Tampere University

COMP.SEC.300-2024-2025-1 Secure Programming

Group name: Pummit

Password manager with 2FA

Project report

Rami Nurmoranta

Antti Santala

Veeti Salminen

The AI tools used in the exercise work and the purpose of their use has been described below:

Name of the tool (and version): Cursor IDE (claude-3.7-sonnet)

Purpose of use and the part in which it was used: AI was used in the code creation as well as debugging the code and help with the design.

Name of the tool (and version): ChatGPT

Purpose of use and the part in which it was used: AI was used to draft the raw version of the report

Table of Contents

1. General description

Our password manager is designed to solve one of the most common and important security challenges today: managing online credentials safely. The application combines encryption, user friendly design, and smart password tools, enabling users to protect their digital lives without the frustration of remembering countless passwords. Key features include:

- Password management: Users can securely store their passwords for various websites using our product. The stored credentials are encrypted and easily accessible, allowing users to manage their passwords without compromising security.
- Password tools: A built-in password generator helps users create strong and customizable passwords for different services. The application also analyses stored passwords, identifies weak or reused ones, and offers recommendations to strengthen overall password quality.
- Secure authentication: The system uses a single master password to derive encryption keys. It supports two-factor authentication (2FA) with time-based one-time passwords adding an extra layer of security. Our 2FA system was fully developed by us. It works by linking a mobile device to the user's account through the database. When a login attempt is made, a challenge is sent to the registered device, which is the only one capable of verifying the login using a unique 6-digit code. Users can also manage which devices are authorized to access their account and revoke access when needed.
- End-to-end encryption: All sensitive data is encrypted locally using AES-256-GCM before it is stored. The encryption keys never leave the user’s device, and data is never saved in plaintext. Decryption happens only temporarily in memory, providing full privacy and security.

Summary of technologies used:

- Expo Go: A framework that accelerated the development of the mobile app, allowing us to test the application very fast
- TypeScript (Mobile App): Used in the mobile app
- JavaScript (Web App): The primary language for the web app
- Node.js: The backend engine that powers both mobile and web apps.
- AES-256-GCM: Used to encrypt sensitive data both at rest and in transit.
- PBKDF2: A secure password hashing method that helps protect user credentials from brute-force attacks.

- TOTP (Time-Based One-Time Password): Part of our two-factor authentication (2FA) system, providing an additional layer of security through time-sensitive authentication codes.
- MongoDB: A NoSQL database that securely stores encrypted credentials, session data, and user information.
- TLS/SSL: Encryption protocols to protect all data transmission between the client and server, preventing unauthorized access.

2. Program usage

In this chapter, there are explained all the scenarios that user of the program might face. First is explained the initial setup, then daily usage of the application and finally how to maintain the application secure.

2.1 Initial setup

To begin using the password manager follow these steps:

1. Create an account by entering your personal information and setting a strong master password.
2. Configure 2FA to improve login security and adjust your security preferences.
3. Import existing credentials and organize them into categories.
4. Review your security settings to ensure everything is properly configured.
5. After you're done, log out to secure your session.

2.2 Daily usage

For daily use follow these steps:

1. Log in with your master password.
2. Complete the 2FA process.
3. Navigate to the credential you need and manually copy the username or password for use.
4. To add new credentials, open the add credential form. Input the account information, generate or enter a strong password, add it to the appropriate category, and save the entry.
5. After you're done, log out to secure your session.

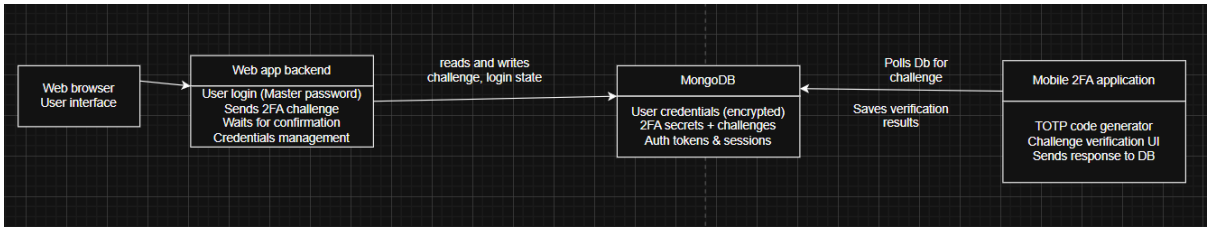
2.3 Security maintenance

To maintain security over time, regularly review your security settings:

