



Daffodil
International
University

Capstone Password Manager Documentation

Name : Sanim Yousuf Fahim

Identification Number : 024231000534 1092

Name : Rakibul Hasan Shuvo

Identification Number : 024231000534 1177

Name : Nayeef Sarker Nafi

Identification Number : 024231000534 1222

Semester : Spring- 24

Batch : 40

Section : D

Course Code : SE-133

Course Name : Software Development Capstone Project

Course Teacher Name : Mr. Md. Abdul Hannan

Designation : Lecturer

Project Name : Capstone Password Manager

Submission Date : 03-06-24

Abstract

The Capstone Password Manager is a robust and user-friendly application designed to greatly enhance the security and management of user credentials. In today's digital age, where cyber threats are constantly evolving, ensuring robust digital security is paramount. This project offers an essential tool for securely and efficiently managing passwords, protecting sensitive information from unauthorized access.

The Capstone Password Manager includes a comprehensive range of features tailored to meet diverse user needs. It supports the seamless creation of new user accounts, secure login processes, and robust password recovery options. Additionally, it allows users to save and retrieve passwords for various websites, simplifying the user experience and promoting the use of strong, unique passwords. By employing XOR encryption, all stored passwords are securely encrypted, safeguarding user information from potential cyber threats.

Beyond these core functionalities, the application offers advanced features such as generating strong, random passwords and evaluating the strength of existing passwords. This empowers users to adopt best practices in password security, mitigating common vulnerabilities. Designed with a user-centric approach, the system balances ease of use with robust security measures, making it accessible to users of all technical backgrounds.

This documentation provides a detailed overview of the Capstone Password Manager, including configuration and operational guidelines. It also offers a technical explanation of the underlying code, covering encryption methods, data storage mechanisms, and user interaction processes. This technical insight highlights the sophisticated technology powering the application.

The Capstone Password Manager is an indispensable tool for enhancing digital security. It offers a harmonious blend of functionality, security, and ease of use, making it an ideal solution for anyone looking to improve their online security. By adopting this tool, users can significantly reduce the risk of unauthorized access and ensure their sensitive information remains protected in an increasingly digital world.

Table of contents

Chapter 1: Introduction

- 1.1 Overview of Password Management
- 1.2 Importance of Password Security
- 1.3 Objectives of the Capstone Password Manager

Chapter 2: User Guide

- 2.1 Creating an Account
- 2.2 Logging In
- 2.3 Forgot Password Functionality
- 2.4 Changing Passwords
- 2.5 Saving Website Passwords
- 2.6 Retrieving Saved Passwords
- 2.7 Generating Strong Passwords
- 2.8 Checking Password Strength

Chapter 3: Technical Overview

- 3.1 Code Structure
- 3.2 Encryption and Decryption Mechanism
- 3.3 File Handling for Account Management
- 3.4 User Interaction and Input Handling

Chapter 4: Code Explanation

- 4.1 Main Function and Program Flow
- 4.2 Account Creation Logic
- 4.3 Login Process
- 4.4 Password Recovery
- 4.5 Password Change Functionality
- 4.6 Saving and Retrieving Website Passwords
- 4.7 Password Generation and Strength Assessment

Chapter 5: Security Considerations

- 5.1 Encryption Techniques
- 5.2 Data Protection Measures
- 5.3 Best Practices for Password Security

Chapter 6: Conclusion

- 6.1 Summary of Features
- 6.2 Impact on User Security
- 6.3 Final Thoughts

Chapter 1: Introduction

1.1 Overview of Password Management

In today's digital age, passwords serve as the primary line of defense against unauthorized access to personal and sensitive information. From email accounts and social media profiles to banking and financial services, nearly every aspect of our online lives requires a password. Effective password management is crucial to maintaining security and ensuring that personal data remains confidential. The Capstone Password Manager is designed to address this need by providing users with a secure and user-friendly tool to manage their passwords efficiently.

1.2 Importance of Password Security

Password security is of paramount importance due to the increasing prevalence of cyber threats and data breaches. Weak, reused, or easily guessable passwords are often exploited by malicious actors to gain unauthorized access to accounts. This can lead to identity theft, financial loss, and significant personal inconvenience. The Capstone Password Manager emphasizes the creation and maintenance of strong, unique passwords for different accounts, thereby reducing the risk of security breaches and enhancing overall digital security.

1.3 Objectives of the Capstone Password Manager

The primary objectives of the Capstone Password Manager are as follows:

User-Friendly Interface: Provide an intuitive and easy-to-use interface for users to manage their passwords without requiring extensive technical knowledge.

