DEPARTMENT OF

UNIVERSITY INSTITUTE OF COMPUTING

CHANDIGARH UNIVERSITY

MINI PROJECT FILE

SUBJECT NAME: OBJECT ORIENTED PROGRAMMING

SUBJECT CODE: 24CAH-201

Submitted by:

Name : Ritesh Raj, Sachin Singh, Shivam Pokhriyal

UID : 24BCA10109, 24BCA10079, 24BCA10080

Section : 24BCA-1B

Submitted To:

Name : Mrs. Konika Rani

Signature : ……..

ACKNOWLEDGMENT

We would like to express our sincere gratitude to **Mrs. Konika Rani**, Faculty, Department of Computer Applications, **Chandigarh University**, for her constant guidance, encouragement, and support throughout the development of our mini project titled **"Online Banking System."** Her valuable suggestions and continuous motivation helped us successfully complete this project as part of the subject **Object Oriented Programming (Subject Code: 24CAH-201).**

We are thankful for her patience, constructive feedback, and valuable time, which helped us understand various concepts of **C++ programming, file handling, and system design** more effectively. Her mentorship inspired us to apply theoretical knowledge to practical implementation.

We also extend our gratitude to the **Department of Computer Applications, Chandigarh University**, for providing us with the necessary resources and guidance to carry out this work successfully. Lastly, we would like to thank our families, friends, and classmates for their constant encouragement and support during the completion of this project.

Submitted By:

- **Ritesh Raj (UID: 24BCA10109)**
- **Sachin Singh (UID: 24BCA10079)**
- **Shivam Pokhriyal (UID: 24BCA10080)**

DATE: 07-11-2025

Table of Contents

MINI PROJECT : ONLINE BANKING SYSTEM

1. INTRODUCTION

The **Online Banking System** is a console-based application developed using the **C++ programming language** that simulates fundamental real-world banking operations. In today's era of digital transformation, banking has shifted from traditional branch-based services to modern online systems that offer customers faster, safer, and more convenient financial management.

This project is designed to replicate essential features of an online banking platform in a simplified environment. It enables users to perform common banking operations such as **creating new accounts, checking balances, depositing or withdrawing funds, applying for debit and credit cards, and managing loans** — all through a secure command-line interface.

The system utilizes **Object-Oriented Programming (OOP)** concepts such as *classes, objects, encapsulation, and abstraction* to structure and manage account-related data effectively. To ensure data persistence, it implements **file handling**, where each account's details are stored permanently in text files. Additionally, a **master database file** maintains a summary of all registered accounts.

To enhance security, the project includes a **PIN-based authentication system** with **basic encryption techniques** to safeguard user credentials. This ensures that unauthorized access to account information is prevented, providing a more realistic and secure banking experience.

Overall, this mini project demonstrates the practical application of **C++ programming, file handling, and security mechanisms** in developing a small-scale yet functional simulation of an **Online Banking Management System**.

2. OBJECTIVES

The main objective of this project is to design and implement a secure and efficient **Online Banking Management System** using the **C++ programming language**. This project aims to combine programming logic, file handling, and object-oriented principles to simulate real-world banking functionalities.

The specific objectives of the system are as follows:

1. **To develop a functional banking management system** using *Object-Oriented Programming (OOP)* concepts such as classes, objects, encapsulation, and abstraction.
2. **To implement secure user authentication** through a *PIN-based login system*, ensuring that only authorized users can access account details.
3. **To provide essential banking services** such as account creation, balance inquiry, deposits, withdrawals, loan processing, and card applications through a simple and interactive console interface.
4. **To demonstrate effective use of file handling** for *data persistence*, enabling permanent storage and retrieval of user information.

