Password Manager Project Specification

Abstract

In this project I will create 2 different parts for this product, a server and a front-end. The front-end will be packaged in two forms, a standalone Electron wrapper for many distributions, including Linux, Mac and Windows, and a progressive web application which is mobile device friendly and allows background syncing. There will be no differences in the code between the progressive app and the Electron front end. The server's job will be to facilitate the users and host the progressive web application. The application will primarily allow users to store and generate high entropy passwords securely with a big number of secure backup solutions.

Cryptography Methods

There will be several encryption methods that will be used in the program. Most importantly, the password manager will feature a private advanced recovery and authentication options not commonly found in other password managers.

Cryptographic container

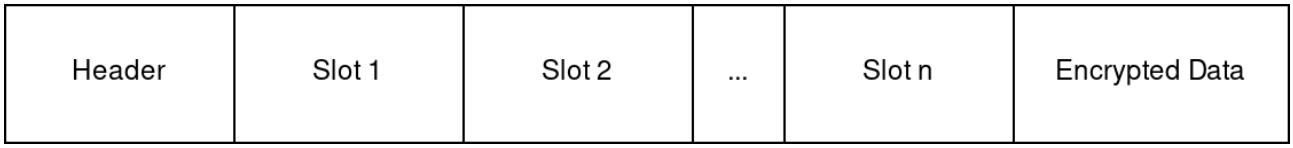

Figure 1 - Database container example.

The cryptographic container will be a Linux Unified Key Setup (aka LUKS) like container for the database and the main encryption key (cryptsetup, 2020) as described by Figure 1. Each slot requires a password meaning multiple passwords can unlock the encrypted data. This could in theory mean that two separate people with two separate passwords could unlock this container given they both had valid passwords. This, of course, becomes not optimal if there are many people that could want access to the container. Because of this, it's likely that not many users will use this function.

Architecture details

The header will contain all the necessary information in regards to the slots that house the master key and the storage on the device. The slots contain a copy of the secret master key, the PBKDF2 salt, the PBKDF2 iteration count and the PBKDF2 hashing algorithm name. Unlike LUKS, the user can have an unlimited (within reason) number of slots, with a minimum of 0 slots. This is by design. Each slot will require a unique password to open. The user will be able to remove or add slots at will. The user will be able to change the number of iteration count only to a higher number, preventing password decay.

Shared secret recovery option

(Adi Shamir, 1979) describes a solution where secret data S is split into n parts and a threshold where k number of pieces are required to recover the passwords (where $0<k\leq n$, and a_i where $a_1,...,a_{k-1}$ are random $k-1$ integers and where $a_0=S$, and $f(x)=a_0+a_1x+a_2x^2+a_3x^3+...+a_{k-1}x^{k-1}$, and D_i is any $(i,f(i))$ where $i=1,...,n$). The only criticism of this paper would be when an attacker gains D_i, it becomes increasingly easier for the attacker to predict what the polynomial is, thus figuring out the solution. This is because of the smooth and predictable nature of polynomials, this is shown in Figure 2. A simple modification can be made to Shamir's scheme to make it resistant to this type of attack. A prime p and threshold k should be chosen such, $p>a_i$, $0<k\leq n<p$ and $f(x)$ should be updated to $f(x)=a_0+a_1x+a_2x^2+a_3x^3+...+a_{k-1}x^{k-1} mod\ p$. The result of this modification can be seen in Figure 3.

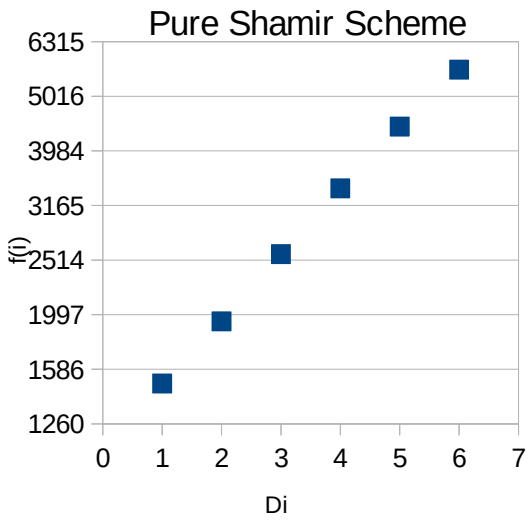

Figure 2: Pure Shamir Scheme shows straight line (logarithmic y-scale)

