Piotr Smietana
CS:4640
Research Project

For my research project, I decided to design a <Local, No Autofill> password manager because I do not support the use of online password managers. I simply do not trust software that I did not write myself. Some people might argue “Why wouldn’t you trust a company like LastPass to take care of your passwords?” The answer is simple. Because they are the biggest in the business and everyone wants to hack them.

§1. What is the general threat model for a password manager of your type?

Since I am designing a local password manager, I am assuming that the attacker has access to a computer with the password manager installed on it. I am also assuming that the attacker can read, modify, and delete any files used by the password manager. Moreover, I am assuming that the attacker knows the structure and design of the password manager. The attacker can inspect the source code of the password manager to find any flaws in the implementation and exploit them. I am assuming that the attacker can run reverse lookup, rainbow table, dictionary, and brute-force attacks to try to crack the master password. Lastly, I am assuming that the attacker could try to inspect the memory of the password manager to extract any sensitive data.

§2. What are the general security properties that one can expect for a password manager of your type?

The first security property expected from a password manager is confidentiality. This means that only a person who knows the master password should be able to read sensitive data stored by the program. An attacker should not be able to view or determine any information about the stored records. This security property can be achieved through encryption.

The second security property expected from a password manager is integrity. This means that a file with stored passwords can be only modified by a person who knows the master password and is using the password manager. If an attacker modifies the encrypted file with stored passwords, we will be able to detect it. This security property can be achieved through a message authentication code.

The last security property is availability. This means that users should have continuous access to the password manager. Because I am designing a local password manager, I am not concerned about this security property. The user will always have reliable and uninterrupted access to the password manager through the command-line interface. Therefore, this security property is always achieved.

§3. Give the overall design of the password manager.

1 OVERVIEW

1.1 FILE STRUCTURE

The password manager will be a command-line Java program that consists of two files. The first file, called “stored_passwords”, will be an encrypted file that contains 0 or more records. A record is an object with a domain, username, and password.

The second file, called “master_password”, will be a file that stores a salt followed by a hash of the master password. This file will be used for checking if the user entered a correct master password by computing a hash of the entered password and comparing it with the stored value.

1.2 REGISTRATION

When the password manager is launched, it will look for the two files listed above. If one or both files are missing, then the password manager will assume that the user is either using the password manager for the first time or someone tried to attack the password manager. In this case, the password manager will take the user through a setup process where a user will create a master password. In the case that both files are present, the password manager will take the user through the login process.

1.3 LOGIN PROCESS

The program will ask the user for the master password allowing three failed attempts. The password manager will compute a salted hash of the entered password and it will check whether the computed hash matches the stored hash. If the user fails all the attempts, the password manager will terminate itself. After entering a correct master password, the program will check the integrity of the file with stored passwords. This is to check that no one manipulated the stored passwords in an unauthorized way. If the integrity check passes, then the program will successfully login the user. If the integrity check fails, then the password manager will terminate itself.

1.4 OTHER OPERATIONS

generateRandomPassword() – This function is responsible for generating a secure random password that consists of at least two lowercase characters, two uppercase characters, two digits, and two special characters. The default length of the random password is 32 characters. Using a secure random generator, the function will randomly pick two lowercase characters, two uppercase characters, two digits, and two special characters. Next, the function will pick 24 remaining characters. Finally, the function will shuffle all characters to randomize the string even more.

storePassword() – This function is responsible for storing a new password for a particular domain and username. The function takes domain and username as input and checks if a record already exists for that particular domain and username. If not, then the function asks the user if they want to generate a random password or if they want to enter their own password. If a record already exists, then a user will receive an error. Finally, the function checks if the input has between 4 and 80 characters and consists of only allowed characters.

getPassword() – This function is responsible for retrieving a stored password for a particular domain and username. The function takes domain and username as input and checks if a record already exists for that particular domain and username. If a record exists, then a password is printed to the user. Otherwise, an error message is displayed.

checkIfRecordExists() – This function is responsible for checking if a record exists. It takes domain and username as arguments and iterates through the list of records checking if a record has that particular domain and username. The result is printed to the user.