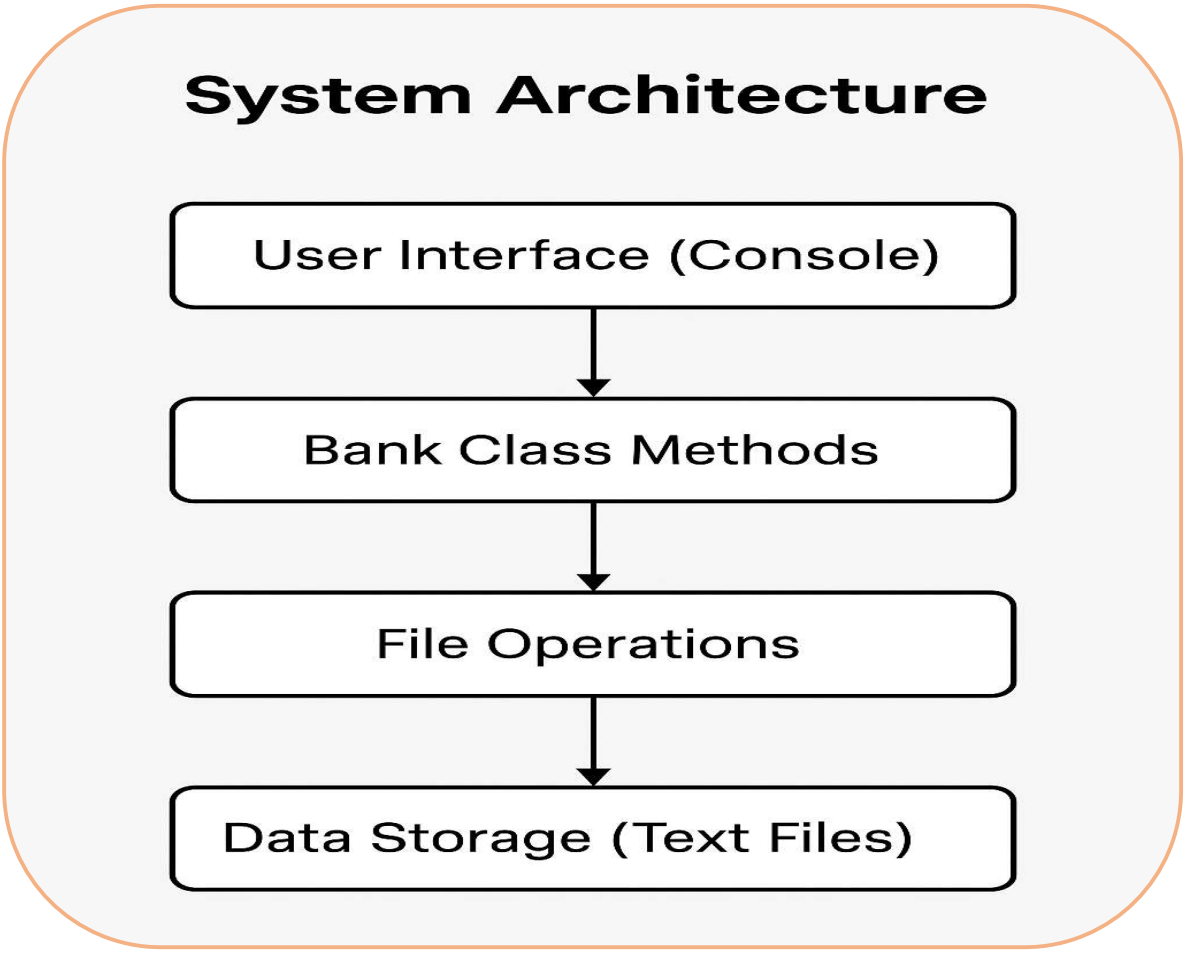
5. **To apply basic encryption methods** for securing sensitive account credentials and transaction details.
6. **To simulate real-world banking operations** by incorporating practical features like *loan approval, repayment, and debit/credit card management*.
7. **To maintain a centralized database** that records all customer accounts and updates automatically after each transaction for consistent record management.

3. SYSTEM DESIGN

The **Online Banking System** is designed using a modular and object-oriented approach. The project architecture ensures clear separation between user interaction, data processing, and storage management. The system design emphasizes reusability, security, and scalability, making it easier to extend with additional features in the future.

3.1 System Architecture

The system follows a layered structure, where each layer performs a specific function to ensure modularity and simplicity.



- **User Interface (Console):** Handles all user interactions and menu selections.
- **Bank Class Methods:** Contain the logic for various banking operations.
- **File Operations:** Manage reading, writing, and updating account information.
- **Data Storage:** Each account is stored as a text file, and a central database file maintains a list of all accounts.

This architecture allows for smooth interaction between users and stored data while maintaining data persistence through file handling.

3.2 Class Structure

The system is built around a single class — **Bank** — which encapsulates all functionalities of the banking system.

Private Data Members

Variable	Description
name	Stores the customer's full name
accNo	Unique account number for identification
balance	Holds the current account balance
debitCard	Boolean variable indicating debit card status
creditCard	Boolean variable indicating credit card status
loanAmount	Tracks the outstanding loan amount
pin	Stores the user's security PIN for authentication

Private Methods

Method	Description
encrypt()	Encrypts the user's PIN using Caesar Cipher for secure storage
decrypt()	Decrypts the stored PIN for login authentication
updateMasterFile()	Updates the centralized database file (BankDatabase.txt) with the latest account details

