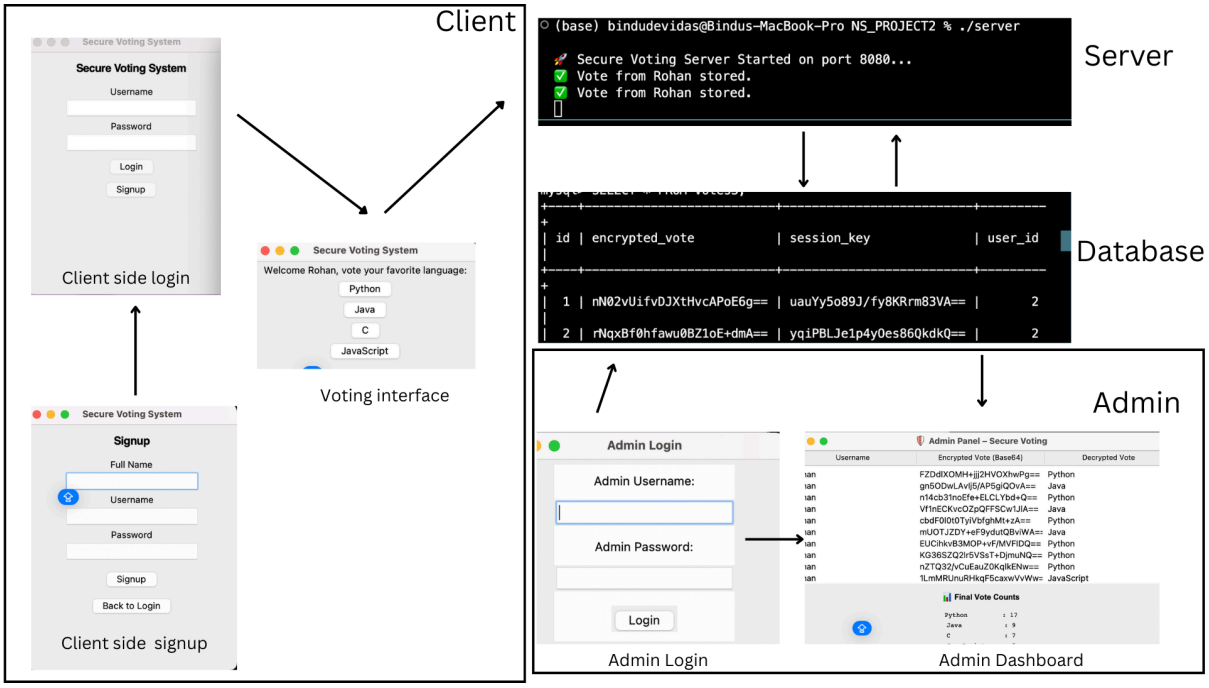

Figure 1: Screenshots of the Client, Admin and Server

4. Design:

The Secure Voting System is designed using a client-server architecture, emphasising data security, integrity, and efficient multi-client handling. The system consists of four major components: Client Application, Server Application, Database, and Admin Panel. Each component plays a crucial role in maintaining a secure and reliable voting process. Below, we detail each component and their interactions.

1. Client Application (Python & Tkinter):

The client application serves as the user interface for registration, login, and voting. It is developed using Python's Tkinter library to provide a user-friendly experience. The main functions of the client application are:

- User Registration: Allows new users to create accounts. The password is hashed using bcrypt before storing it in the database to prevent plaintext password leakage.
- User Login: Authenticates users by comparing hashed passwords stored in the database.
- Vote Casting: Upon successful login, users select their preferred programming language to vote. The vote is encrypted using AES with a randomly generated session key.

- Session Key Encryption: The AES session key is encrypted using the server's public RSA key to ensure secure transmission.
- Data Transmission: Both the encrypted vote and the encrypted session key are sent to the server via TCP.

