PASSWORD MANAGER

A PROJECT REPORT

Submitted by

PRASHANTH REDDY C

NOVEMBER - 2024

ABSTRACT

The Password Manager application is a secure tool designed to store and retrieve passwords for various user accounts. Using Java and the AES encryption algorithm, this application ensures that all stored passwords are encrypted and protected from unauthorized access. The main goal of the application is to simplify password management while maintaining high levels of security. The application provides users with the ability to securely store multiple accounts' details, including usernames, passwords, and account names, and to retrieve this information when needed. The project utilizes file handling to save encrypted data and ensures that only authorized users, authenticated with a master password, can access the stored credentials.

INTRODUCTION

In the modern world, managing multiple online accounts is an essential but often overlooked task. Users are required to remember numerous passwords, many of which are difficult to store securely. A password manager solves this issue by offering a secure, centralized location for storing passwords. The Password Manager application, built with Java, uses AES encryption to protect sensitive data, ensuring that user credentials remain safe from unauthorized access. This project aims to provide a user-friendly interface to store, retrieve, and delete account information while maintaining strong security practices.

OBJECTIVE

The primary objectives of this project are:

- To develop a secure password manager that allows users to safely store, retrieve, and manage passwords.
- To implement AES encryption for securing passwords in the application.
- To provide a user-friendly interface for interacting with the stored passwords.
- To ensure that only authorized users, authenticated via a master password, can access the stored data.
- To implement basic features like adding, deleting, and viewing stored passwords, and handling errors such as incorrect login attempts.

LITERATURE REVIEW

Password management has been an essential area in the field of cybersecurity. With the increase in online accounts and services, users often find it difficult to manage and remember their passwords. Several studies and projects highlight the importance of encrypted storage systems to protect sensitive data. Existing password management tools use advanced encryption techniques such as AES (Advanced Encryption Standard) to safeguard passwords from being easily accessed or leaked. Tools such as LastPass, Dashlane, and 1Password are widely used for this purpose. This project aims to create a simplified password manager that focuses on user experience and robust security measures, leveraging encryption techniques to protect user data.

SYSTEM DESIGN AND IMPLEMENTATION

Architecture

The architecture of the Password Manager application consists of two primary components:

- Front-End (User Interface): The UI is developed using Java Swing, which provides a graphical interface for users to interact with the application. It allows users to add, view, and delete passwords in a secure manner.
- Back-End (Logic): The backend handles password encryption and decryption, user authentication, and file management. Passwords are securely stored in an encrypted format in a local file, and decryption occurs only when the user provides the correct master password.

Components

- Master Password: A master password is required to unlock the application. This password is never stored but is used for encrypting and decrypting stored passwords.
- Encryption: AES encryption is used to encrypt the passwords before storing them in the local file.
- File Handling: All account data is saved in a file named accounts.data, which contains encrypted account details.
- User Interface: A simple interface with buttons to add, view, and delete account details, along with a display field to show the selected account's information.

CODE

Main.java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your master password: ");
        String masterPassword = scanner.nextLine();

        PasswordManager passwordManager = new PasswordManager(masterPassword);

        if (passwordManager.isAuthenticated()) {
            System.out.println("\n--- Welcome to Password Manager ---\n");
            while (true) {
                System.out.println("1. Add Account");
                System.out.println("2. View All Accounts");
                System.out.println("3. Retrieve Account Password");
                System.out.println("4. Delete Account");
                System.out.println("5. Exit");
                System.out.print("Choose an option: ");
                int choice = scanner.nextInt();
                scanner.nextLine(); // Consume newline

                switch (choice) {
                    case 1:
                        System.out.print("Enter account name: ");
                        String accountName = scanner.nextLine();
                        System.out.print("Enter username: ");
                        String username = scanner.nextLine();
                        System.out.print("Enter password: ");
                        String password = scanner.nextLine();
                        passwordManager.addAccount(accountName, username, password);
                        break;
                    case 2:
                        passwordManager.viewAllAccounts();
                        break;
                    case 3:
                        System.out.print("Enter account name to retrieve: ");
                        String retrieveAccount = scanner.nextLine();
                        passwordManager.retrievePassword(retrieveAccount);
                        break;
                    case 4:
                        System.out.print("Enter account name to delete: ");
```

```
                String deleteAccount = scanner.nextLine();
                passwordManager.deleteAccount(deleteAccount);
                break;
            case 5:
                System.out.println("Exiting... Goodbye!");
                System.exit(0);
                break;
            default:
                System.out.println("Invalid choice! Please try again.");
        }
      }
    } else {
        System.out.println("Authentication failed. Exiting...");
    }

    scanner.close();
  }
}
```

PasswordManager.java

```
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.ArrayList;

class NoteAppFrame extends JFrame {
  private final JTextArea noteArea;
  private final DefaultListModel<String> notesListModel;
  private final JList<String> notesList;
  private final ArrayList<File> notesFiles;
  private final File notesDirectory;
  public NoteAppFrame() {
    // Configure Frame
    setTitle("Note-Taking Application");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(800, 600);
    setLayout(new BorderLayout());
    // Notes Storage Directory
    notesDirectory = new File("Notes");
    if (!notesDirectory.exists()) {
      notesDirectory.mkdir();
    }
    // Notes List Panel
    notesListModel = new DefaultListModel<>();
    notesList = new JList<>(notesListModel);
```