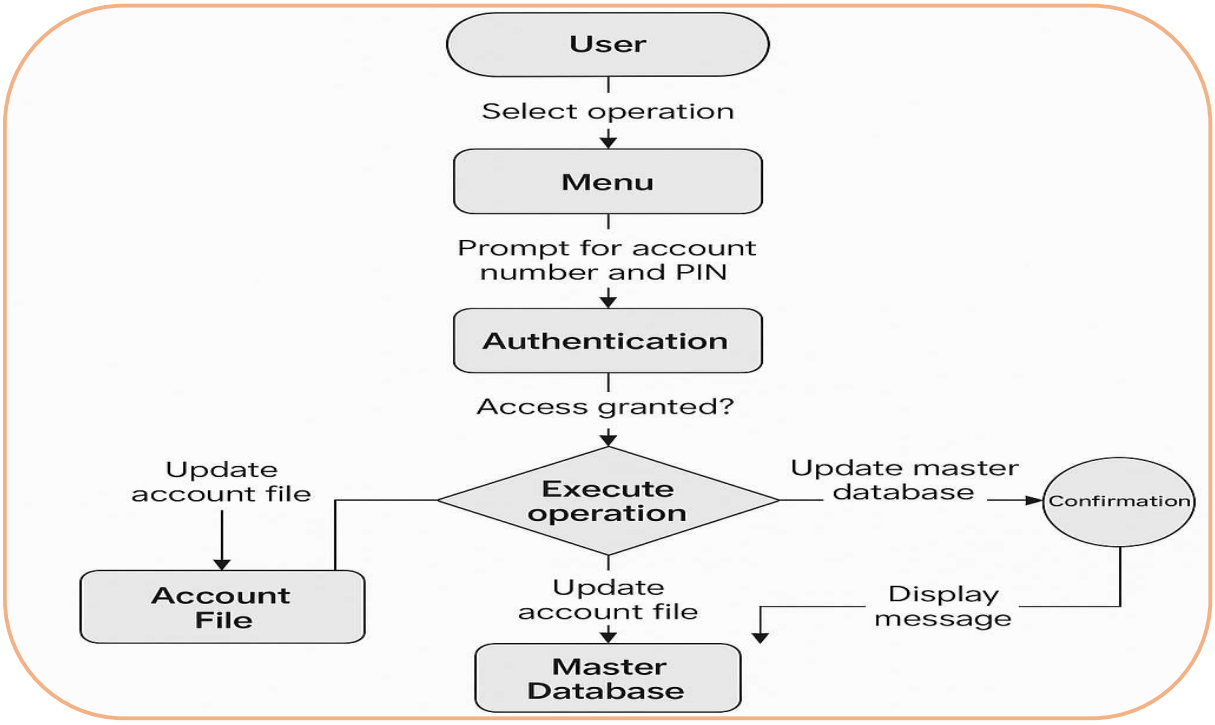
Public Methods

Method	Description
createAccount()	Creates a new user account and initializes all details
loadAccount()	Loads existing account information from the file
checkBalance()	Displays the current account balance
addCash()	Deposits a specified amount into the account
withdraw()	Withdraws funds if sufficient balance is available
applyDebitCard()	Applies for a debit card (if not already issued)
applyCreditCard()	Applies for a credit card (if not already issued)
takeLoan()	Allows the user to apply for a loan and adds the amount to the balance
payLoan()	Enables repayment of outstanding loan amount
showDetails()	Displays complete account details including cards and loans
saveToFile()	Saves all current account details to a file permanently
accountExists()	Checks whether an account with the entered number already exists

3.3 Data Flow Diagram (DFD)

The data flow of the system can be summarized as follows:

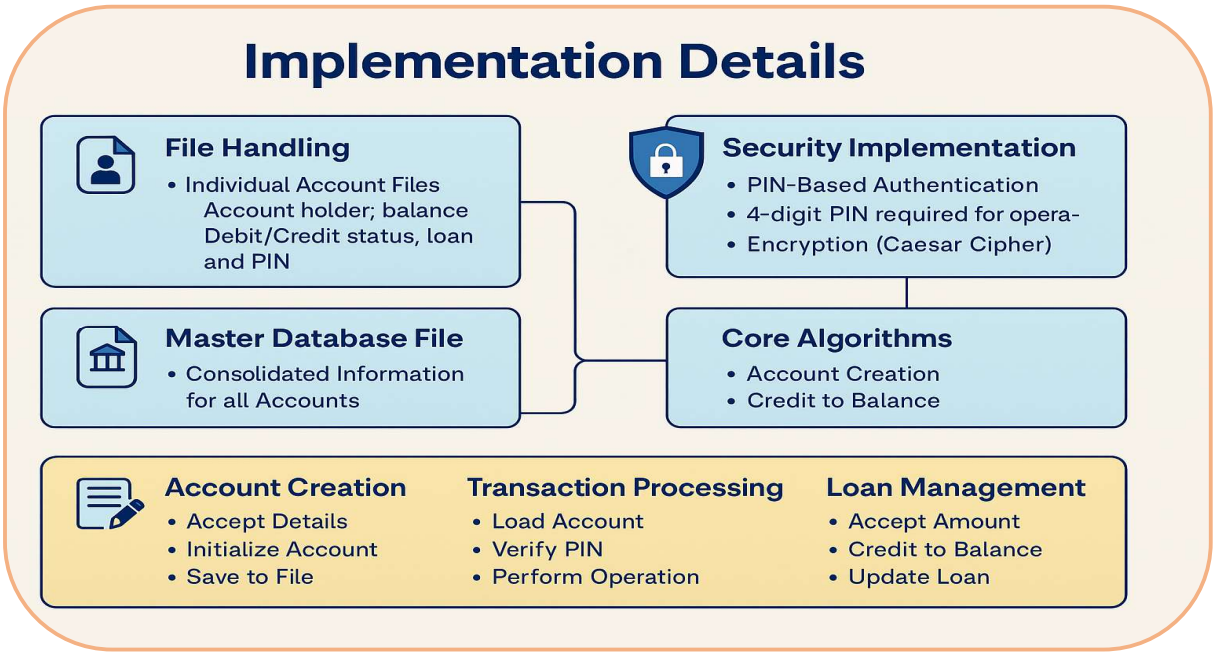
1. The user selects an operation from the console-based menu.
2. The system prompts for the **account number** and **PIN** for authentication.
3. The entered PIN is verified using the decryption process.
4. Upon successful authentication, the requested operation (deposit, withdraw, loan, etc.) is executed.
5. The system updates the account file (<accNo>.txt) with new data.
6. The **master database file (BankDatabase.txt)** is updated to reflect the latest changes.
7. A confirmation message is displayed to the user.



This flow ensures that each transaction or update is both **secure and persistent**, reflecting real-world banking mechanisms.

4. IMPLEMENTATION DETAILS

The implementation of the **Online Banking System** focuses on achieving data persistence, security, and real-time transaction management using C++ programming principles. The system integrates **object-oriented programming**, **file handling**, and **encryption** to simulate real-world banking operations.



4.1 File Handling

The system employs **two primary types of files** to ensure data persistence and maintain an organized structure for account management.

