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.

Salting disallows precomputed tables

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

The password vault is stored as a \*.vault file. This binary file contains the encrypted passwords, alongside the password hash salt, nonce, and authentication tag. The C# class “DataPacker” is used to pack and unpack the binary file into usable byte arrays. See the table below for a visual representation of the file.

|  |  |  |  |
| --- | --- | --- | --- |
| **Cipher Text** | **Salt** | **Nonce** | **Tag** |
| Variable Bytes | 32 Bytes | 12 bytes | 16 bytes |

The cipher text is generated using AES GCM (Galois/Counter Mode) as per the OWASP cryptographic storage recommendations. The GCM cipher mode provides both data authenticity (integrity) and confidentially. GCM does not require the use of an external “Encrypt-then-MAC” scheme which is commonly used during CBC and CTR cipher modes (as they don’t provide any guarantees on the authenticity of the encrypted data).

The key used for AES GCM is a 256bit value obtained from a Key Derivation Function (KDF). This KDF is mentioned in more detail in the previous section, “Master Password Authentication”. In summary, the key is derived from the user’s master password going through a PBKDF2-HMAC-SHA512 function with 200,000 iterations.

In order to access the vault, the user must supply the same master password which they used to first create the vault. Without this master password, the contents of the vault cannot be decrypted (the salt, nonce and tag alone cannot decrypt the vault). The users master password, generated hash, or key are never stored to disk.

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.

<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>