1. Log in with your master password.
2. Verify that 2FA is functioning correctly.
3. Check the activity log for any signs of suspicious behaviour.
4. Update your master password if necessary.
5. After you're done, log out to secure your session.

3. Structure of the program

The overall system consists of two main components: a mobile two-factor authentication (2FA) application and a web-based secure credentials manager. Each application has been developed independently but works together to make 2FA possible. Below, we describe the structure and functionality of both applications with system architecture diagram.



Picture 1: System architecture diagram

3.1 Two-factor authentication mobile application

The mobile application uses modular component-based architecture. The overall structure of the application is in the picture below. Folder *android/* contains Android-specific native code and configurations. It includes Android project settings, build configurations, and native modules. Folder *app/* has the main application code. Basically, there is three things: Firstly, there is *screens/*, which contains the main screen of the app. Secondly, there is *utils/*, which contains *api.ts* for API communication and *crypto.ts* for cryptographic operations. Thirdly, there is *constants/,* which includes config.ts for configuration constants and settings. Then *assets/* has static resources like images and fonts. There are also configuration files, for example app.json for Expo configuration and index.js is the entry point of the application.

```
mobilephone_2F_login_app/
├── android/            # Android-specific native code
├── app/               # Main application code
│   ├── screens/       # Screen components
│   ├── utils/         # Utility functions
│   ├── constants/     # Constant values
│   └── App.tsx        # Root application component
├── assets/            # Static assets (images, fonts, etc.)
├── index.js           # Entry point
└── app.json           # Expo configuration
```

Picture 2: Mobile application structure

When creating the structure of the application we always had in mind the security. The 2FA system adds an extra layer of security on top of the master password. Its backend uses MongoDB to store verification codes and manage sessions securely. The architecture has protection against unauthorized access with features: secure authentication code generation, and time-based authentication expiry.

To generate authentication codes, the program uses cryptographically secure random number generators and algorithms, for example expo-crypto. This guarantees that the codes are unpredictable and unique, which help preventing pattern recognition and replay attacks. The codes are stored in an encrypted format in MongoDB, with all database connections secured via TLS/SSL and access tightly controlled using permissions.

Codes are valid only for a limited time, after which they expire. Every code is also single use once it has been used for authentication, it cannot be reused. When a user attempts to log in using a code, the system compares the input against stored values in constant time to avoid timing attacks.

3.2 Secure credentials manager

The credentials manager is a web application that focuses on the secure storage and handling of user login data. Below is the structure of the program. The source code of the program is in *src/* folder: Firstly, *app/* contains the application's pages and routes. It uses uses Next.js app router for routing and implements server-side rendering (SSR). Secondly, *components/* contains reusable UI components and follows component-based architecture. Thirdly, *lib/* contains utility functions, database connections, API handlers and business logic. Fourthly, *types/* contains TypeScript definitions.

```
secure-credentials-app/
├── src/                    # Source code
│   ├── app/                # Next.js app router pages
│   ├── components/         # Reusable React components
│   ├── lib/                # Utility functions and libraries
│   └── types/              # TypeScript type definitions
├── public/                 # Static assets
├── next.config.ts          # Next.js configuration
├── postcss.config.mjs      # PostCSS configuration
└── tsconfig.json           # TypeScript configuration
```

Picture 3: Secure credentials manager

Also, when designing this application we had security in mind. It uses end-to-end encryption to protect user credentials, meaning that all sensitive data is encrypted locally on the user's device before being stored, and decrypted only in memory when needed.

All stored passwords and credentials are encrypted using AES-256-GCM, one of the most secure encryption methods available. The encryption keys are derived using PBKDF2 with salting, making them resilient against brute-force attacks. Each encryption operation uses a unique initialization vector to ensure that patterns cannot be deduced, and authentication tags are used to verify data integrity.