1. Individual Account Files

- Each customer account is stored in a separate text file named as {accountNumber}.txt.
- These files contain all the essential details of a single account, making it easier to access, modify, and secure specific user information.

File Structure Example:

```
Name
Account Number
Current Balance
Debit Card Status (0/1)
Credit Card Status (0/1)
Loan Amount
Encrypted PIN
```

Each time a user performs an operation (deposit, withdrawal, loan, etc.), the corresponding account file is read, updated, and rewritten with the latest data.

2. Master Database File (BankDatabase.txt)

- This file acts as a **centralized record** for all customer accounts.
- It contains summarized information such as name, account number, balance, loan amount, and card status.
- The master database is updated automatically after every account modification, serving as a permanent reference for the bank.

4.2 Security Implementation

To ensure that user information and transactions remain secure, the system implements **PIN-based authentication** and **basic encryption mechanisms**.

PIN-Based Authentication

- Each user is required to create a **4-digit PIN** during account registration.
- The PIN must be entered before performing any operation, ensuring secure access to the account.
- All account operations (deposit, withdrawal, loan, etc.) are accessible only after successful authentication.

Encryption Mechanism (Caesar Cipher)

To prevent unauthorized access to PIN data, the system applies a **Caesar Cipher** with a shift of **3** for encryption and decryption.

- **Encryption Process:**
  Each character in the PIN is shifted **forward by 3 ASCII positions** before being stored in the file.
  Example:
  - Input: 1234
  - Encrypted: 4567
- **Decryption Process:**
  When a user logs in, the stored encrypted PIN is **shifted backward by 3 positions** to retrieve the original PIN for verification.

This simple encryption ensures that user credentials are not stored in plain text, thereby enhancing data security.

4.3 Core Algorithms

1. Account Creation Algorithm

Steps:

1. Accept user details — name, account number, and PIN.
2. Check whether the entered account number already exists.
3. Initialize default values:
   - balance = 0
   - debitCard = false
   - creditCard = false
   - loanAmount = 0
4. Encrypt the entered PIN using the Caesar Cipher method.
5. Save account details in an individual file {accountNumber}.txt.
6. Update the central database file (BankDatabase.txt) with the new account record.

2. Transaction Processing Algorithm

Steps:

1. Prompt user for account number and PIN.
2. Load account data from the corresponding text file.

3. Verify the entered PIN with the decrypted value.
4. Perform the requested transaction:
    - **Deposit:** Add amount to balance.
    - **Withdrawal:** Check sufficient balance before deducting amount.
5. Update the modified balance and save changes to both the account file and the master database.

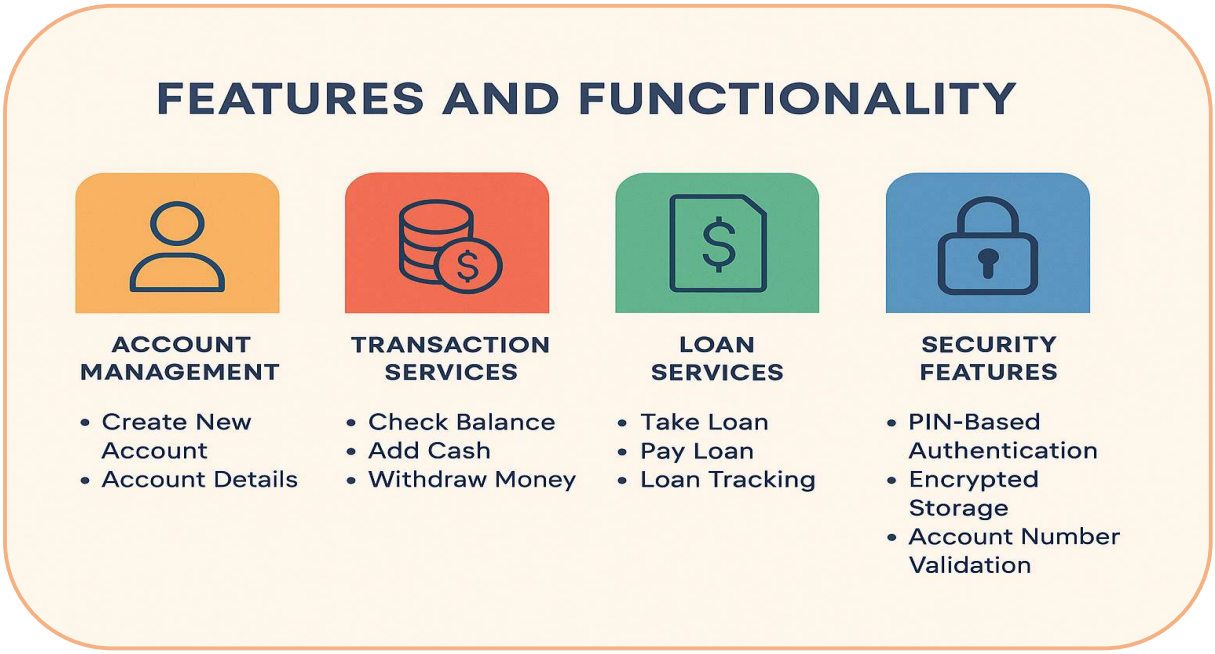
3. Loan Management Algorithm

Steps:

1. Accept desired loan amount from the user.
2. Add the loan amount to the outstanding loan balance.
3. Credit the same amount to the current account balance.
4. Save updated data to both the individual account file and the master database.
5. For loan repayment, deduct the payment amount from both balance and loan fields accordingly.