removePassword() – This function is responsible for removing a stored password for a particular domain and username. The function takes domain and username as input and checks if a record already exists for that particular domain and username. If a record exists, then the function removes that record from the stored passwords. If a record doesn't exist, then a user will receive an error.

changePassword() – This function is responsible for changing a stored password for a particular domain and username. The function takes domain and username as input and checks if a record already exists for that particular domain and username. If a record exists, then the function removes that record from the stored passwords and then adds a new record. If a record doesn't exist, then a user will receive an error. Next, the function asks the user if they want to generate a random password or if they want to enter their own password. Finally, the function checks if the input has between 4 and 80 characters and consists of only allowed characters.

2 SECURITY

The password manager will have 256-bit security. Because of the birthday attack, the length of any hash functions should double the key length of block ciphers. Therefore, any encryption will be done using 256-bit keys and hashing will be done using 512 bits.

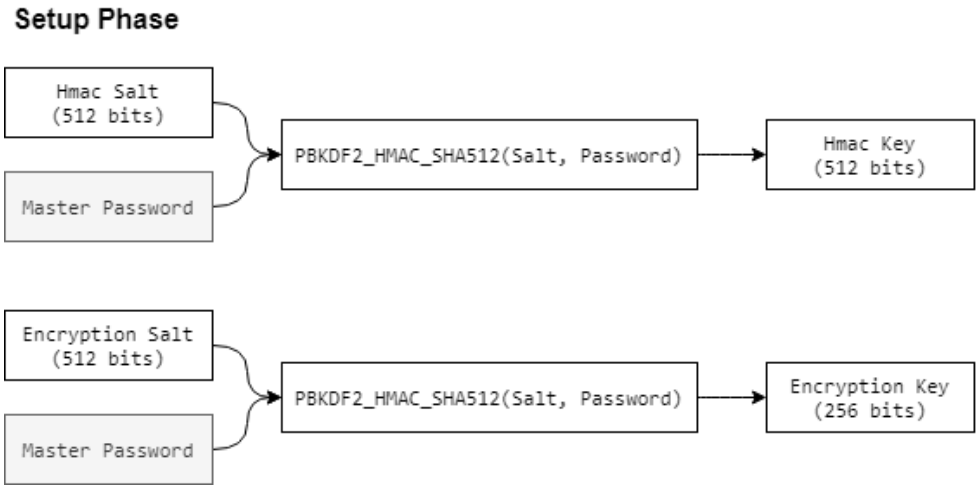
2.1 CONFIDENTIALITY

- Encryption and decryption will be done using AES-256 in the CBC mode with PKCS5Padding.
- The length of the initialization vector (IV) will be 256 bits.
    - Generated using a cryptographically secure random generator.
    - A new IV will be generated after every operation to ensure that two files with the same stored passwords will never have the same signature.
- The length of the key will be 256 bits.
    - Generated using password-based key derivation function (PBKDF2).

2.2 INTEGRITY

- Integrity will be guaranteed using HMAC-SHA512.
- The length of the key will be 512 bits.
    - Generated using password-based key derivation function (PBKDF2).