The application includes features for organizing credentials using categories and favourites. A search function allows users to quickly find stored data. All search queries are handled with secure indexing to prevent information leakage.

To manage user access, a master password system is used. This is the only password users need to remember, and it is never stored directly. Instead, it is securely used to derive encryption keys. The system handles session management securely, with timeouts and revocation features to prevent unauthorized access. An audit log tracks all changes and access events, which can be reviewed to ensure compliance and detect suspicious behaviour.

The user interface provides a clear and user-friendly dashboard. From there, users can navigate their stored credentials, view security status, and take common actions like adding new entries or generating strong passwords.

4. Secure programming solutions

Secure programming practices were integrated into both applications to ensure resilience against common security threats. Throughout development, we referred to well-known checklists such as the OWASP Top 10 and SANS CWE Top 25 to guide our implementation. The secure solutions are in table below.

OWASP Risk	Web Application	Mobile Application
A1:2021 – Broken Access Control	Role-based access control	Device authentication
A2:2021 – Cryptographic Failures	AES-256-GCM, PBKDF2	Secure key storage
A3:2021 – Injection	MongoDB parameterized queries	Input validation
A4:2021 – Insecure Design	Password strength policies	Secure device registration
A5:2021 – Security Misconfiguration	HTTP headers	Platform security
A6:2021 – Vulnerable and Outdated Components	Up-to-date dependencies	Latest SDK
A7:2021 – Identification and Authentication Failures	2FA, secure password storage	-
A8:2021 – Software and Data Integrity Failures	Data integrity checks	Challenge-response
A9:2021 – Security Logging and Monitoring Failures	Centralized logging, alerting systems	Local logging, remote telemetry
A10:2021 – Server-Side Request Forgery (SSRF)	Input sanitization, network restrictions	Secure API gateways, controlled backend access

The previous table compares how the OWASP Top 10 security risks of 2021 manifest in both web and mobile application environments. While the core risks are common across platforms, their implementation and mitigation strategies differ. For instance, mobile applications often rely on device-specific features such as biometric authentication or secure key storage, whereas web applications typically focus on browser-based configurations and server-side protections. This comparison highlights the platform-specific considerations developers and security professionals should consider when designing secure systems.

In addition, input validation and sanitation were implemented across the application to prevent injection attacks. All user inputs are validated on both the client and server sides, and sensitive operations like password storage are protected with proper cryptographic practices. Access control is enforced using role-based access (RBAC), ensuring that users can only perform actions appropriate for their authentication level. Every feature that needs additional security uses 2FA to verify the action.

Sensitive data exposure is mitigated by using encryption at all stages: in transit, at rest, and in memory. Passwords, codes, and session data are never stored in plaintext. The database

is securely connected via TLS and access to it is strictly restricted. Authentication codes are compared in constant time to defend against timing attacks, and error messages are designed to reveal as little information as possible to potential attackers.

Where possible, our team has also documented critical security implementations directly in the source code with comments. This improves code readability and helps future developers maintain security standards during further development.

5. Security testing

We decided to make manual test cases to ensure that all use cases work and promised functionality is implemented properly. Each test case is evaluated either success or failure, and the results are documented in the table below.

Description of the test case	Result
Signing up with existing username or password	Success
Signing up with password without numbers or symbols	Success
Signing up with unique username, password and valid passwords	Success
Signing up with different passwords in the password and confirmation field	Success
Logging in with correct username and password	Success
Pairing device to user account during sign up process with security code	Success
As a logged in user, saving credentials to the app	Success
Resetting password with username, email and device verification	Success
All login attempts saved to database	Success
Cannot login without confirmation from the device	Success
Password information not visible on the browser storage	Success
Password information hashed in the database	Success

All the implemented functionality worked as expected and we found no security issues.

6. Future improvements

Here are a couple of improvement ideas if the development of the project is continued in the future:

- Backup codes for lost 2FA devices.
- Autofill functionality, when login in browser.