5. FEATURES AND FUNCTIONALITY

The **Online Banking System** provides a comprehensive set of features that closely resemble real-world banking operations. It integrates account management, transaction handling, card services, and loan management into a secure and user-friendly console application.



5.1 Account Management

The system allows users to manage their banking accounts effectively through the following features:

- **Create New Account:**
  Users can create a new account by entering their personal details, selecting a unique account number, and setting a secure 4-digit PIN.
- **Account Details:**
  Displays complete information including the customer's name, balance, debit/credit card status, and outstanding loan details.
- **Centralized Storage:**
  All account details are securely stored in individual text files, while a central database file (BankDatabase.txt) maintains an overview of all accounts.

5.2 Transaction Services

The system provides essential transaction functionalities for smooth and secure financial operations.

- **Check Balance:**
  Allows users to check their current balance after PIN authentication.
- **Add Cash (Deposit):**
  Enables customers to deposit any amount into their account, automatically updating the balance in both the individual file and the master database.
- **Withdraw Money:**
  Allows users to withdraw money with proper balance validation to prevent overdrafts.

5.3 Card Services

The application includes digital simulation of debit and credit card services.

- **Debit Card Application:**
  Users can apply for a debit card facility if not already issued.
- **Credit Card Application:**
  Enables eligible users to apply for a credit card service.
- **Duplicate Prevention:**
  The system ensures that a user cannot apply for the same card service more than once.

5.4 Loan Services

To replicate real banking behavior, the system offers a loan management module for users.

- **Take Loan:**
  Allows users to apply for a loan, with the approved amount immediately credited to their account balance.
- **Pay Loan:**
  Enables users to repay their loans either partially or fully, depending on the available account balance.
- **Loan Tracking:**
  Displays the total outstanding loan amount for transparent and easy monitoring.

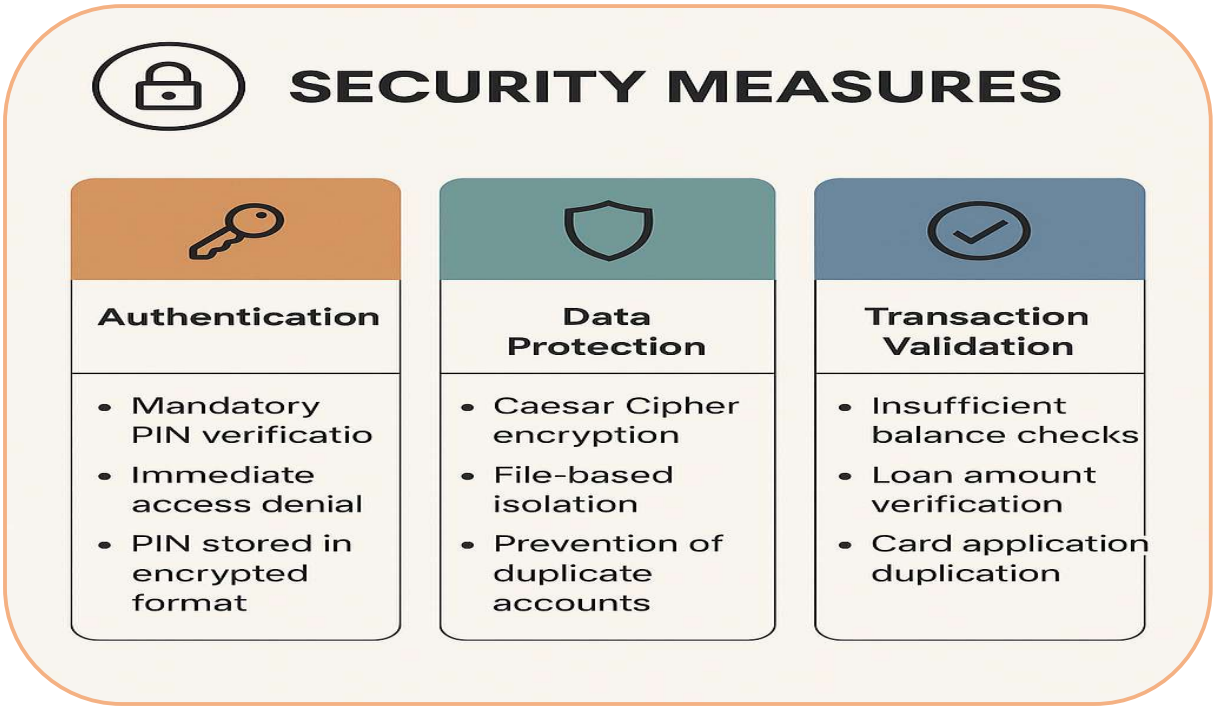
5.5 Security Features

Security is a core focus of the system's design, ensuring safe access to user data and transactions.

- **PIN-Based Authentication:**
  All banking operations require a valid 4-digit PIN to access or modify account data.
- **Encrypted Data Storage:**
  User PINs are stored in encrypted form using a **Caesar Cipher** algorithm to protect sensitive information.
- **Duplicate Account Protection:**
  The system validates each account number to prevent multiple accounts with the same ID.

6. SECURITY MEASURES

Security is a critical aspect of any banking system, and the **Online Banking System** is designed with multiple layers of protection to ensure that all transactions and user data remain secure. The system employs authentication, encryption, and validation mechanisms to safeguard sensitive information and maintain operational integrity.