Secure Storage: Implement robust encryption mechanisms to ensure that stored passwords are protected against unauthorized access.

Password Generation: Offer a feature to generate strong, random passwords that meet best practices for security.

Password Strength Assessment: Enable users to check the strength of their passwords and receive feedback to improve them.

Recovery Options: Include functionalities to recover or reset passwords in case users forget them, ensuring they are never locked out of their accounts.

Comprehensive Management: Allow users to save, retrieve, and manage passwords for various websites and services in a centralized and secure manner.

The Capstone Password Manager aims to be a comprehensive solution for individuals seeking to enhance their password security practices, offering a blend of convenience, functionality, and robust security measures.

Chapter 2: User Guide

This chapter provides a detailed user guide for the Capstone Password Manager, including instructions on how to create an account, log in, and use the various functionalities offered by the program.

2.1 Creating an Account

To create an account in the Capstone Password Manager, follow these steps:

Launch the Program:

```
Welcome to the Capstone Password Manager

1. Create Account
2. Log-in
3. Forget Password
4. Exit

Enter your choice:
```

Run the executable file for the Capstone Password Manager.
The main menu will be displayed with several options.

Select Create Account:

Enter 1 to select the option to create a new account. Press Enter.

Enter Account Details:

The program will prompt you to **enter a username**. Type your desired username and press Enter.

Next, enter your email address and press Enter.

Enter your desired password and press Enter.

Re-enter the password for confirmation and press Enter.

```
Enter your choice: 1

Enter username: user
Enter E-mail: user@gmail.com
Enter password: user12345
Enter password again: user12345
Account created successfully
```

Account Creation Confirmation:

If the passwords match, your account details will be encrypted and saved.

A confirmation message "**Account created successfully**" will be displayed.

2.2 Logging In

To log in to your account:

Launch the Program:

Run the executable file for the Capstone Password Manager.

The main menu will be displayed.

Select Log-in:

Enter 2 to select the option to log in.

Press Enter.

Enter Login Details:

Enter your username and press Enter.

Enter your email address and press Enter.

Enter your password and press Enter.

```
Enter your choice: 2  
  
Enter username: user  
Enter e-mail: user@gmail.com  
Enter password: user12345  
Log-in successful
```

Log-in Confirmation:

If the credentials are correct, you will see the message "Log-in successful".
You will be redirected to your user dashboard.

2.3 Forgot Password Functionality

If you forget your password, you can retrieve it using the following steps:

Launch the Program:

Run the executable file for the Capstone Password Manager.
The main menu will be displayed.

Select Forget Password:

Enter 3 to select the option for forgetting the password.
Press Enter.

Enter Account Details:

Enter your username and press Enter.
Enter your email address and press Enter.

```
Enter your choice: 3  
  
Enter username: user  
Enter e-mail: user@gmail.com  
Your password is: user12345
```


Password Retrieval:

If the details match, the program will display your password.
Use this password to log in.

2.4 Changing Passwords

To change your account password:

Log in to Your Account:

Follow the steps in **section 2.2** to log in to your account.

Access the Dashboard:

Once logged in, you will be in the user dashboard.

Select Change Password:

Enter 3 to select the option to change your account password.
Press Enter.

Enter New Password:

Enter your new password and press Enter.
The program will encrypt and update your password.

```
Enter your choice: 3
Enter new password: user6789
Enter new password again: user6789
Password changed successfully
```

Password Change Confirmation:

A message "Password changed successfully" will be displayed.

2.5 Saving Website Passwords

To save passwords for websites:

Log in to Your Account:

Follow the steps in **section 2.2** to log in to your account.

Access the Dashboard:

Once logged in, you will be in the user dashboard.

Select Save New Password:

Enter 1 to select the option to save a new password.

Press Enter.

Enter Website Details:

Enter the website name and press Enter.

Enter the website password and press Enter.

Re-enter the website password for confirmation and press Enter.

Enter a numerical security key and press Enter.

```
Enter your choice: 1

Enter Website name: website.com
Enter Website Password: random123
Enter Website password again: random123
Enter a security key (Numerical): 123
Password saved successfully
```

Password Save Confirmation:

A message "Password saved successfully" will be displayed.

2.6 Retrieving Saved Passwords

To retrieve passwords saved for websites:

Log in to Your Account:

Follow the steps in **section 2.2** to log in to your account.

Access the Dashboard:

Once logged in, you will be in the user dashboard.

Select Show Passwords:

Enter 2 to select the option to show passwords.