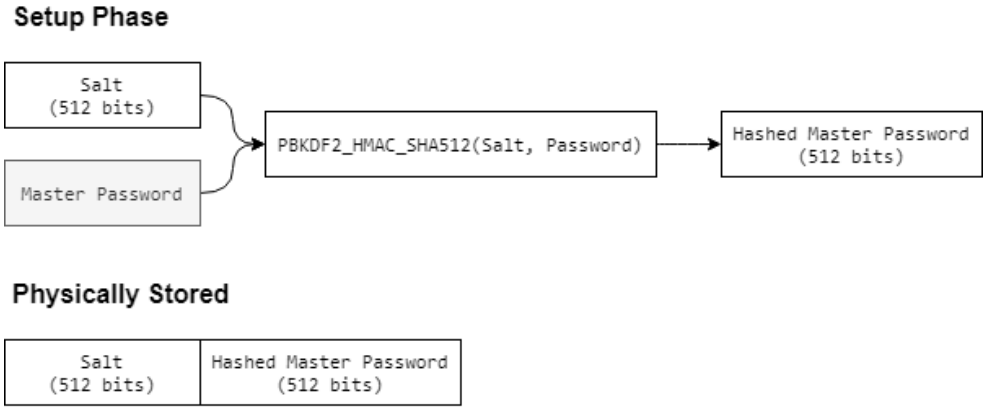
2.3 Deriving Hmac and Encryption Keys



HMAC and encryption keys will be used to provide confidentiality and integrity. Therefore, these keys should be cryptographically strong and secure. The way to do this is to use a key stretching algorithm that converts a weak key, usually a password, into a long random key that can be used for encryption. In this case, the password manager will be using PBKDF2_HMAC_SHA512 to generate both keys.

- The length of the salt will be 512 bits.
    - Generated using a cryptographically secure random generator.
- The number of iterations will be 65536.
    - This allows a modern laptop to compute one hash per second.

2.4 Hash of the Master Password

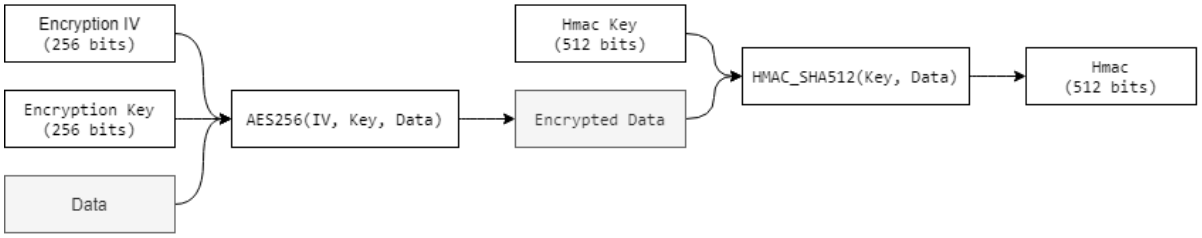


A salt followed by a hash of the master password will be stored in the file called “master_password” to authenticate the user. The salt will provide protection against any attacks that use reverse lookup or rainbow tables. The hash will be generated using PBKDF2_HMAC_SHA512 which is a key derivation function that is designed to be computationally expensive to make brute-force attacks difficult. This means that key derivation functions should be used for creating an encryption key and storing a hashed password.

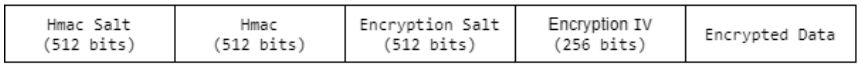
§4. Give the detailed format of your password file in which you are going to store the username, password, and website information.

1 OVERVIEW

Workflow



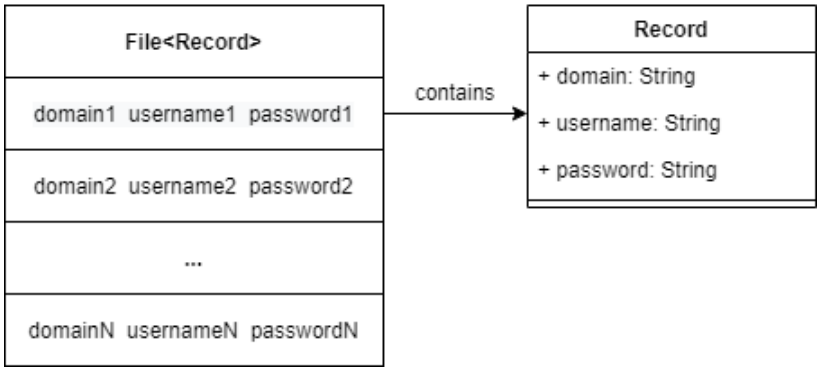
Physically Stored



The file called “stored_passwords” will physically store an HMAC salt, HMAC, encryption salt, encryption IV, and encrypted records. As mentioned in the previous section, HMAC and encryption salts will be used with the master password to derive HMAC and encryption keys. The password manager will use an encryption IV, encryption key, and stored records to encrypt the sensitive data. Then, the program will use the HMAC key and encrypted data to generate an HMAC signature which will be used to verify the integrity of the data. This model is known as cryptography as the “Encrypt-then-MAC” scheme.