6.1 Authentication

To ensure that only authorized users can access or modify account information, the system uses a **PIN-based authentication mechanism**.

- **Mandatory PIN Verification:**
  Every transaction or account-related operation requires the user to enter a valid 4-digit PIN.
- **Access Control:**
  If an incorrect PIN is entered, the system immediately denies access and prevents any unauthorized activity.
- **Encrypted PIN Storage:**
  The user's PIN is never stored in plain text. It is encrypted before being written to the account file, ensuring data confidentiality.

This approach prevents unauthorized access to account information and provides a secure gateway for all user interactions.

6.2 Data Protection

The system ensures strong data protection and storage integrity through file-level security and encryption techniques.

- **Caesar Cipher Encryption:**
  The system applies the **Caesar Cipher algorithm** with a shift of 3 to encrypt and decrypt PIN data, preventing exposure of sensitive credentials.
- **File-Based Isolation:**
  Each user's account details are stored in a separate file ({accountNumber}.txt), minimizing data overlap and ensuring better privacy.
- **Duplicate Account Prevention:**
  Before creating a new account, the system verifies if the account number already exists, thereby preventing duplicate or conflicting entries.

Together, these mechanisms guarantee secure, organized, and protected data storage for all customers.

6.3 Transaction Validation

Transaction-level security is implemented to ensure correctness and prevent financial inconsistencies during operations.

- **Balance Validation:**
  The system checks available balance before approving withdrawal or loan repayment requests to prevent overdrafts.
- **Loan Verification:**
  During loan repayment, the system ensures that users do not pay more than the outstanding amount.

- **Card Application Validation:**
  Duplicate debit or credit card requests are restricted, ensuring that each customer can only have one of each type.

These transaction validation checks maintain financial accuracy, prevent misuse, and ensure reliability across all banking operations.

7. TESTING AND RESULTS

The **Online Banking System** was tested comprehensively to ensure that all features and operations function correctly and securely. The testing phase focused on verifying both **functional requirements** (such as transactions and account creation) and **security requirements** (like authentication and data validation).

All test cases were executed using valid and invalid input scenarios to assess system robustness and error handling capabilities.

7.1 Test Cases

Test Case	Description	Input	Expected Output	Result
1. Account Creation	To verify new account creation	Name = "Ritesh" Account = "1001" PIN = "1234"	Account created successfully	✅ Pass
2. Deposit Transaction	To check deposit functionality	Account = "1001" PIN = "1234" Amount = 5000	Balance updated to ₹5000	✅ Pass
3. Withdrawal (Sufficient Balance)	To validate withdrawal with available funds	Account = "1001" PIN = "1234" Amount = 2000	Withdrawal successful Balance = ₹3000	✅ Pass
4. Withdrawal (Insufficient Balance)	To verify system response for low balance	Account = "1001" PIN = "1234" Amount = 10000	Insufficient balance error	✅ Pass
5. PIN Authentication Failure	To test access security	Account = "1001" PIN = "0000"	Access denied due to incorrect PIN	✅ Pass
6. Loan Application and Repayment	To check loan issue and payment process	Loan Amount = 10000 Payment = 5000	Loan balance updated to ₹5000	✅ Pass

7.2 Results Summary

All defined test cases executed successfully, confirming that the **Online Banking System** performs as intended under both normal and edge conditions.

The following results were observed:

- The system accurately handled **account creation, deposits, and withdrawals**.
- Proper **balance validation** prevented overdrafts and incorrect deductions.
- **PIN-based authentication** effectively restricted unauthorized access.
- **Loan processing** and **repayment functions** maintained correct outstanding balances.
- All input/output operations through **file handling** executed without data corruption or loss.

Hence, the system is **functionally reliable, secure, and stable** for all tested operations.

