Password Storage

Adam Jens Jóelsson[1 [s202094]], Jeff Gyldenbrand[2 [s202790]], and Jón Ágúst Hannesson[3 [s202057]]

[1] Technical University of Denmark, Copenhagen, Denmark,
s202094@student.dtu.dk
[2] Technical University of Denmark, Copenhagen, Denmark,
s202790@student.dtu.dk
[3] Technical University of Denmark, Copenhagen, Denmark
s202057@student.dtu.dk

Abstract. This report presents the design and evaluation of a secure password store ,run locally on your computer and on the web. The design makes use of AES encryptions aswell as password hashing functions. In this design all security- and functional requirements made at the start were met. A user can store all of his passwords in one place and get instant access to them using a master password created at signup.

Keywords: Encryption · Security · Passwords

1 Introduction

The system described in this report is an client-server password manager. A password manager is a software application which allows users to store passwords and sign-in details for multiple services securely. The application can also generate strong passwords. The user only has to set up and remember one *Master password* which will replace all the endless different passwords for different platforms. All the online credentials are then encrypted in an online vault which is only accessible by the user with his *Master password*, which gives him instant access to any credentials he might need. The application will offer two-factor authentication via SMS for extra security. To secure the user from a malicious server, all encryption/decryption will occur on the client side and will the server never have access to any passwords or credentials inside the vault.

1.1 Definitions, abbreviations, symbols

AES	Advanced Encryption Standard
CBS	Cipher Block Chaining
HTTPS	Hypertext Transfer Protocol Secure
TLS	Transport Layer Security
SSL	Secure Sockets Layer
PBKDF-2	Password-Based Key Derivation Function
SMTPS	Simple Mail Transfer Protocol Secure
SFTP	Secure File Transfer Protocol,
VPN	Virtual Private Network
NIST	National Institute of Standards and Technology
2FA	Two factors of authentication
DoS	Denial of Service
SYN	Synchronize
ACK	Acknowledge
SYN-ACK	Acknowledge of sync.
HDW	Hardware
MP	Master Password
ISP	Internet Service Provider

2 Requirements

When designing a system it is important to figure out the requirements of the system. Once these are known, they can be addressed in the design.

2.1 Functional

The functional requirements describe what the user should be able to achieve with the system. The functional requirements for our system are as follows:

- The system should allow users to store multiple passwords and login details and get access to them with a single master password
- The system should allow users to store other sensitive data, e.g. credit card info, security codes, sensitive notes
- The system should allow users to access their passwords from multiple machines.
- Users should be able to view their passwords, even though they are offline.
- The system should include a password generation tool.
- The client should be available on multiple platforms.

2.2 Security

The security requirements describe what security features should be present in the system. The security requirements of our system are as follows:

- Implement Proper Password Strength Controls on Master Password
- Zero knowledge encryption (vendor should not be able to view user data)
- Data should be encrypted locally, before transit
- Data should be decrypted locally
- The system should offer two factor authentication (2FA)
- The system must be resistant to DoS attacks
- Login information should not be stored in plain-text
- Encryption keys should not be stored on the server

3 Risk Analysis

The risks in our system are considered by finding the assets that our system contains and defining the values of said assets[1]. Based on our assets, we have build a threat model that takes into account any vulnerabilities in our system[3.2]. From this we are able to postulate the likelihood of exploitation with some certainty[3.3].

3.1 Assets

The assets of our system consists of both hardware, software and data. The hardware of our system is the physical servers. These are costly if damaged or lost, but replaceable. The software running our system is not open source, which makes it hard to analyze for an adversary looking for weaknesses. The data is arguable the most valuable asset. If data is lost or corrupted, it can be very detrimental to the company's reputation and lead to loss of users.

Asset	Confidentiality	Integrity	Availability
Hardware			
- Servers			DoS
- Clients			Loss of hardware
Software			
- Server software		Tampered	Certificate Expires
- Client software		Tampered	
Data			
- Master password	Disclosure, Guess	Modified	Lost/Deleted
- Passwords	Disclosure	Modified	Lost/Deleted

Table 1: Assets

3.2 Threat model

Our threat model is based on the Dolev Yao model[8] where an adversery can overhear, intercept, and synthesize any message, and thus is only limited by our cryptograhic means. This means that the adversary can take the roles of any party: users, server administrators etc.