Press Enter.

Enter Security Key:

Enter the security key used when saving the password and press Enter.

```
Enter your choice: 2
Enter your security key: 123
Website: website.com, Password: random123
```

Display Saved Passwords:

The program will display all saved websites along with their decrypted passwords.

2.7 Generating Strong Passwords

To generate a strong password:

Log in to Your Account:

Follow the steps in **section 2.2** to log in to your account.

Access the Dashboard:

Once logged in, you will be in the user dashboard.

Select Generate Password:

Enter 4 to select the option to generate a password.

Press Enter.

Enter Password Length:

Enter the desired length for the password and press Enter.

```
Enter your choice: 4  
  
Enter length of password: 8  
Generated Password: vjud.iDp
```

Password Generation:

The program will generate a random, strong password and display it.

2.8 Checking Password Strength

To check the strength of a password:

Log in to Your Account:

Follow the steps in **section 2.2** to log in to your account.

Access the Dashboard:

Once logged in, you will be in the user dashboard.

Select Check Password Strength:

Enter 5 to select the option to check password strength.

Press Enter.

Enter Password:

Enter the password you want to check and press Enter.

```
Enter your choice: 5  
  
Enter password to check its strength: user6789  
Password strength: Medium
```

Display Password Strength:

The program will analyze the password and display its strength as "**Weak**", "**Medium**", or "**Strong**".

Chapter 3: Technical Overview

This chapter provides a detailed technical overview of the Capstone Password Manager, focusing on the code structure, encryption and decryption mechanisms, file handling for account management, and user interaction and input handling.

3.1 Code Structure

The Capstone Password Manager is structured into several functions, each responsible for a specific feature. The main components are as follows:

Main Function (main)

This is the entry point of the program. It displays the main menu and handles user choices to either create an account, log in, retrieve a forgotten password, or exit the program.

Account Management Functions

createAccount(): Handles user input for creating a new account, encrypts the password, and saves the account details to a file.

login(): Verifies user credentials by decrypting the stored password and comparing it with the input.

forgotPassword(): Retrieves and displays the decrypted password if the user forgets it.

chanword(): gePass Allows users to change their account password by updating the stored encrypted password.

Password Management Functions

savePassword(): Saves encrypted passwords for different websites.

showPassword(): Displays stored website passwords after decrypting them.

generatePass(): Generates strong random passwords.

passStrength(): Evaluates the strength of a given password.

Helper Functions

xorEncryptDecrypt(): Performs XOR encryption and decryption.

Password strength checking functions (hasLowerCase(), hasUpperCase(), hasDigit(),

hasSpecialChar()): Validate password complexity.

User Dashboard

dashboard(): Provides an interactive menu for logged-in users to access the password management functions.

3.2 Encryption and Decryption Mechanism

The Capstone Password Manager uses a simple XOR encryption mechanism for encrypting and decrypting passwords.

1. Encryption and Decryption Function (xorEncryptDecrypt)

This function takes a string and a key as input.

It performs XOR operation between each character of the string and the key, effectively encrypting or decrypting the string.

```
1  string xorEncryptDecrypt(const string& str, char key) {
2      string result = str;
3      for (size_t i = 0; i < str.size(); ++i) {
4          result[i] ^= key;
5      }
6      return result;
7  }
8
```

2. Usage

During account creation, the password is encrypted using a fixed key and stored.


During login and password retrieval, the stored encrypted password is decrypted using the same key for verification.

3.3 File Handling for Account Management

File handling is used extensively for storing and retrieving user account details and passwords. The program uses file streams to manage data persistence.

1. Creating and Writing to Files

ofstream: Used to create and write to files. For example, saving account details to accounts.txt:

A screenshot of a code editor window with a dark background and light-colored text. The code is written in C++ and demonstrates how to use an ofstream to write to a file named 'accounts.txt'. The code includes comments and uses the ios::app flag to append data. It also includes an error handling section that prints a message if the file cannot be opened.

```
1 ofstream file("accounts.txt", ios::app);  
2 if (file.is_open()) {  
3     file << username << " " << email << " " << encryptedPassword << endl;  
4     file.close();  
5 } else {  
6     cout << "Error: Unable to open file\n";  
7 }  
8
```

2. Reading from Files

ifstream: Used to read from files. For example, verifying login credentials by reading from accounts.txt:

```

1 ifstream file("accounts.txt");
2 if (file.is_open()) {
3     while (file >> storedUsername >> storedEmail >> storedPassword) {
4         string decryptedPassword = xorEncryptDecrypt(storedPassword, key);
5         if (storedUsername == username && storedEmail == email && decryptedPassword == password) {
6             file.close();
7             return true;
8         }
9     }
10    file.close();
11 } else {
12     cout << "Error: Unable to open file\n";
13 }
14

```

3. Temporary Files

Used for tasks like changing the password. Data is first written to a temporary file and then the original file is replaced.