```
        notesList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        notesFiles = new ArrayList<>();
        loadNotes();
        notesList.addListSelectionListener(e -> {
            if (!e.getValueIsAdjusting()) {
                loadSelectedNote();
            }
        });
        JScrollPane notesScrollPane = new JScrollPane(notesList);
        notesScrollPane.setPreferredSize(new Dimension(200, getHeight()));
        add(notesScrollPane, BorderLayout.WEST);

        // Note Area
        noteArea = new JTextArea();
        JScrollPane noteScrollPane = new JScrollPane(noteArea);
        add(noteScrollPane, BorderLayout.CENTER);
        // Buttons Panel
        JPanel buttonsPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        add(buttonsPanel, BorderLayout.NORTH);
        JButton newButton = new JButton("New Note");
        newButton.addActionListener(e -> createNewNote());
        buttonsPanel.add(newButton);

        JButton saveButton = new JButton("Save Note");
        saveButton.addActionListener(e -> saveCurrentNote());
        buttonsPanel.add(saveButton);

        JButton deleteButton = new JButton("Delete Note");
        deleteButton.addActionListener(e -> deleteSelectedNote());
        buttonsPanel.add(deleteButton);

        setVisible(true);
    }

    private void loadNotes() {
        notesListModel.clear();
        notesFiles.clear();
        for (File file : notesDirectory.listFiles((dir, name) -> name.endsWith(".txt"))) {
            notesFiles.add(file);
            notesListModel.addElement(file.getName().replace(".txt", ""));
        }
    }

    private void loadSelectedNote() {
        int index = notesList.getSelectedIndex();
        if (index != -1) {
            File selectedFile = notesFiles.get(index);
```

```
        try (BufferedReader reader = new BufferedReader(new FileReader(selectedFile))) {
            noteArea.setText("");
            String line;
            while ((line = reader.readLine()) != null) {
                noteArea.append(line + "\n");
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "Error loading note: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }
}

private void createNewNote() {
    noteArea.setText("");
    notesList.clearSelection();
}

private void saveCurrentNote() {
    String noteTitle = JOptionPane.showInputDialog(this, "Enter Note Title:");
    if (noteTitle == null || noteTitle.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Note title cannot be empty.", "Error",
JOptionPane.ERROR_MESSAGE);
        return;
    }

    File noteFile = new File(notesDirectory, noteTitle + ".txt");
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(noteFile))) {
        writer.write(noteArea.getText());
        loadNotes();
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Error saving note: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
    }
}

private void deleteSelectedNote() {
    int index = notesList.getSelectedIndex();
    if (index != -1) {
        File selectedFile = notesFiles.get(index);
        if (selectedFile.delete()) {
            loadNotes();
            noteArea.setText("");
        } else {
            JOptionPane.showMessageDialog(this, "Error deleting note.", "Error",
JOptionPane.ERROR_MESSAGE);
        }
```

```
        } else {
            JOptionPane.showMessageDialog(this, "No note selected.", "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }
} import java.util.HashMap;
import java.util.Map;

public class PasswordManager {
    private final String masterPassword;
    private final Map<String, Account> accounts;
    private final FileHandler fileHandler;
    private boolean authenticated;

    public PasswordManager(String masterPassword) {
        this.masterPassword = masterPassword;
        this.accounts = new HashMap<>();
        this.fileHandler = new FileHandler();
        this.authenticated = authenticate();
        if (authenticated) {
            loadAccounts();
        }
    }

    private boolean authenticate() {
        String savedHash = fileHandler.getMasterPasswordHash();
        return savedHash == null || savedHash.equals(EncryptionUtils.hashPassword(masterPassword));
    }

    public boolean isAuthenticated() {
        return authenticated;
    }

    public void addAccount(String accountName, String username, String password) {
        String encryptedPassword = EncryptionUtils.encrypt(password, masterPassword);
        Account account = new Account(accountName, username, encryptedPassword);
        accounts.put(accountName, account);
        saveAccounts();
        System.out.println("Account added successfully!");
    }

    public void viewAllAccounts() {
        if (accounts.isEmpty()) {
            System.out.println("No accounts available.");
            return;
        }
```

```
        for (Account account : accounts.values()) {
            System.out.println(account);
        }
    }

    public void retrievePassword(String accountName) {
        Account account = accounts.get(accountName);
        if (account == null) {
            System.out.println("Account not found!");
            return;
        }