2. Server Application (C & POSIX Threads):

The server application is implemented in C, leveraging POSIX threads to handle multiple client connections simultaneously. The primary functions of the server are:

- Connection Handling: Uses TCP sockets to accept incoming connections. Each client connection is managed by a separate thread, allowing concurrent vote submissions.
- Vote Reception: Receives encrypted votes and encrypted session keys from clients.
- Session Key Decryption: Uses the private RSA key to decrypt the session key.
- Vote Decryption: Utilizes the decrypted session key to decrypt the actual vote.
- Database Insertion: The server inserts the encrypted vote into the database, along with the user ID and session key, after successful decryption.
- Concurrency Management: Uses mutex locks to ensure that database operations are performed without conflict between threads.

3. Database (MySQL):

The MySQL database acts as the central repository for storing user credentials and encrypted votes. It contains two primary tables:

- Users2 Table: Stores user information, including username, hashed password, and admin status.
- Votes3 Table: Stores the encrypted vote, user ID, and session key.

Data flow to the database occurs when:

- A new user registers (credentials are stored in users2).
- A vote is successfully cast (encrypted data is inserted into Votes3).
- Admin queries the database for vote results.
  The database ensures data consistency and efficient querying, even when accessed by multiple threads concurrently.

4. Admin Panel (Python & Tkinter):

The admin panel provides a graphical interface for authorized personnel to view voting results. Key features include:

- Admin Authentication: Ensures that only authorized users can access the results.

- Vote Decryption: Retrieves encrypted votes from the database and decrypts them using the stored session key.
- Vote Counting: Displays the total count for each programming language in a user-friendly format.
- Error Handling: If decryption fails, the system marks the vote as invalid.

Component Interactions:

- Client to Server: Clients send encrypted votes and session keys via TCP.
- Server to Database: After decryption, the server inserts the encrypted vote and session key into the database.
- Admin Panel to Database: The panel fetches vote data, decrypts it, and displays the results.
- Data Security: The RSA + AES hybrid encryption model ensures that the transmission between client and server remains secure.

By maintaining distinct roles for each component, the system ensures scalability, security, and efficient vote management, making it robust against unauthorized access and data tampering.