3.4 User Interaction and Input Handling

User interaction is handled through console input and output, ensuring a straightforward and user-friendly experience.

1. Console Input

cin: Used to capture user input for various operations like creating accounts, logging in, and entering passwords.

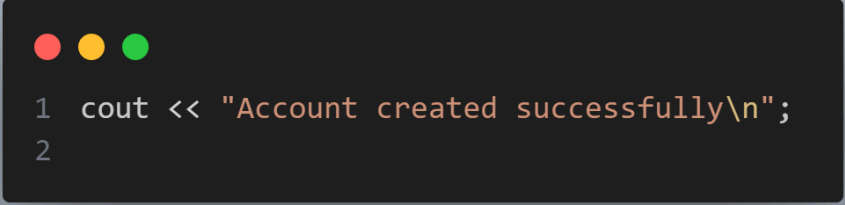
```

1 cout << "Enter username: ";
2 cin >> username;
3

```


2. Console Output

cout: Used to display prompts, messages, and results to the user.

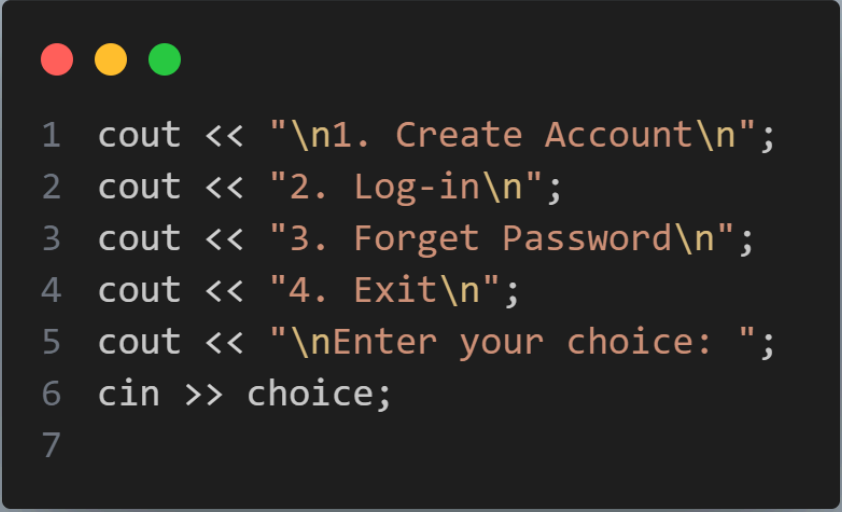


```
1 cout << "Account created successfully\n";  
2
```

3. Menu Navigation

The main menu and user dashboard provide options for users to navigate through the program.

Example:



```
1 cout << "\n1. Create Account\n";  
2 cout << "2. Log-in\n";  
3 cout << "3. Forget Password\n";  
4 cout << "4. Exit\n";  
5 cout << "\nEnter your choice: ";  
6 cin >> choice;  
7
```

In summary, the Capstone Password Manager is a robust and secure application built with clear code structure, efficient file handling, and simple encryption. It ensures user data security while providing easy-to-use functionalities for password management.

Chapter 4: Code Explanation

This chapter provides a detailed explanation of the code for the Capstone Password Manager, breaking down the main function, account creation logic, login process, password recovery, password change functionality, saving and retrieving website passwords, and password generation and strength assessment.

4.1 Main Function and Program Flow

The main function serves as the entry point for the program, handling user interaction and directing them to various functionalities based on their choices.

Code:

```
1  int main() {
2      int choice;
3      string loggedInUser;
4
5      cout << "Welcome to the Capstone Password Manager\n";
6
7      while (true) {
8          cout << "\n1. Create Account\n";
9          cout << "2. Log-in\n";
10         cout << "3. Forget Password\n";
11         cout << "4. Exit\n";
12         cout << "\nEnter your choice: ";
13         cin >> choice;
14
15         switch (choice) {
16             case 1:
17                 createAccount();
18                 break;
19             case 2:
20                 if (login(loggedInUser)) {
21                     cout << "Log-in successful\n";
22                     dashboard(loggedInUser);
23                 } else {
24                     cout << "Log-in failed\n";
25                 }
26                 break;
27             case 3:
28                 forgotPassword();
29                 break;
30             case 4:
31                 cout << "Exiting...\n";
32                 exit(0);
33             default:
34                 cout << "Invalid choice\n";
35         }
36     }
37
38     return 0;
39 }
40
```

Explanation:

- The program starts by displaying a welcome message.
- It enters an infinite loop, displaying the main menu options for creating an account, logging in, recovering a password, and exiting the program.
- Based on the user's choice, it calls the appropriate function (createAccount, login, forgotPassword) or exits the program.
- If the login is successful, it directs the user to the dashboard function.