8. IMPLEMENTATION CODE

```
#include <iostream>

#include <fstream>

#include <string>

using namespace std;


class Bank {

private:

   string name, accNo;

   float balance;

   bool debitCard, creditCard;

   float loanAmount;

   string pin; // Security PIN


   // Encrypt PIN (simple Caesar Cipher)

   string encrypt(string p) {

      for (char &c : p) c += 3;
```

```
    return p;
}

// Decrypt PIN
string decrypt(string p) {
    for (char &c : p) c -= 3;
    return p;
}

// Update permanent bank database file
void updateMasterFile() {
    // First, read all existing data (to remove duplicate entries for this account)
    ifstream fin("BankDatabase.txt");
    string line, allData = "";
    bool updated = false;

    while (getline(fin, line)) {
        if (line.find("Account No: " + accNo) == string::npos) {
            allData += line + "\n";
        }
    }
    fin.close();

    // Add or update this account's record
    allData += "Name: " + name +
               " | Account No: " + accNo +
```

```
                " | Balance: ₹" + to_string(balance) +
                " | Loan: ₹" + to_string(loanAmount) +
                " | Debit: " + (debitCard ? "Yes" : "No") +
                " | Credit: " + (creditCard ? "Yes" : "No") + "\n";

        // Save updated data back to master file
        ofstream fout("BankDatabase.txt");
        fout << allData;
        fout.close();
    }

public:
    Bank() {
        balance = 0;
        debitCard = false;
        creditCard = false;
        loanAmount = 0;
    }

    bool accountExists(string ac) {
        ifstream fin(ac + ".txt");
        return fin.good();
    }

    void createAccount() {
        cout << "\nEnter Name: ";
```

```
    cin.ignore();
    getline(cin, name);

    cout << "Enter Account Number: ";
    cin >> accNo;

    if (accountExists(accNo)) {
        cout << "\n⚠ Account number exists! Try another.\n";
        return;
    }

    cout << "Set 4-digit PIN: ";
    cin >> pin;

    balance = 0;
    debitCard = false;
    creditCard = false;
    loanAmount = 0;

    saveToFile();
    updateMasterFile(); // Update central database
    cout << "\n ✅ Account Created Successfully & Secured with PIN!\n";
}

void saveToFile() {
    ofstream fout(accNo + ".txt");
```

```
    fout << name << endl;
    fout << accNo << endl;
    fout << balance << endl;
    fout << debitCard << endl;
    fout << creditCard << endl;
    fout << loanAmount << endl;
    fout << encrypt(pin) << endl;
    fout.close();
}

bool loadAccount(bool askPin = true) {
    cout << "\nEnter Account Number: ";
    cin >> accNo;

    ifstream fin(accNo + ".txt");
    if (!fin) {
        cout << "\n❌ Account not found!\n";
        return false;
    }

    fin >> ws;
    getline(fin, name);
    fin >> accNo >> balance >> debitCard >> creditCard >> loanAmount;
    string encryptedPin;
    fin >> encryptedPin;
    fin.close();
```

```
    pin = decrypt(encryptedPin);

    if (askPin) {
        string enteredPin;
        cout << "Enter PIN: ";
        cin >> enteredPin;
        if (enteredPin != pin) {
            cout << "\n❌ Incorrect PIN! Access Denied.\n";
            return false;
        }
    }
    return true;
}

void checkBalance() {
    if (loadAccount()) {
        cout << "\n👤 Account Holder: " << name;
        cout << "\n💰 Balance: ₹" << balance << "\n";
    }
}

void addCash() {
    if (loadAccount()) {
        float amt;
        cout << "\nEnter amount to deposit: ";
```

```
        cin >> amt;
        balance += amt;
        saveToFile();
        updateMasterFile();
        cout << "\n✅ Amount Added Successfully!\n";
    }
}

void withdraw() {
    if (loadAccount()) {
        float amt;
        cout << "\nEnter amount to withdraw: ";
        cin >> amt;
        if (amt > balance) {
            cout << "\n❌ Insufficient Balance!\n";
        } else {
            balance -= amt;
            saveToFile();
            updateMasterFile();
            cout << "\n✅ Withdrawal Successful!\n";
        }
    }
}

void applyDebitCard() {
    if (loadAccount()) {
```

```
        if (!debitCard) {
            debitCard = true;
            saveToFile();
            updateMasterFile();
            cout << "\n✅ Debit Card Applied Successfully!\n";
        } else {
            cout << "\n⚠ You already have a debit card!\n";
        }
    }
}

void applyCreditCard() {
    if (loadAccount()) {
        if (!creditCard) {
            creditCard = true;
            saveToFile();
            updateMasterFile();
            cout << "\n✅ Credit Card Applied Successfully!\n";
        } else {
            cout << "\n⚠ You already have a credit card!\n";
        }
    }
}

void takeLoan() {
    if (loadAccount()) {
```

```
        float amt;
        cout << "\nEnter loan amount: ";
        cin >> amt;
        loanAmount += amt;
        balance += amt;
        saveToFile();
        updateMasterFile();
        cout << "\n✅ Loan Approved! Amount added to balance.\n";
    }
}

void payLoan() {
    if (loadAccount()) {
        if (loanAmount <= 0) {
            cout << "\nℹ No outstanding loan to pay!\n";
            return;
        }

        float amt;
        cout << "\nEnter amount to pay for loan: ";
        cin >> amt;

        if (amt > balance) {
            cout << "\n❌ Insufficient Balance to pay loan!\n";
        } else if (amt > loanAmount) {
            cout << "\n⚠ Amount exceeds loan amount! Paying only required amount.\n";
```

```
            balance -= loanAmount;
            loanAmount = 0;
            saveToFile();
            updateMasterFile();
            cout << "\n✅ Loan fully paid!\n";
        } else {
            balance -= amt;
            loanAmount -= amt;
            saveToFile();
            updateMasterFile();
            cout << "\n✅ Loan Payment Successful!\n";
        }
    }
}

void showDetails() {
    if (loadAccount()) {
        cout << "\n===== ACCOUNT DETAILS =====\n";
        cout << "👤 Name: " << name << "\n";
        cout << "🏦 Account No: " << accNo << "\n";
        cout << "💰 Balance: ₹" << balance << "\n";
        cout << "💳 Debit Card: " << (debitCard ? "Yes" : "No") << "\n";
        cout << "💳 Credit Card: " << (creditCard ? "Yes" : "No") << "\n";
        cout << "📄 Loan Taken: ₹" << loanAmount << "\n";
    }
}
```

```
};

int main() {
    Bank b;
    int choice;

    do {
        cout << "\n========== 🔒 ONLINE BANKING SYSTEM ==========\n";
        cout << "1. Create New Account\n";
        cout << "2. Check Balance\n";
        cout << "3. Add Cash\n";
        cout << "4. Withdraw Money\n";
        cout << "5. Apply Debit Card\n";
        cout << "6. Apply Credit Card\n";
        cout << "7. Take Loan\n";
        cout << "8. Pay Loan\n";
        cout << "9. Account Details\n";
        cout << "10. Exit\n";
        cout << "Enter option: ";
        cin >> choice;

        switch (choice) {
        case 1: b.createAccount(); break;
        case 2: b.checkBalance(); break;
        case 3: b.addCash(); break;
        case 4: b.withdraw(); break;
```

```
        case 5: b.applyDebitCard(); break;
        case 6: b.applyCreditCard(); break;
        case 7: b.takeLoan(); break;
        case 8: b.payLoan(); break;
        case 9: b.showDetails(); break;
        case 10: cout << "\n🙏 Thank You for Using Secure Online Banking!\n"; break;
        default: cout << "\n❌ Invalid option!\n";
        }
    } while (choice != 10);

    return 0;
}
```