What can go wrong:

Data Our system maintains one database with three tables; *Users*, *PW Data* and *2FA*. As observed in the database table[2], the only data not hashed or encrypted is the salt in the user table, and the phone number and type chosen, in the 2FA table. In the case of a rouge administrator or an hacker gains access to the server database, this adversary cannot read any vital information. However, the phone number is still exposed and thus the adversary can link the phone number to the corresponding hashed username and salt. A possible attack would be for an adversary to perform social engineering by calling the phone number and lure out the plaintext username and possible other information to crack the password.

DoS As observed in the asset table[1] an DoS-attack of the server can affect the availability of the system to legitimately users. An adversary can do packet-flooding attack where the server is flooded with SYN-requests. These are normally used to initiate and establish a connection, however here, the adversary, never finalizes the connection.

To counter this our server uses *SYN cookies* to sends a hashed SYN-ACK back to the client, constructed from the clients IP-address and port number. This hash is then included in the clients ACK, and only then, the server, allocates memory for the connection.

HDW Stensikker A/S only have one server running. This means a single point of failure if that servers hardware fails somehow. This could be due to

natural disasters, theft, water damage etc.

To prevent this we could have multiple servers running concurrently in a distributed system on physically different places. However, due to the added complexity, we only have one. To mitigate the problem, we daily do a backup of the entire system. This backup is kept on two different offline devices. If a system crashes backup hardware is available and the backup would be installed. Expected time before the system is running again is within few ours from crash.

3.3 Likelihood of exploitation

Establishing Risk Levels: Without any concrete data its not possible to provide exact estimations to the likelihood of exploitation of our system. Thus we estimates it on subjective probability based on expert opinions on the algorithms used in this project. Thats why we, with confidence, consider the overall confidentiality and integrity of our system very high. The algorithms we use are considered some of the best encryption and hashing algorithms *AES-CBC* and *Pbkdf2*. These are explained in greater details in section[5.1]. Furthermore, as explained, we deal with potential problem of a DoS-attack or loss of hardware. Based on this, we have created a risk diagram that shows the likelihood of an exploitation on confidentiality[1], availability[2] of our system.

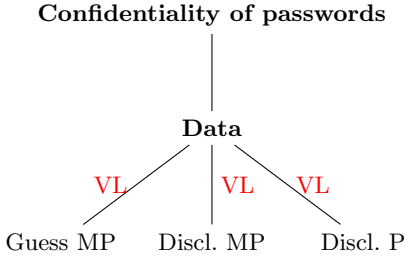


Fig. 1: VL = Very low., MP = Master Password, P = Password

Guessing of master password is considered very low. We enforce users to pick a master password on at least eight characters with at least one uppercase letter, one special character and one number. We calculate the password entropy[7], E, from: $E = \log_2(R^L)$ where R is all symbols in the set: upper- and lowercase letters 52, numbers are 0-9, special characters are 32. The length, $L \geq 8$. From equation (1) we observe that an attacker needs at most $2^{52.4}$ guesses to crack the password.

$$\log_2(94^8) = 52.4 bits \tag{1}$$

Disclosure of master passwords are also considered very low because this password is only known to the user. If our servers and databases we publicly leaked, only a hashed version of the master password would be disclosed. The same goes for **disclosure of passwords**, these are stored encrypted and only decrypted locally at the user side.

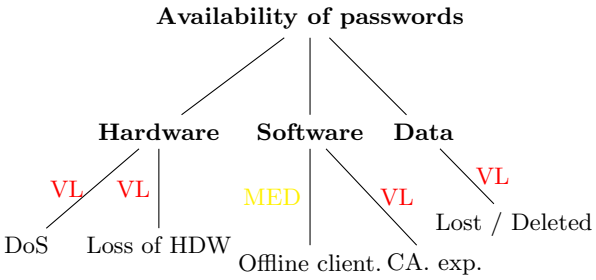


Fig. 2: VL = Very low. Med = Medium

Denial of Service-attacks on hardware is considered low. As mentioned earlier we mitigate this in order to prevent exhausting of both hardware resources. Natural disasters in Denmark are not frequent, or more precisely non-existent. As mentioned earlier we have a solid backup plan in case of **loss of hardware**, thus this is considered very low risk.