4.2 Account Creation Logic

The createAccount function is responsible for handling the entire process of creating a new user account. This includes gathering user input for the username, email, and password, ensuring that the password is entered correctly twice for confirmation. Once the passwords match, it encrypts the password using XOR encryption with a predefined key. The function then securely stores the account details, including the encrypted password, into a file. Additionally, it creates a separate file for the user, setting up the infrastructure needed for storing website passwords associated with the account. This comprehensive approach ensures that user data is both securely handled and properly organized from the moment of account creation.

Code:

```

1 void createAccount() {
2     string username, password, cPass, email;
3     char key = 'K'; // Using a simple fixed key for account password encryption
4
5     cout << "\nEnter username: ";
6     cin >> username;
7
8     cout << "Enter E-mail: ";
9     cin >> email;
10
11    cout << "Enter password: ";
12    cin >> password;
13
14    cout << "Enter password again: ";
15    cin >> cPass;
16
17    if (cPass != password) {
18        cout << "Password doesn't match. Try again!" << endl;
19        return; // Exit the function if passwords don't match
20    }
21
22    string encryptedPassword = xorEncryptDecrypt(password, key);
23
24    ofstream file("accounts.txt", ios::app);
25    if (file.is_open()) {
26        file << username << " " << email << " " << encryptedPassword << endl;
27        file.close();
28        ofstream userFile(username + ".txt");
29        userFile.close();
30        cout << "Account created successfully\n";
31    } else {
32        cout << "Error: Unable to open file\n";
33    }
34 }
35

```

Explanation:

- Prompts the user for their username, email, and password (entered twice for confirmation).
- Checks if the passwords match; if not, it exits the function.
- Encrypts the password using the xorEncryptDecrypt function.
- Opens accounts.txt in append mode to save the account details (username, email, encrypted password).
- Creates an individual file for the user to store their website passwords.
- Displays a success message or an error if the file cannot be opened.

4.3 Login Process

The login function verifies user credentials by decrypting the stored password and comparing it with the input.

Code:

```
1 bool login(string &loggedInUser) {
2     string username, password, email, storedUsername, storedPassword, storedEmail;
3     char key = 'K'; // Using the same fixed key for account password encryption
4
5     cout << "\nEnter username: ";
6     cin >> username;
7
8     cout << "Enter e-mail: ";
9     cin >> email;
10
11    cout << "Enter password: ";
12    cin >> password;
13
14    ifstream file("accounts.txt");
15    if (file.is_open()) {
16        while (file >> storedUsername >> storedEmail >> storedPassword) {
17            string decryptedPassword = xorEncryptDecrypt(storedPassword, key);
18            if (storedUsername == username && storedEmail == email && decryptedPassword == password) {
19                file.close();
20                loggedInUser = username;
21                return true;
22            }
23        }
24        file.close();
25    } else {
26        cout << "Error: Unable to open file\n";
27    }
28
29    return false;
30 }
31
```

Explanation:

- Prompts the user for their username, email, and password.
- Opens accounts.txt and reads stored account details.
- Decrypts each stored password and compares it with the input.
- If a match is found, sets loggedInUser to the username and returns true.
- If no match is found, returns false.

4.4 Password Recovery

The forgotPassword function retrieves and displays the decrypted password if the user forgets it.

Code:

```
1 void forgotPassword() {
2     string username, email, storedUsername, storedEmail, storedPassword;
3     char key = 'K'; // Using the same fixed key for account password encryption
4
5     cout << "\nEnter username: ";
6     cin >> username;
7
8     cout << "Enter e-mail: ";
9     cin >> email;
10
11     ifstream file("accounts.txt");
12     if (file.is_open()) {
13         while (file >> storedUsername >> storedEmail >> storedPassword) {
14             if (storedUsername == username && storedEmail == email) {
15                 string decryptedPassword = xorEncryptDecrypt(storedPassword, key);
16                 cout << "Your password is: " << decryptedPassword << endl;
17                 file.close();
18                 return;
19             }
20         }
21         cout << "No account found with the provided username and email" << endl;
22         file.close();
23     } else {
24         cout << "Error: Unable to open file\n";
25     }
26 }
27
```

Explanation:

- Prompts the user for their username and email.
- Opens accounts.txt and reads stored account details.
- If a match is found, decrypts and displays the stored password.
- If no match is found, displays an error message.