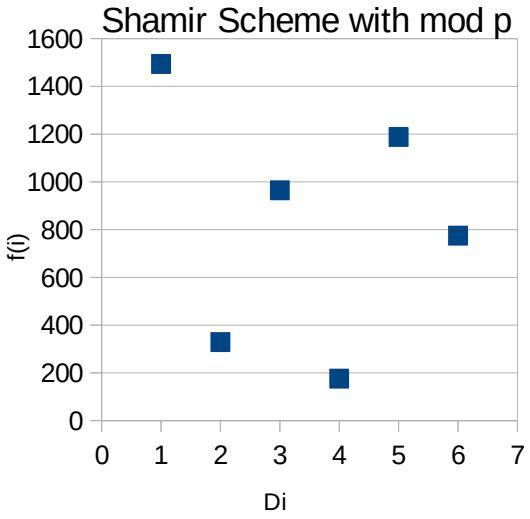

Figure 3: Shamir Scheme with mod p

Server

The server will provide may services for the users and the user will be able to configure their own servers to use. They are outlined below.

Entropy generation service

For devices that are low entropy, for people that do not trust their own entropy, or devices where entropy generation would take a long time, an entropy generation service can be provided for such cases. Large numbers of entropy could be generated from something as simple as lava lamps as shown in (Tom Scott, 2017) described in (Joshua Liebow-Feeser, 2017).

Point to IPs

For devices that do not have a static IP address, a solution can be used where the service can point to the other devices' public IP address. This allows for connections to occur directly without going through the web server. This solution increases privacy and lowers bandwidth used by the server.

This service will keep pings sent from the devices and the time the device has pinged back to the server and keep them to allow proper syncing. To increase privacy and to protect individuals, the data will be fully encrypted by the individual's master key.

Middle-point

For devices that cannot connect directly to each other, the sever will need to facilitate users to tunnel traffic through the server to their devices. This will make the server act as a relay and allow devices to connect behind strict firewalls. The data passing through the server will be device-to-device encrypted. This means that there will be no visible data being transferred over the server - moreover,

Application

The application will provide the storage and the front-end interfaces for the user. The storage will be securely encrypted on the device and will be inaccessible when the password manager is not in use.

Password Database

The application will keep it's own version of the database of passwords as well as the metadata associated with them. The user will be able to add any metadata associated with a password. This includes fields that they define. The default fields will be:

- Website Name
- Website URL
- Password
- Username

The application will utilise the localStorage Web API (Mozilla, 2021) which allows up to 5MB of storage to be used. In internal experimental testing, a typical user exported Firefox Lockwise passwords to a CSV file. There were in total 304 passwords contained in the file. Uncompressed passwords exported from Firefox Lockwise only took 54 KiB (exactly 55,369 bytes) of storage.

$$\frac{5\,MB}{55{,}369\,Bytes} * 304\,Passwords = 27{,}452\,Passwords$$

Figure 4 - Password exhaustion calculation

Figure 4 gives an estimate that more than 27,452 passwords would exhaust the 5MB limit, which realistically no user would reach given the user has been using the password manager (with a migrated password list from another password manager) for 5 years. This means that this limitation is only likely to occur in very specific scenarios (e.g. user tries to add a thousands of passwords).

Sync Activity

The application will sync automatically when there is internet access. The sync will be done via the most optimal and secure route that the algorithm chooses.

Bibliography

Adi Shamir (1979) ‘How to Share a Secret’, *Massachusetts Institute of Technology*, 22(11), pp. 612–613.

cryptsetup (2020) ‘Cryptsetup and LUKS - open-source disk encryption’, *What the ...?*, 21 December. Available at: https://gitlab.com/cryptsetup/cryptsetup (Accessed: 14 January 2021).

Joshua Liebow-Feeser (2017) ‘LavaRand in Production: The Nitty-Gritty Technical Details’, *The Cloudflare Blog*, 11 June. Available at: https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details/ (Accessed: 14 January 2021).

Mozilla (2021) ‘Window.localStorage’, *MDN Web Docs*, 11 January. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (Accessed: 14 January 2021).

Tom Scott (2017) ‘The Lava Lamps That Help Keep The Internet Secure’, 6 November. Available at: https://www.youtube.com/watch?v=1cUUfMeOijg (Accessed: 14 January 2021).

Table of Figures