Offline client. When the client is offline, users can't retrieve their passwords. This is considered to be a more likely risk, because ISP downtime or general loss of connection happens from time to time. Even though there exists cases where **CA**s have been exploited or certificates have been issued to a malicious actor, confidentiality of CAs are still considered strong. Renewal of certificates is easy and a reminder tells it in advance of expiration. Therefore, the risk on availability is considered very low.

Lost or deleted data is considered very low because of the daily backups and separation of said backups.

3.4 Usability

Our system is fairly easy to use for the general user, meaning that users do have to download the software and install it before use, instead of a browser or website-based password manager, where they could directly create their user and login. However that trade off would result in a larger software project that would spawn even more security concerns that had to be addressed. The larger and more complex a system is, the greater the risks of errors in both software itself, between the services in the system and potential vulnerabilities an adversary can exploit.

Our system focuses on being simple, robust and rather secure; both in the sense of integrity and availability. This trade off is unfortunately, a little bit, at the expense of usability. However, the sign up process[4.2] is rather simple and designed for the average user. Furthermore, the Client application[4.6] allows for offline functionality, which increases usability even more.

4 Design

This section will dive deeper into the description of the system. Figure[3] illustrates an overview of the system. Particulary it shows the process of a user recieving a password from either local- or cloud storage.

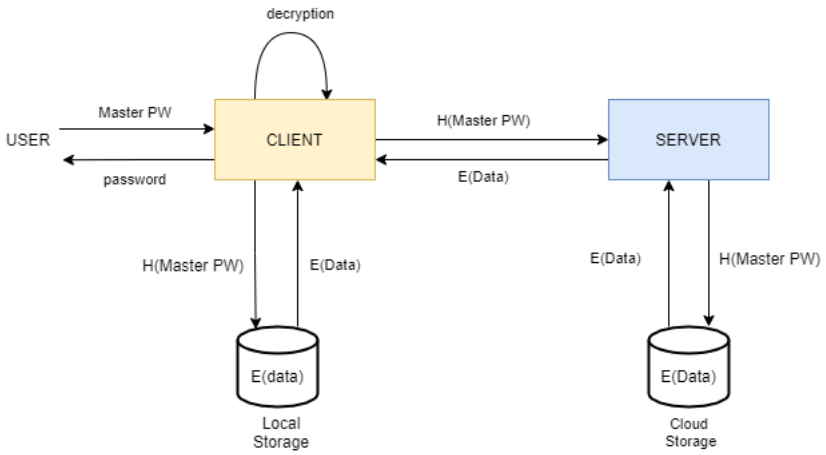


Fig. 3: Overview of the system

The system uses a client/server architecture. Each user installs the desktop application downloaded from the Stensikker A/S website at https://stensikker.org. The download of the application is done over the *TLS/SSL*-encrypted protocol, *HTTPS*, to ensure the integrity of the application. This is elaborated in greater details in section 4.1. The mobile application can be downloaded from the Apple App store or the Google Play store, depending on the users platform.

4.1 Certificates & HTTPS

The website for Stensikker A/S has an *SSL Certificate* file in its server which secure encrypted communication between the site and its users. This certificate consists of a key pair: a public and a private key, which together can establish

encrypted connections. The certificate is digitally signed by a trusted *CA*. This verifies the Stensikker A/S websites identity is authenticated. This ensures the integrity of downloads of the desktop application, and prevents *man in the middle*-attacks. A simplified description of the steps is illustrated below.

TLS/SSL protocol:

- **Browser:** A user connects to https://stensikker.org. His browser requests that the server identify itself.
- **Server:** Now the server response by sending a copy of its TLS/SSL certificate and its public key.
- **Browser:** The browser now validates the certificate against a list of trusted certificate authorities. The browser also checks that the certificate is not expired.
- **Browser:** The browser then creates a symmetric session key, encrypts it with the servers public key and sends it to the server.
- **Server:** The server decrypts the session key with its private key and send back an acknowledgment encrypted with the session key.
- **Server:** The server and the users browser now have a secure channel.