4.5 Password Change Functionality

The changePassword function allows users to change their account password by updating the stored encrypted password.

Code:

```

1 void changePassword(string username, string newPassword) {
2     ifstream inFile("accounts.txt");
3     ofstream outFile("temp.txt");
4     char key = 'K'; // Using the same fixed key for account password encryption
5
6     string storedUsername, storedEmail, storedPassword;
7     bool userFound = false;
8
9     if (inFile.is_open() && outFile.is_open()) {
10         while (inFile >> storedUsername >> storedEmail >> storedPassword) {
11             if (storedUsername == username) {
12                 string encryptedPassword = xorEncryptDecrypt(newPassword, key);
13                 outFile << storedUsername << " " << storedEmail << " " << encryptedPassword << endl;
14                 userFound = true;
15             } else {
16                 outFile << storedUsername << " " << storedEmail << " " << storedPassword << endl;
17             }
18         }
19         inFile.close();
20         outFile.close();
21
22         remove("accounts.txt");
23         rename("temp.txt", "accounts.txt");
24
25         if (userFound) {
26             cout << "Password changed successfully\n";
27         } else {
28             cout << "User not found\n";
29         }
30     } else {
31         cout << "Error: Unable to open file\n";
32     }
33 }
34

```

Explanation:

- Reads accounts.txt and writes to a temporary file temp.txt.
- If the username matches, encrypts the new password and writes the updated account details.
- If the username does not match, writes the original account details.

- Replaces accounts.txt with temp.txt.
- Displays a success message if the user is found, or an error if the file cannot be opened.

4.6 Saving and Retrieving Website Passwords

The savePassword and showPassword functions handle storing and retrieving website passwords.

Save Password Function:

```

1  void savePassword(const string &username) {
2      string websiteName, webPass, cPass, securityKey;
3
4      cout << "\nEnter Website name: ";
5      cin >> websiteName;
6
7      cout << "Enter Website Password: ";
8      cin >> webPass;
9
10     cout << "Enter password again: ";
11     cin >> cPass;
12
13     if (cPass != webPass) {
14         cout << "Password doesn't match. Try again!" << endl;
15         return; // Exit the function if passwords don't match
16     }
17
18     cout << "Enter a security key (Numerical): ";
19     cin >> securityKey;
20
21     ofstream file(username + ".txt", ios::app);
22     if (file.is_open()) {
23         string encryptedPass = xorEncryptDecrypt(webPass, securityKey[0]);
24         file << websiteName << " " << encryptedPass << endl;
25         file.close();
26         cout << "Password saved successfully\n";
27     } else {
28         cout << "Error: Unable to open file\n";
29     }
30 }
31

```


Show Password Function:

```

1 void showPassword(const string &username) {
2     string websiteName, storedWebsiteName, storedPassword, securityKey;
3
4     cout << "Enter your security key: ";
5     cin >> securityKey;
6
7     ifstream file(username + ".txt");
8     if (file.is_open()) {
9         while (file >> storedWebsiteName >> storedPassword) {
10             string decryptedPassword = xorEncryptDecrypt(storedPassword, securityKey[0]);
11             cout << "Website: " << storedWebsiteName << "\tPassword: " << decryptedPassword << endl;
12         }
13         file.close();
14     } else {
15         cout << "Error: Unable to open file\n";
16     }
17 }
18

```

Explanation:

- **savePassword:** Prompts the user for the website name, password (entered twice for confirmation), and a security key. Encrypts the password using the security key and saves it to the user's file.
- **showPassword:** Prompts the user for their security key, decrypts the stored passwords using the key, and displays them.

4.7 Password Generation and Strength Assessment

The generatePass and passStrength functions provide features for generating strong passwords and assessing password strength.