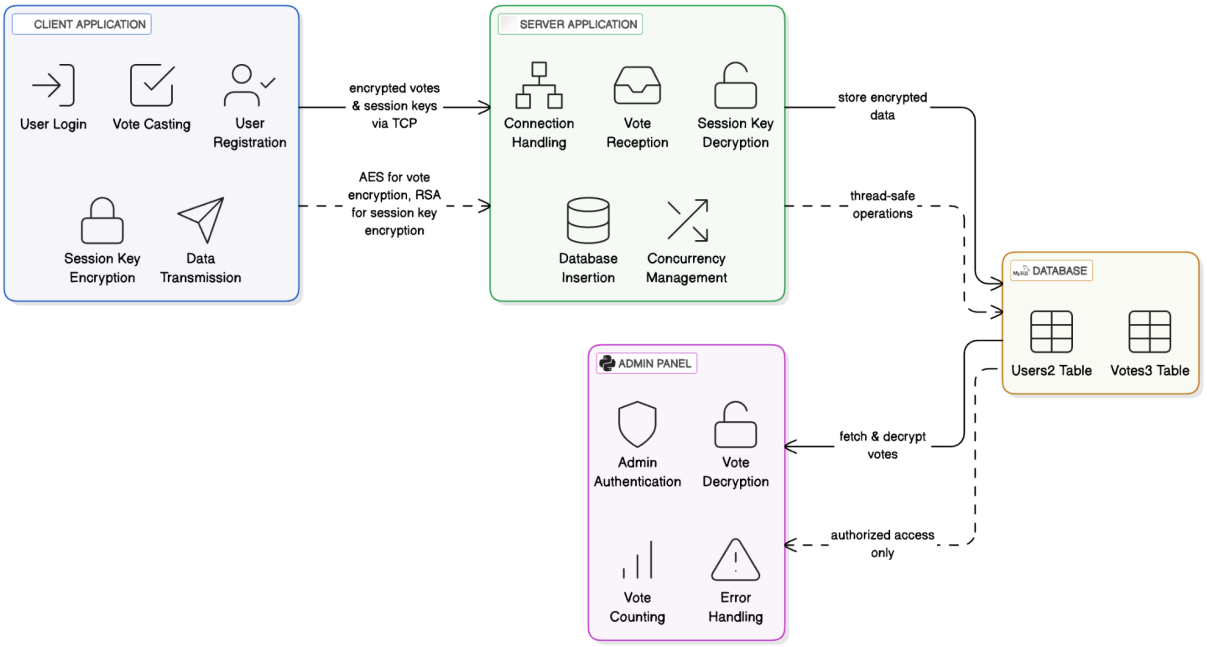

Figure 2: Dataflow and Design of VoteGuard

5. Security Features:

The Secure Voting System is meticulously designed to address several critical network security challenges commonly encountered in electronic voting systems. These challenges include data integrity, confidentiality, authentication, secure transmission,

and multi-client handling. Below is a detailed breakdown of the security features incorporated into the system.

1. Hybrid Encryption (AES + RSA):

One of the primary security features of the system is the implementation of a hybrid encryption model that combines AES (Advanced Encryption Standard) and RSA (Rivest-Shamir-Adleman) encryption techniques.

AES for Vote Encryption:

- AES is a symmetric encryption algorithm used to encrypt the actual vote data.
- The client generates a random AES session key for each voting session.
- The vote is padded to a fixed length and encrypted using the AES key.
- The use of AES-128 with ECB (Electronic Codebook) mode is appropriate due to the short and fixed-length nature of the vote data.

RSA for Key Exchange:

- RSA, an asymmetric encryption algorithm, is used to encrypt the AES session key.
- The client uses the server's public key to encrypt the session key before sending it.
- On the server side, the private RSA key decrypts the received session key.

Rationale for Hybrid Encryption:

- Using RSA alone for the entire vote would be computationally expensive, as RSA is not suitable for large data.
- AES is faster for bulk encryption, while RSA ensures secure key exchange.
- This combination ensures both security and performance.

2. Secure Data Transmission (TCP Sockets):

- The client-server communication occurs over TCP sockets, which provide reliable and ordered data transmission.
- Unlike UDP, TCP ensures that all data packets are delivered and in the correct order, which is crucial for maintaining the integrity of votes.
- During transmission, both the encrypted vote and the encrypted session key are sent to the server, minimizing the risk of plaintext data being intercepted.
- Using localhost (127.0.0.1) for development ensures that data remains within the local machine, reducing exposure during testing.

3. Password Security (Bcrypt Hashing):

- User passwords are never stored as plaintext in the database.

- During registration, passwords are hashed using the bcrypt library, which adds a random salt to each password.
- The bcrypt algorithm is designed to be computationally intensive, making brute-force attacks significantly more challenging.
- During login, the entered password is hashed again and compared with the stored hash, preventing direct comparison with plaintext.

4. Multi-Client Security (Thread Safety):

- The server is designed to handle multiple clients concurrently using POSIX threads.
- To prevent race conditions when multiple clients submit votes simultaneously, a mutex lock is employed during database operations.
- This ensures that only one thread can access the database at a time, maintaining data consistency.

5. Data Integrity and Storage Security:

- Encrypted votes and session keys are stored directly in the MySQL database.
- Even if the database is compromised, the stored data remains secure due to its encrypted format.
- The database is only accessible to the server, reducing the risk of unauthorized access.

6. Admin Access Control:

- Only authorized administrators can view vote results via the admin panel.
- The admin login is protected by a password check, and any unauthorized access attempts are logged and displayed as errors.
- The admin panel also decrypts the votes only when accessed by a legitimate user, ensuring that the decrypted results are not stored or exposed elsewhere.

***7. Error Handling and Logging*:**

- The system implements robust error handling to manage failed decryption attempts, invalid votes, and unauthorized admin access.
- Whenever an error occurs during decryption or database interaction, the error is logged without exposing sensitive information.
- This approach not only maintains security but also aids in auditing and debugging.