4.2 Signup process

In the client application users are able to create a new account or login to an existing one. When creating a new account, the user must provide a username, which acts as an identifier for the user. The server then checks whether the hash of the username is found in the user table. If it is already in use then the user is notified and has to pick another username. Furthermore he must choose a master password for the account. The user is asked to repeat the master password to ensure simple typing mistakes. Lastly, the user has the option to provide a hint for his password, in case the user forgets his password. The authentication is covered in section 4.4. During signup a 128-bit salt is generated by a Pseudo random number generator and saved server-side.

4.3 Server

The server acts like a cloud storage solution in our system. The server stores three tables, one for user information, one for two-factor authentication information and the third one for the password storage.

The user table has four columns, the hash of the username, the salt, the salted hash of the password and the password hint. The storage table has two columns, one for the hashed username and another for the encrypted data, which consists of login information and password to multiple different platforms for the user. The 2FA table has five columns, one for the hashed username, one for the users phone number, a timestamp, a hashed IP column and a type column. The 2FA is covered in detail in section 4.5.

User table

Hashed user	Salt	Hashed password	Hint
...	...	...	...
...	...	...	...

Vault data table

Hashed user	encrypted data
...	...
...	...

2FA table

Hashed user	Phone #	Timestamp	IP	Type
...	...	...	...	...
...	...	...	...	...

Table 2: Database

4.4 Authentication

The system uses the master password both for authentication and as an encryption key. The server stores the hashed password in a table, linking it to a salt and the hashed username.

If the password hash in the user table has the same or less iterations, an intruder could learn the encryption key simply by hashing the password hash again. To prevent this the authentication hash is a hash of the encryption key, which is derived from the master password with a hashing function.

If we let h be some cryptographic hashing function, then the relationship between the master password, encryption key and authentication hash would be as follows:

$$h(master\ password + salt) = encryption\ key \tag{2}$$

$$h(encryption\ key + salt) = authentication\ hash \tag{3}$$

As cryptographic hash functions are "one way", it is easy to derive the encryption key and authentication hash from master password, but not the other way around. The same goes for retrieving the authentication hash from the encryption key.

To prevent rainbow-table attacks the system uses a salt for the the authentication hash. The salt is generated when the user signs up for the system and is kept in the user table. The salt consists of characters, generated by a Pseudo-random number generator.

When signing in, the client sends the hashed username to the server and the server then looks up user and replies with the salt if the hashed username is valid. User then hashes the password,with the salt 500.000 times and saves that hash in local memory to later use as encryption key. The client then also hashes the password, with the salt, 1,000,000 times and sends the hashed password to the server. The server checks whether the hashed username and if password hash match. If it is valid log-in information, the server checks whether the user has

opted for 2FA. If the user has opted in for some type of 2FA, the server goes through that process and once complete, the server returns the encrypted data matching to that hashed username.

4.5 2FA

The 2FA uses a phone number as the second factor. This is an optional feature, as some users don't care for the extra security. The 2FA is configurable and users can choose from one of the four options:

- None: No 2FA
- Every New IP: The user is required to use 2FA, every time they log in from a new IP-address
- Every New IP + two weeks: The user is required to use 2FA, every time they log in from a new IP-address and once every two weeks
- Every time: The user is required to use 2FA every time they log in.

A user's known IP-addresses are kept in the 2FA table. This column is a string containing the hash of every IP-address the users has logged in from, separated by and ";". When a new IP-address has to be added, it is concatenated to the string. When a user, who chooses to use this feature, logs in the server hashes their IP-address and checks if the hash is in the IP column. If it isn't the user is required to use 2FA.

The timestamp keeps track of the last time the user used 2FA. If the users chooses so, they can be required to use 2FA once every two weeks (at least). For those users, during login, the server checks the timestamp to see if it is older than two weeks, then they are required to use 2FA, otherwise not.