Generate Password Function:

```

1 void generatePass() {
2     int length;
3     string password;
4     const string charset = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()";
5
6     cout << "Enter the length of password: ";
7     cin >> length;
8
9     srand(time(nullptr));
10
11     for (int i = 0; i < length; ++i) {
12         password += charset[rand() % charset.size()];
13     }
14
15     cout << "Generated Password: " << password << endl;
16 }
17

```

Password Strength Function:

```

1 void passStrength() {
2     string password;
3     cout << "Enter password: ";
4     cin >> password;
5
6     bool hasLower = false, hasUpper = false, hasDigit = false, hasSpecial = false;
7
8     for (char c : password) {
9         if (islower(c)) hasLower = true;
10        if (isupper(c)) hasUpper = true;
11        if (isdigit(c)) hasDigit = true;
12        if (ispunct(c)) hasSpecial = true;
13    }
14
15    cout << "Password strength: ";
16    if (hasLower && hasUpper && hasDigit && hasSpecial) {
17        cout << "Strong\n";
18    } else if ((hasLower && hasUpper) || (hasLower && hasDigit) || (hasUpper && hasDigit)) {
19        cout << "Medium\n";
20    } else {
21        cout << "Weak\n";
22    }
23 }
24

```

Explanation:

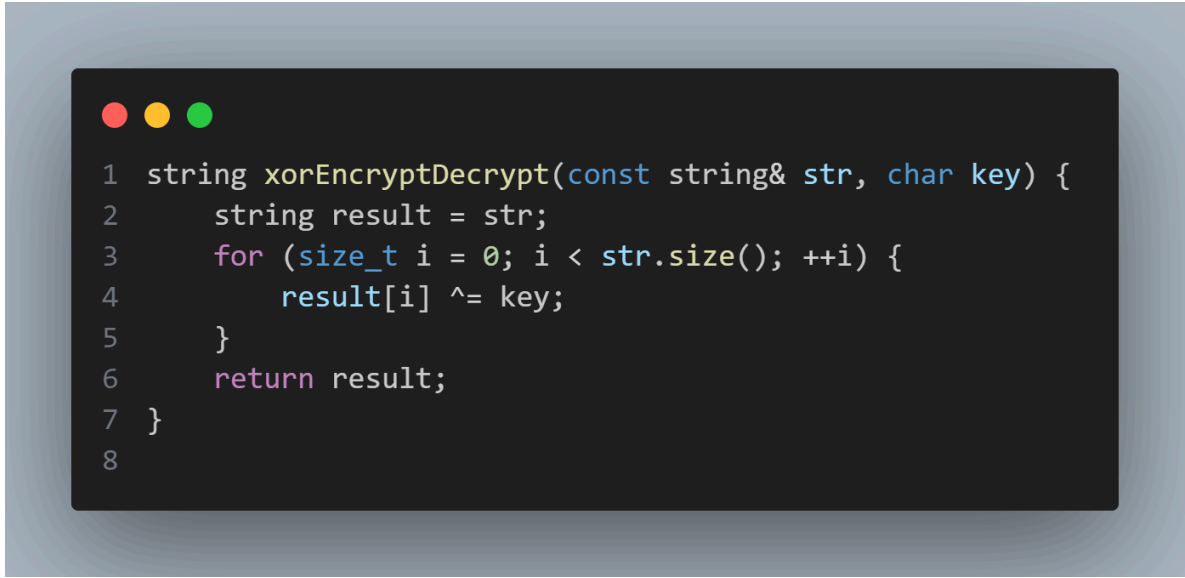
- **generatePass:** Prompts the user for the desired password length, generates a random password from a set of characters, and displays it.
- **passStrength:** Prompts the user for a password, checks for the presence of lowercase, uppercase, digit, and special characters, and evaluates the password strength based on these criteria.

Chapter 5: Security Considerations

Security is paramount when managing passwords and sensitive user information. This chapter discusses the encryption techniques used, data protection measures implemented, and best practices for maintaining password security.

5.1 Encryption Techniques

The Capstone Password Manager employs a simple XOR encryption technique to protect user passwords.

XOR Encryption:

```
1 string xorEncryptDecrypt(const string& str, char key) {  
2     string result = str;  
3     for (size_t i = 0; i < str.size(); ++i) {  
4         result[i] ^= key;  
5     }  
6     return result;  
7 }  
8
```

Explanation:

- **XOR Encryption:** This method uses the XOR bitwise operator to encrypt and decrypt data. Each character in the string is XORed with a fixed key.
- **Fixed Key:** In this project, a fixed key (char key = 'K';) is used for both encryption and decryption.

Advantages and Disadvantages:

Advantages: Simple and fast, suitable for basic encryption needs.

Disadvantages: Not very secure against more sophisticated attacks, as it relies on a single key and can be easily broken if the key is known.

5.2 Data Protection Measures

Several measures are taken to ensure the security and integrity of user data:

File Handling:

User Accounts: Account information (username, email, and encrypted password) is stored in accounts.txt.