2 RECORDS

Format



The file called “stored_passwords”, will be an encrypted file that contains 0 or more records. A record is an object with a domain, username, and password. Record fields will be separated by a space and multiple records will be separated by a newline. This design will allow the user to use all possible characters to store a record. This can be easily implemented in a programming language like Java. For example, a scanner object can be used to read input from the user. The scanner object will automatically escape all special characters and will make sure that any line feed characters will be compatible among all operating systems.

2.1 INSERTION

insert() – This function is responsible for adding a new record to the file. The function takes a record as input and decrypts the encrypted file. Next, the function writes a record into the file followed by a new line feed. A new IV is generated to ensure that two files with the same stored passwords will never have the same signature. Finally, the function encrypts all records with the new IV and recomputes the HMAC signature.

2.2 DELETION

delete() – This function is responsible for deleting an existing record inside the file. The function takes a record as input and decrypts the encrypted file. Next, the function searches for an existing record and deletes that line from the file. A new IV is generated to ensure that two files with the same stored passwords will never have the same signature. Finally, the function encrypts all records with the new IV and recomputes the HMAC signature.

2.3 SEARCH

search() – This function is responsible for searching for an existing record inside the file. The function takes a record as input and decrypts the encrypted file. Next, the function iterates through the file and check if each line is equal to the searched record. If yes, then the function returns the line number. Otherwise, the function returns -1.

§5. What is the considered threat model of your password manager in comparison with Question 1? If you can support the full threat model of Question 1, then explicitly mention how you are protecting them.

Addressing the threat model:

1. Since I am designing a local password manager, I am assuming that the attacker has access to a computer with the password manager installed on it.
   → I will make sure that the password manager gives confidentiality and integrity guarantees.
2. I am assuming that the attacker knows the structure and design of the password manager. The attacker can inspect the source code of the password manager to find any flaws in the implementation and exploit them.
   → I will use built-in security libraries and secure random number generators for cryptographic operations.
   → I will use well-known cryptographic algorithms that do not have any vulnerabilities.
3. I am assuming that the attacker can run reverse lookup, rainbow table, dictionary, and brute-force attacks to try to crack the master password.
   → I will use a salt for calculating the hash of the master password.
   → I will use a slow hashing algorithm for calculating the hash of the master password.
4. I am assuming that the attacker could try to inspect the memory of the password manager to extract any sensitive data.
   → I will use a memory secure programming language like Java. Java runs inside a sandbox, has garbage collection, index checking, doesn't use pointers, and supports private keywords.

§6. What security properties from Question 2 can your design fulfill?

My design can fulfill the following security properties:

- Confidentiality of the stored passwords through encryption.
- Integrity of the stored password through HMAC.
- Confidentiality of the master password through sow hashing.
- Availability of the password manager at all times.

It does not guarantee:

- Integrity of the master password
    - Anyone could generate and replace the hash of the master password to successfully login into the password manager. However, they will not be able to read any of the encrypted data since the stored password are encrypted using a derived key that is based on the original master password.
    - In the end, the only way to obtain access to decrypted stored passwords is to brute force the hash of the master password.
- Freshness
    - If an attacker somehow obtains a copy of the encrypted file with stored passwords and replaces the current version with that copy, the password manager will not be able to detect an older version of the password file.
    - Ideally, the user should keep the password manager on a USB flash drive. The USB should be stored in a safe offline location. This drastically reduced the chances of the attackers getting access to the encrypted stored passwords.

§7. Implement the password manager in any programming language.

Password manager implemented in Java. Please check the zip file!

Works Cited

*"*1Password Security Design." *1Password*, www.1password.com/files/1Password-White-Paper-v0.3.1.pdf.

"How PBKDF2 Strengthens Your Master Password." *1Password*, www.support.1password.com/pbkdf2/.

Reichl, Dominik. "Security - KeePass." *KeePass*, www.keepass.info/help/base/security.html.