4.6 Client

Each user installs the client locally to use the system. The user uses the client to sign up, login and view their stored data. All data is decrypted/encrypted locally by the client, so all data is encrypted in transport. All decrypted data is never stored on file, only kept in memory during use. When signing in to the server, the client hashes the username and the master password and sends them to the server. The master password is hashed enough times locally to get the authentication hash, instead of sending the encryption key and hashing that again server-side.

To allow offline functionality, the client can store encrypted data on file, similarly to the server. Thus the client can store encrypted data on file, for offline use. When the client is online, it syncs their data to that of the server. This offline functionality would be optional.

Users can download the client as an desktop application or a mobile app.

4.7 Cryptographic Functions

The system will use PBKDF-2 as a key derivation function and AES-CBC as an encryption algorithm.

4.8 Password Generation

The client application has the tool for users to generate strong randomized passwords with the click of a button. By using *Shannon Entropy* for passwords, defined in section[3.3], the generated passwords are defined to have the length, $L = 14$. Of this set three is designated numbers, $0 - 9$, two symbols, 32, and nine upper- and lowercase letters, $26 * 2$. An example of such generated password is seen below:

password = **GV2fYFC!o59w@H**

This means an adversary would have to, at most, perform $2^{91.76}$ guesses to crack the password:

$$\log_2(94^{14}) = 91.76bits \tag{4}$$

When the password is generated, the user can save it to the local- or cloud storage.

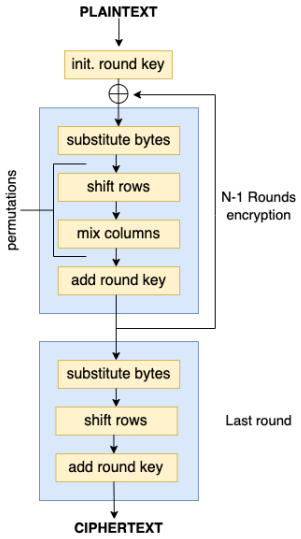
5 Evaluation and analysis

5.1 The algorithms used in our project

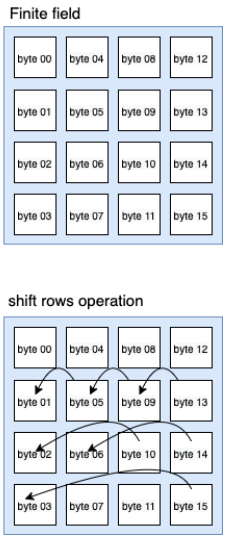
AES-CBC For the vault encryption, we have chosen *AES-CBC*, for encryption of stored data on both local and cloud storage.

How it works:
AES is a symmetric key algorithm used for both encryption and decryption of data. For encryption of a message, the message is split into blocks of 128-bits. A key of 128-, 192- or 256 bits is chosen for the algorithm, see figure[4a]. Each of the blocks is then combined with the key by a bitwise XOR-operation, and run through the AES-algorithm, $n - 1$ rounds, where $n = 10$, 12 or 15, depending on the key bit-size chosen. Each of these rounds performs operations, all done in a $4x4$ grid, also referred as a finite field, containing one byte in each cell, see figure[4b]:

- **Substitution:** the cells of the grid is substituted accordingly to a substitution-box, also called a lookup table, where each byte is mapped to another byte based on a function in this field. Some important points to mention is that no bytes are substituted with itself and no opposite bytes exists, E.g 10111010 $\rightarrow$ 01000101, to ensure as much complexity in obscuring the relationship between the key and the ciphertext as possible.



(a) Block Cipher Encryption.



(b) Top: 4x4 Matrix (finite field), containing a 128-bit message block. Each cell is one byte. Bottom: shows the row operation.

Fig. 4: AES-CBC algorithm

- **Shifting rows:** the three last rows of the grid is shifted to the left. First row is not changed. Second row is shifted left by one cell. The third row is shifted left by two cells and the fourth row, three cells.
- **Mixing columns:** each of the columns are multiplied with a fixed matrix. This together with the shifting rows-operation provides diffusion in the cipher, which means statistical relationships between the ciphertext and the plaintext is well hidden.
- **Round keys:** initially, AES bitwise XOR's the initial 128-bit key into the first round. From here, the key is expanded to the $n-1$ rounds. The key is derived by doing bitwise XOR on each of the columns in its own block. When the round key is generated, it then bitwise XOR's into the next round, until the last round is reached, where a last key-expansion is done.

