Password manager With 2FA

Pummit : Antti Santala, Rami Nurmoranta and Veeti Salminen

Inspiration

We built this program out of thin air

Our motivation being only to create a secure password manager

And what better way of doing that than using homemade 2FA

This was quite a bit of work but gladly there were 3 group members to share the workload with

Aim of the program

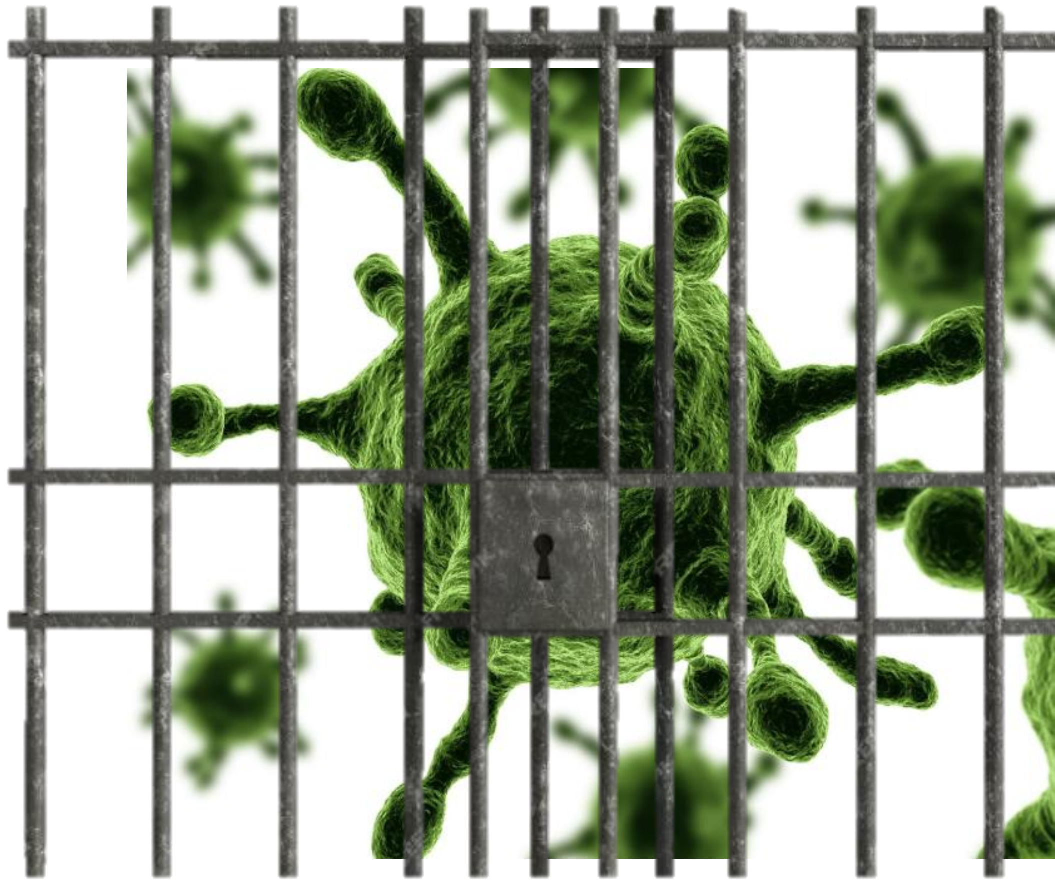
Program Goal:
- To simplify and secure online credential management by combining strong encryption, intuitive design, and smart password tools.

How It Works:

- Initial Setup
  - Create account with a master password
  - Configure custom-built 2FA
  - Import and organize credentials
- Daily Use
  - Log in with master password + 2FA
  - Access or add credentials
  - Copy and find passwords securely
  - Log out to end session
- Security Maintenance
  - Regularly update settings and passwords
  - Monitor device access and activity logs
  - Revoke compromised devices

Security features

- Secure Authentication
  - Master password never stored; used for key derivation
  - Custom 2FA with TOTP and device-based verification
  - Manage trusted devices
- End-to-End Encryption
  - AES-256-GCM applied locally; keys stay on device
  - Decryption only in memory
- Password Tools & Management
  - Password strength analysis
  - Encrypted storage
- Secure Programming Practices
  - Based on OWASP Top 10 & SANS CWE Top 25
  - For example, session control and secure error handling



OWASP Risk Mitigation Examples

OWASP Risk	Web Application	Mobile Application
A1: Broken Access Control	Role-based access	Device auth
A2: Cryptographic Failures	AES-256-GCM, PBKDF2	Secure key storage
A3: Injection	Input validation	Parameterized queries
A4: Insecure Design	Password policy	Secure device registration
A5: Security Misconfig	HTTP headers	Platform security
A6: Outdated Components	Up-to-date libs	Latest SDKs
A7: Auth Failures	2FA, password storage	—
A8: Integrity Failures	Data checks	Challenge-response
A9: Logging Failures	Central logs & alerts	Local + telemetry
A10: SSRF	Input sanitization	Secure API access

Technical solutions

- **React Native with Expo (Mobile App)**
- **React with Next.js + Node.js (Web App)**
- **PBKDF2**: Password hashing and encryption key creation.
- **AES-256-GCM**: Encrypting the user stored credentials.
  - PBKDF2 and AES-256-GCM implementations from Node.js crypto module
- **TOTP (Time-Based One-Time Password)**: Part of our two-factor authentication (2FA) system, time-sensitive authentication codes.
- **MongoDB**: A NoSQL database that stores encrypted credentials, logs, and user information.

Applications testing

Description	Test Success/Fail
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

Manual testing as test method

Use of AI

- Tool: Cursor IDE (Claude 3.7 Sonnet)
  Purpose:
  - Assisted in code creation
  - Supported debugging process
  - Provided input for design decisions

- Tool: ChatGPT
  Purpose:
  - Drafted the initial version of the report