User-Specific Files: Each user has a dedicated file (username.txt) for storing encrypted website passwords.

Code Example:

```
1 ofstream file("accounts.txt", ios::app);
2 if (file.is_open()) {
3     file << username << " " << email << " " << encryptedPassword << endl;
4     file.close();
5     ofstream userFile(username + ".txt");
6     userFile.close();
7     cout << "Account created successfully\n";
8 } else {
9     cout << "Error: Unable to open file\n";
10 }
11
```

Access Controls:

Read and Write Operations: The code ensures that files are opened in appropriate modes (`ios::app` for appending, `ifstream` for reading) to prevent unauthorized modification.

Error Handling: The program checks if files can be opened and provides error messages if they cannot, preventing silent failures.

5.3 Best Practices for Password Security

To enhance the security of passwords, users should follow these best practices:

Use Strong Passwords:

Characteristics: A strong password should include a mix of uppercase and lowercase letters, digits, and special characters.

Length: The password should be at least 12 characters long to increase its complexity.

Avoid Reusing Passwords:

Unique Passwords: Use different passwords for different accounts to limit the impact of a single compromised password.

Regularly Update Passwords:

Change Periodically: Regularly updating passwords reduces the risk of long-term exposure if a password is compromised.

Enable Two-Factor Authentication (2FA):

Additional Layer: Use 2FA whenever possible to add an extra layer of security beyond just the password.

Secure Storage of Passwords:

Password Managers: Use a reputable password manager to securely store and manage passwords, ensuring they are encrypted and protected by a master password.

Example of Generating a Strong Password:

```

1 void generatePass() {
2     int length;
3     string password;
4     const string charset = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()";
5
6     cout << "Enter the length of password: ";
7     cin >> length;
8
9     srand(time(nullptr));
10
11     for (int i = 0; i < length; ++i) {
12         password += charset[rand() % charset.size()];
13     }
14
15     cout << "Generated Password: " << password << endl;
16 }
17

```

Explanation:

- **Charset:** The character set includes a variety of characters to enhance password strength.
- **Randomization:** The use of `srand` with the current time ensures the randomness of generated passwords.

By following these best practices and leveraging the Capstone Password Manager, users can significantly improve their password security and protect their online accounts.

Chapter 6: Conclusion

In this chapter, we will summarize the features of the Capstone Password Manager, discuss its impact on user security, and provide some final thoughts on the project.

6.1 Summary of Features

The Capstone Password Manager offers a comprehensive suite of features designed to help users manage their passwords securely and efficiently. Key features include:

- **Account Creation and Management:** Users can create accounts with a username, email, and password, which are stored securely using XOR encryption.
- **Secure Login:** The login process verifies user credentials against stored data, ensuring only authorized access.
- **Forgot Password Functionality:** Users can recover their passwords by verifying their username and email.
- **Password Change:** Users can change their account passwords, which updates the encrypted password stored in the system.
- **Website Password Management:** Users can save and retrieve passwords for various websites, with each password encrypted for security.
- **Password Generation:** The tool can generate strong, random passwords to enhance security.
- **Password Strength Assessment:** Users can check the strength of their passwords based on length and complexity.

6.2 Impact on User Security

The Capstone Password Manager significantly enhances user security by implementing several key measures:

- **Encryption:** Using XOR encryption ensures that stored passwords are not kept in plain text, reducing the risk of exposure.
- **Data Segregation:** User-specific files ensure that each user's data is isolated, which limits the impact of potential breaches.
- **User Guidance:** The password strength assessment and generation features guide users towards creating strong passwords, which are critical for online security.
- **Error Handling and User Feedback:** The system provides clear feedback on operations, such as successful account creation or login failure, helping users understand and rectify issues promptly.

6.3 Final Thoughts

The Capstone Password Manager represents a significant step towards better password management and security for users. While the XOR encryption technique used in this project is simple and may not be the most robust against advanced attacks, it serves as an effective learning tool for understanding the basics of encryption and secure data handling.

Future improvements could include implementing more advanced encryption methods, adding support for multi-factor authentication, and enhancing the user interface for better user experience. Despite its limitations, the Capstone Password Manager is a valuable tool that helps users adopt better password management practices and underscores the importance of cybersecurity in everyday digital interactions.

In conclusion, this project highlights the fundamental principles of secure password management and provides a solid foundation for further development and enhancement. By continuing to build on this foundation, we can create even more secure and user-friendly password management solutions in the future.