Why we choose AES-CBS:
The *AES* is one of the most popular algorithm in the world, when it comes to data encryption, and is used widely in many applications and protocols such as SMTP, FTPS and HTTPS and VPNs. AES was chosen and is endorsed by

NIST. Even NSA uses AES to autherize transmission of classified data on top secret level[4], thus a natural choice for our implementation.

AES comes with several modes of operation which makes it applicable to a lot of specific areas of applications. In our case, we are using the CBS mode of operation because it is very good with plaintext encryption or data that is repeated a lot.

Pbkdf2 In our system we use Pbkdf2 (Password based key derivation function 2) to derive encryption keys from password [6]. This function returns a fixed length output from various length inputs. To use this algorithm a few parameters must be set to achieve the right output.

- PRF: Pseudo random function
- P:Password
- S: Salt
- c: Iteration count
- dkLen: Length of desired key (in bits)

And the output will be:

- DK: Derived key

The algorithm uses the PRF to derive a key from a password and a salt. The inputs we will use to derive the are:

$$DK = Pbkdf2(HMAC - SHA256, Password, Salt, 500000, 256) \quad (5)$$

With P as the master password, the derived key DK is the key used to encrypt the users data. To derive the password hash for user authentication, P is still the master password, but ,c, the iteration count is set to 1,000,000.We decided to use 500,000 and 1,000,000 iterations to significantly slow down any brute force attacks.

Why we chose Pbkdf2

There are many different Key derivation functions cryptographers can chose from. We chose to use Pbkdf2 because of it's widespread use in similar systems 1Password[1], Bitwarden[2], and the recommendation from NIST [3].

The algorithm is also quite flexible, as the amount of computation required can be tuned with the number of iterations and the size of the salt.

6 State of the art

Currently there are many different solutions available for password storage. The design of these systems is very similar, and our system is no exception. The use

[4] https://techjury.net/blog/what-is-aes/#gref

of a master password, to retrieve the encrypted password, can be seen in most systems.

Some systems offer a recovery process in case the user forgets the master password. Nordpass for example supply the user with recovery code, which they can enter in case they forget the master password [4]. In the case of lastpass, they offer a password recovery process in their mobile app and a password hint as well [5].

Most systems offer a variety of platforms to download their client applications. Desktop applications, mobile apps and browser extensions are common among these kinds of systems, where the browser extensions can automatically insert passwords to password fields on web pages.

7 Group contribution

Section	Adam	Jeff	Jón
Introduction	30%	30%	40%
Requirements	35%	30%	35%
Risk Analysis	30%	35%	35%
Design	35%	35%	30%
Evaluation and analysis	35%	35%	30%
State of the art	30%	35%	35%
This list	100%	0%	0%

References

1. 1Password Homepage, https://support.1password.com/pbkdf2/. Last accessed 31. Oct 2021
2. Bitwarden Homepage, https://bitwarden.com/help/article/what-encryption-is-used/. Last accessed 31. Oct 2021
3. M. Sönmex Turan, E. Barker, W. Burr, L. Chen:Recommendation for Password-Based Key Derivation. National Institute of Standards and Technology (2010)
4. Nordpass homepage, https://support.nordpass.com/hc/en-us/articles/360002376557-How-to-generate-a-new-Recovery-Code-. Last accessed 31. Oct 2021
5. Lastpass homepage, https://blog.lastpass.com/2019/05/never-lose-access-lastpass-account-recovery-mobile/. Last accessed 31. Oct 2021
6. B. Kaliski:PKCS #5: Password-Based Cryptography Specification Version 2.0. RSA Laboratories (2000)
7. Shannon Entropy, slide 14, http://www.cs.cornell.edu/courses/cs5430/2016sp/l/15-passwords2/lec.pdf.
8. Dolev Yao model, https://ieeexplore-ieee-org.proxy.findit.dtu.dk/stamp/stamp.jsp?tp=&arnumber=1056650.