The Secure Voting System effectively addresses network security challenges through the integration of hybrid encryption, secure transmission protocols, hashed password storage, and multi-client handling. By employing industry-standard encryption algorithms (AES and RSA) and robust authentication mechanisms (bcrypt), the

system ensures vote confidentiality, integrity, and secure client-server communication. These security measures collectively make the voting system resilient to data tampering, interception, and unauthorized access.

6. System Requirements

1. Operating System: macOS

2. Programming Languages:

- Python 3.11+ (for client and admin applications)
- C (GCC 9.3+) (for server application)
- Database Management System:
- MySQL Server 8.0+
- MySQL Connector for Python (mysql-connector-python)

3. Libraries and Dependencies:

Python Libraries:

- Tkinter: For GUI development
- Bcrypt: For password hashing
- PyCryptodome: For AES and RSA encryption
- Socket: For client-server communication
- Base64: For encoding binary data

C Libraries:

- OpenSSL: For AES and RSA encryption
- Pthread: For multi-client handling
- MySQL C API: For database connectivity

Package Management:

- pip (Python package installer)
- apt-get (for installing required libraries on Linux)

4. Network Requirements:

- Protocol: TCP/IP
- Port: 8080 (can be configured)
- Localhost (127.0.0.1): For testing in a secure, offline environment
- Firewall: Open port 8080 to allow client-server communication
- SSL/TLS (optional): For enhanced network security in a production environment

7. Open-source libraries and tools

The Secure Voting System leverages several open-source libraries and tools to ensure secure communication, encryption, GUI development, and database management. Below is a summary of the key libraries and tools used:

Python Libraries:

Tkinter (Built-in with Python 3.11):

- GUI toolkit for creating user interfaces.
- Supports labels, buttons, text entries, and tables.

bcrypt (Version: 4.0.1):

- Password hashing with salting.
- Provides secure storage of user credentials.

PyCryptodome (Version: 3.17):

- Cryptographic functions including AES and RSA.
- Used for vote encryption and session key handling.

mysql-connector-python (Version: 8.0.33):

- Connects Python applications to MySQL databases.
- Supports executing queries and managing connections.

Socket (Built-in with Python 3.11):

- Facilitates TCP/IP communication between client and server.

Base64 (Built-in with Python 3.11):

- Encodes binary data to text for secure transmission and storage.

C Libraries:

OpenSSL (Version: 3.0.8):

- Provides AES and RSA encryption functions.
- Used for secure data transmission and decryption.

Pthread (POSIX Threads, Built-in on macOS):

- Supports multi-threading for handling concurrent clients.
- Ensures thread-safe database operations.

MySQL C API (Version: 8.0.33 via Homebrew):

- Interacts with MySQL for storing encrypted votes.
- Handles database connections and queries efficiently.

Development and Management Tools:

Homebrew (Version: 4.0.3):

- Package manager for installing libraries on macOS.
- Used to install MySQL, OpenSSL, and other dependencies.

GCC (Apple clang version 14.0.0):

- C compiler for building the server application.
- Supports multi-threading and encryption features.

OpenSSL CLI (Version: 3.0.8):

- Generates RSA key pairs for secure key exchange.

8. Implementation and Testing

The Secure Voting System was implemented as a client-server application using hybrid encryption (AES + RSA) for secure vote transmission. The client, built in Python with Tkinter, handles user registration, login, and vote submission. The server, built in C with POSIX threads, manages multi-client connections and stores encrypted votes in a MySQL database.

Testing Approach:

Functional Testing:

- Verified user registration, login, vote encryption, and secure transmission.
- Ensured server decryption and vote storage worked correctly.
- Validated the admin panel's ability to display decrypted votes.

Security Testing:

- Tested password hashing with bcrypt and vote encryption using AES and RSA.
- Attempted unauthorized access to ensure admin security.
- Simulated interception attacks to verify data remained encrypted.

