COMPX518-21A

Assignment 3

# Application Design

## Creating a Vault

This flow describes the process of creating a new vault.

## Saving a Vault

This flow describes the process of saving a vault. AES GCM is used to ensure authenticity and integrity of the cipher text. Alternatively, CTR or CBC could have been used alongside a method such as Encrypt-then-MAC.

## Opening a Vault

This flow describes the process of opening a vault.

# Secure Random Number Generator

To guide my decision in choosing a random number generator for this application, I read the OWASP Cryptographic Storage Cheat Sheet. This document states that a Cryptographically secure pseudo-random number generator should be used when dealing with cryptography, and that in .NET / C#, I should use the **RNGCryptoServiceProvider** class instead of the standard **Random** class.

**RNGCryptoServiceProvider** is better suited for password generation because it provides a more secure random function that is not as repeatable as **Random** (at the expense of speed).

The graph below shows a distrubution of generated characters (50 million 80 character passwords). The least common character was **k** with *42538130* occurances, while the most common character was **w** with *42572116* occurances (a difference of 33,986).

Chart, bar chart

Description automatically generated

# Master Password Authentication

…

TODO: Diagram.

When the user enters their master password, it’s hashed using one of the following methods:

* PBKDF2-HMAC-SHA1: 720,000 iterations
* PBKDF2-HMAC-SHA256: 310,000 iterations
* PBKDF2-HMAC-SHA512: 120,000 iterations

# Password Storage

## Vault File Segments

|  |  |  |  |
| --- | --- | --- | --- |
| **Cipher Text** | **Salt** | **Nounce** | **Tag** |
| Variable Bytes | 32 Bytes | 13 bytes | 16 bytes |

The password vault is stored as a \*.vault file. This binary file contains the encrypted passwords, alongside the

These are some quick notes so I don’t forget

The password vault is stored as a \*.vault file. This file contains a salt (first X bytes), and then an encrypted JSON payload of all the passwords. When reading from the vault, the salt will be read and the encrypted payload. The user will enter their master password which will be combined with the salt to produce a PBKDF2 hash. This hash is the AES encryption key used to unlock the vault.

KDF: Key derivation function

Password stored AES 256bit

32byte salt

// The authenticity and integrity of the data is ensured using a HMAC-SHA-256 hash of the ciphertext (Encrypt-then-MAC scheme).

<https://keepass.info/help/base/security.html>

<https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rfc2898derivebytes?view=net-5.0>

# Notes

When running the program on macOS, you may get an error like “No usable version of libssl was found”. See <https://docs.microsoft.com/en-us/dotnet/standard/security/cross-platform-cryptography#aes-ccm-and-aes-gcm-on-macos>

Also see: <https://blog.bokan.io/posts/aesccm-and-aesgcm-in-dotnet-core-on-macos-catalina/>

Personally, I already had OpenSSL installed, I added the following environment variable:

* ***export DYLD\_LIBRARY\_PATH=/usr/local/opt/openssl@1.1/lib***

# Resources

* <https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html>
* <https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rngcryptoserviceprovider>
* <https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.aesgcm?view=net-5.0>
* <https://owasp.org/www-community/password-special-characters>
* <https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2>