        String decryptedPassword = EncryptionUtils.decrypt(account.getPassword(), masterPassword);
        System.out.println("Password for " + accountName + ": " + decryptedPassword);
    }

    public void deleteAccount(String accountName) {
        if (accounts.remove(accountName) != null) {
            saveAccounts();
            System.out.println("Account deleted successfully!");
        } else {
            System.out.println("Account not found!");
        }
    }

    private void saveAccounts() {
        fileHandler.saveAccounts(accounts);
    }

    private void loadAccounts() {
        Map<String, Account> savedAccounts = fileHandler.loadAccounts();
        if (savedAccounts != null) {
            accounts.putAll(savedAccounts);
        }
    }
}
```

FileHandler.java

```
import java.io.*;
import java.util.HashMap;
import java.util.Map;

public class FileHandler {
    private static final String DATA_FILE = "data/accounts.data";
    private static final String PASSWORD_FILE = "data/master_password.data";

    @SuppressWarnings("unchecked")
    public Map<String, Account> loadAccounts() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(DATA_FILE))) {
            return (Map<String, Account>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            return new HashMap<>();
        }
    }

    public void saveAccounts(Map<String, Account> accounts) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(DATA_FILE))) {
            oos.writeObject(accounts);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public String getMasterPasswordHash() {
        try (BufferedReader reader = new BufferedReader(new FileReader(PASSWORD_FILE))) {
            return reader.readLine();
        } catch (IOException e) {
            return null;
        }
    }
}
```

EncryptionUtils.java

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.security.MessageDigest;
import java.util.Base64;

public class EncryptionUtils {
    private static final String ALGORITHM = "AES";

    public static String encrypt(String data, String key) {
        try {
            SecretKeySpec secretKey = getKey(key);
            Cipher cipher = Cipher.getInstance(ALGORITHM);
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return Base64.getEncoder().encodeToString(cipher.doFinal(data.getBytes()));
        } catch (Exception e) {
            throw new RuntimeException("Encryption error", e);
        }
    }

    public static String decrypt(String data, String key) {
        try {
            SecretKeySpec secretKey = getKey(key);
            Cipher cipher = Cipher.getInstance(ALGORITHM);
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new String(cipher.doFinal(Base64.getDecoder().decode(data)));
        } catch (Exception e) {
            throw new RuntimeException("Decryption error", e);
        }
    }

    public static String hashPassword(String password) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hash = md.digest(password.getBytes());
            return Base64.getEncoder().encodeToString(hash);
        } catch (Exception e) {
            throw new RuntimeException("Hashing error", e);
        }
    }

    private static SecretKeySpec getKey(String key) {
        try {
            byte[] keyBytes = key.getBytes("UTF-8");
            MessageDigest sha = MessageDigest.getInstance("SHA-256");
            keyBytes = sha.digest(keyBytes);
```

```
            return new SecretKeySpec(keyBytes, ALGORITHM);
        } catch (Exception e) {
            throw new RuntimeException("Key generation error", e);
        }
    }
}
```

Account.java

```
import java.io.Serializable;

public class Account implements Serializable {
    private final String name;
    private final String username;
    private final String password;

    public Account(String name, String username, String password) {
        this.name = name;
        this.username = username;
        this.password = password;
    }

    public String getName() {
        return name;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    @Override
    public String toString() {
        return "Account{" +
                "name='" + name + '\'' +
                ", username='" + username + '\'' +
                '}';
    }
}
```

RESULTS

Screenshots:

```
--- Welcome to Password Manager ---

1. Add Account
2. View All Accounts
3. Retrieve Account Password
4. Delete Account
5. Exit
Choose an option: 1
Enter account name: Gmail
Enter username: user@example.com
Enter password: MyGmailPassword
Account added successfully!
1. Add Account
2. View All Accounts
3. Retrieve Account Password
4. Delete Account
5. Exit
Choose an option: 1
Enter account name: Facebook
Enter username: user2@example.com
Enter password: MyFBPassword
Account added successfully!
1. Add Account
2. View All Accounts
3. Retrieve Account Password
4. Delete Account
5. Exit
Choose an option: 2
Account{name='Gmail', username='user@example.com'}
Account{name='Facebook', username='user2@example.com'}
1. Add Account
2. View All Accounts
3. Retrieve Account Password
4. Delete Account
5. Exit
Choose an option: 3
Enter account name to retrieve: Gmail
Password for Gmail: MyGmailPassword
```

CONCLUSION

The Password Manager application successfully provides secure password storage and retrieval using AES encryption. The application allows users to add, view, and delete accounts with proper encryption and error handling. By leveraging Java Swing and encryption libraries, the project offers both a user-friendly interface and robust security.

REFERENCES

[1] Oracle Documentation on Java.

[2] Effective Java, 3rd Edition by Joshua Bloch

[3] Java: The Complete Reference by Herbert Schildt