9. OUTPUT

MAIN MENU:

```
========== 🔒ONLINE BANKING SYSTEM ==========
1. Create New Account
2. Check Balance
3. Add Cash
4. Withdraw Money
5. Apply Debit Card
6. Apply Credit Card
7. Take Loan
8. Pay Loan
9. Account Details
10. Exit
Enter option:
```

1. CREATE NEW ACCOUNT:

```
Enter option: 1

Enter Name: Ritesh
Enter Account Number: 123456
Set 4-digit PIN: 1235

✅Account Created Successfully & Secured with PIN!
```

2. CHECK BALANCE:

```
Enter option: 2

Enter Account Number: 123456
Enter PIN: 1235

👤Account Holder: Ritesh
💰Balance: ₹0
```

3. ADD CASH :

```
Enter option: 3

Enter Account Number: 123456
Enter PIN: 1235

Enter amount to deposit: 1000

✅Amount Added Successfully!
```

4. WITHDRAW CASH:

```
Enter option: 4

Enter Account Number: 123456
Enter PIN: 1235

Enter amount to withdraw: 500

✅Withdrawal Successful!
```

5. APPLY DEBIT CARD:

```
Enter option: 5

Enter Account Number: 123456
Enter PIN: 1235

✅Debit Card Applied Successfully!
```

6. APPLY CREDIT CARD:

```
Enter option: 6

Enter Account Number: 123456
Enter PIN: 1235

✅Credit Card Applied Successfully!
```

7. TAKE LOAN:

```
Enter option: 7

Enter Account Number: 123456
Enter PIN: 1235

Enter loan amount: 5000

✅Loan Approved! Amount added to balance.
```

8. PAY LOAN:

```
Enter option: 8

Enter Account Number: 123456
Enter PIN: 1235

Enter amount to pay for loan: 5000

✅Loan Payment Successful!
```

9. ACCOUNT DETAILS :

```
Enter option: 9

Enter Account Number: 123456
Enter PIN: 1235

===== ACCOUNT DETAILS =====
🧍Name: Ritesh
🏦Account No: 123456
💰Balance: ₹500
💳Debit Card: Yes
💳Credit Card: Yes
📄Loan Taken: ₹0
```

10. EXIT:

```
Enter option: 10

🙏Thank You for Using Secure Online Banking!


...Program finished with exit code 0
Press ENTER to exit console.
```

11. SPECIFIC FILE WHICH CONTAIN SPECIFIC ACCOUNT INFORMATION:

main.cpp | 123456.txt | BankDatabase.txt

```
1  Ritesh
2  123456
3  500
4  1
5  1
6  0
7  4568
8
```

12. BANK DATABASE FILE :

main.cpp | 123456.txt | BankDatabase.txt

```
1  Name: Ritesh | Account No: 123456 | Balance: ₹500.000000 | Loan: ₹0.000000 | Debit: Yes | Credit: Yes
2
```

10. CONCLUSION

The **Online Banking System** successfully demonstrates the design and implementation of a secure, modular, and efficient banking application using the **C++ programming language**. The project achieves its key objectives of providing essential banking functionalities — including **account management, transactions, card services, and loan operations** — while ensuring data security through **PIN-based authentication** and **encryption mechanisms**.

By integrating **object-oriented programming concepts**, **file handling**, and **basic cryptography**, the system showcases the practical application of theoretical computer science principles in a real-world financial context. It offers users a simple yet robust interface for managing their banking activities, ensuring both functionality and user-friendliness.

While the current version focuses on core operations within a console-based environment, it provides a strong foundation for future enhancements such as **graphical user interfaces (GUI)**, **database integration**, **multi-user networking**, and **real-time transaction systems**.

The development of this project has provided valuable insights into:

- The importance of **secure software design** in financial systems
- The implementation of **data persistence** using file handling
- The role of **user authentication and encryption** in protecting sensitive information
- The overall **software development lifecycle**, from design to testing

This project has been a significant learning experience, enhancing both technical and analytical skills. It establishes a solid groundwork for building more advanced, scalable, and secure banking systems in the future.

11. REFERENCES

1. Bjarne Stroustrup, *"The C++ Programming Language,"* 4th Edition, Addison-Wesley, 2013.
2. Herbert Schildt, *"C++: The Complete Reference,"* 4th Edition, McGraw-Hill Education, 2003.
3. E. Balagurusamy, *"Object-Oriented Programming with C++,"* 8th Edition, McGraw Hill Education, 2019.
4. William Stallings, *"Cryptography and Network Security: Principles and Practice,"* 7th Edition, Pearson, 2017.
5. GeeksforGeeks, *"File Handling in C++,"* Available at: https://www.geeksforgeeks.org/file-handling-c-classes/
6. CPlusPlus.com, *"Input/Output with Files,"* Available at: http://www.cplusplus.com/doc/tutorial/files/