Results:

- The system successfully handled secure vote transmission, hybrid encryption, and admin result viewing.

- Encryption and hashing methods effectively protected vote data.

9. Results

Features Supported:

The Secure Voting System successfully implements the following key features:

Secure Vote Submission:

- Uses hybrid encryption (AES + RSA) to ensure that votes are securely transmitted between client and server.
- AES encrypts the vote, while RSA encrypts the session key for secure key exchange.

User Authentication:

- Users can register and log in using a username and password.
- Passwords are securely hashed using bcrypt before being stored in the database.
- Only authenticated users can cast votes, and only authorized admins can view results.

Multi-Client Handling:

- The server supports concurrent connections using POSIX threading, allowing multiple users to cast votes simultaneously.
- Mutex locks ensure thread safety when accessing the database.

Vote Decryption and Counting:

- The admin panel displays both encrypted and decrypted votes for verification.
- Real-time vote counting for each programming language (Python, Java, C, JavaScript).
- Handles invalid votes gracefully by marking them as "Invalid".

User-Friendly GUI:

- The client and admin interfaces are built using Tkinter, offering a simple and intuitive user experience.
- Users can easily navigate through registration, login, and voting.

Future Work:

As of now, the project is relatively simple and uses Tkinter for the user interface. ***This choice was made because the primary focus was on securing votes and learning encryption techniques***. In the future, I would like to enhance the system by:

Improved Frontend:

- Redesigning the client and admin interfaces using React and Tailwind CSS for a more modern and responsive UI.

Cloud-Based Database:

- Integrating a cloud database like Supabase to make the system scalable and accessible from multiple locations.

Additional Functionalities:

- Sending confirmation emails after a successful vote submission to enhance user engagement.
- Upgrading the admin panel to include graphical visualizations of vote counts and more advanced analytics.

These improvements will not only enhance usability and scalability but also add more features that align with modern web application standards.

10. Conclusion

Through this project, I successfully designed and implemented a Secure Voting System that ensures the confidentiality, integrity, and security of votes. The system employs a hybrid encryption model combining AES and RSA, enabling secure transmission of votes from client to server. The project also features a robust user authentication mechanism using bcrypt for password hashing, and a multi-client handling capability via POSIX threading. Additionally, the system includes an admin panel that displays encrypted and decrypted votes with real-time vote counting. The client and admin interfaces were developed using Tkinter, making the application interactive and user-friendly.

Working on this project provided me with valuable insights and practical experience in several key areas:

Cryptography and Security:

- I gained hands-on experience with hybrid encryption techniques, understanding how AES (symmetric encryption) and RSA (asymmetric encryption) complement each other to ensure secure data transmission.
- I learnt the importance of key management and how to securely exchange session keys between client and server.
- Implementing bcrypt taught me how to securely hash and verify user passwords, enhancing my understanding of password security.

Network Programming:

- I developed skills in using TCP sockets for reliable communication between the client and server.
- I learnt how to set up multi-threaded servers using POSIX threads to handle multiple client connections concurrently.
- Managing thread safety using mutex locks was crucial to preventing race conditions during database operations.

Database Management:

- I gained practical knowledge in using MySQL for storing encrypted data securely.
- I learnt how to efficiently query and manage data while maintaining data integrity in a multi-client environment.

Software Design and Architecture:

- This project helped me understand how to integrate different components (client, server, database) into a cohesive system.
- I learnt the importance of maintaining modularity, ensuring that each component handled its specific responsibilities efficiently.

This project challenged me to think critically about security vulnerabilities and adopt best practices in secure coding. It strengthened my problem-solving skills, especially when dealing with encryption issues and multi-threading challenges. Additionally, it gave me a deeper understanding of real-world applications of cryptography, which is essential in secure software development.

Overall, building the Secure Voting System was a rewarding experience that not only deepened my technical knowledge but also enhanced my ability to apply theoretical concepts to practical, real-world problems.