

Donald Willetts

CS4640: Computer Security

May 12, 2021

Designing a Secure Password Manager <Local, No Autofill>

Question 1 (5 points): What is the general threat model for a password manager of your type?

A general threat model for my password manager <Local, no autofill> is a brute force attack on the master-file where all the password information is stored. Although a local, no autofill password manager is slightly less convenient for the user, having no autofill security properties (i.e., not having to worry about the password manager's autofill leaking password information to an adversary). It is still a secure powerful tool used to both store and generate random passwords.

Question 2 (5 points): What are the general security properties that one can expect for a password manager of your chosen type?

General security properties that one can expect for a <local, no autofill> password manager is using a secure password algorithm (AES) to secure the master-file and account information. Also, this password manager should be able to create strong cryptographically secure passwords. Characteristics of strong passwords include: At least 8 characters, mixture of uppercase and lowercase letters, mixture of letters and numbers, at least one special character, e.g. (! @ # ?).

Question 3 (50 points): Give you overall design of the password manager in light of the 7 operations discussed above (e.g., generate a random and unpredictable password for an account).

1. Generate a random and unpredictable password.

To generate a random and unpredictable password for my password manager I am going to have it be 10 characters in length, a mixture of uppercase and lowercase letters, digits, and 2 special characters. I am going to build a string taking 2 randomly selected (using `secureRandomGen`) uppercase letters, two lowercase letters, two digits and two special symbols. The last two characters will be randomly selected (again using `secureRandomGen`) of any uppercase, lowercase, digit, or special symbol. I will then use a shuffle function to shuffle all the characters before returning the new password to the user.

2. Store username and password

To store the username and password I will have a master-file that is encrypted using the master password. I will also store the username and password by encryption. The purpose of storing the username and password with encryption is if, somehow an adversary was able to get ahold of the unencrypted master-file the username and passwords would still be secure. The main method will be using salt created from the first 16 bytes of the master-password (The only requirement of a salt is to be

globally unique) and taking 16 bytes from the master-password to create the IV and a different 16 bytes to create the key. I will then store this username and password into the master-file.

3. Retrieve username and passwords.

To retrieve username and passwords I will decrypt and search my master-file for the account name field and return the encrypted account username and password. I will then decrypt the username and password then return to the user as a string.

4. Check if username and password for particular account.

This is the same as above. Decrypt the master-file and then search for the account name. If found return a Boolean stating if the account was found.

5. Register an account and create master password.

To register an account and create a master-password the password manager will first search the directory to see if it can find an already encrypted master-file. This will then decide to ask the user to enter their master-password or to create the master-file and generate or enter a master-password. If the user chooses to generate a master-password, one will be created for them. If the user chooses to enter their own master-password, they will be prompted to enter it.

6. Remove account.

To remove an account the password manager will first check if the account exists. If the account exists, the password generator will delete that account in the master-file where it is stored. I am going to read and write data to the master-file as a `ArrayList<String>` data type. This way I can easily perform insertion and deletion.

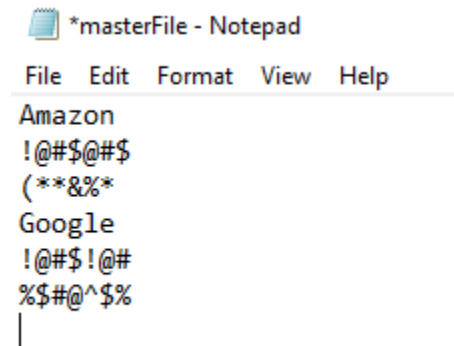
7. Change account password

To change an accounts password. The password manager will query the master-file for the specific account and then replace the old password with the newly encrypted password. Using the `.replace` method for `ArrayLists`.

Question 4 (20 points): Give the detailed format of your password file in which you are going to store the username, password, and website information.

The password file has two main-levels of security. The first level is the encryption of the password file itself. Before and insertion, deletion and search operation are performed the master-password must decrypt the password file. The second level is the contents themselves being encrypted. Each username and password are salted and encrypted. Ideally, my password file will be read and written to from my password manager as an `ArrayList<String>` (so I can easily perform operations with build in methods) According to *Why people (don't) use password managers effectively* by Sarah Pearman, Shikun Aerin Zhang, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, most users that have password-protected accounts have less than 100 accounts to protect. Therefore, I am not concerned about my program running in $O(n)$ time for searches.

My password file is formatted as an ArrayList<String> [accountName, username, password, accountName2, username, password, ...] with the username and password also being encrypted. Figure 1 shows an example of why my passwordfile should look like.



```
*masterFile - Notepad
File Edit Format View Help
Amazon
!@#$$@#$
(**&%*
Google
!@#$$!@#
%$#@^$%
|
```

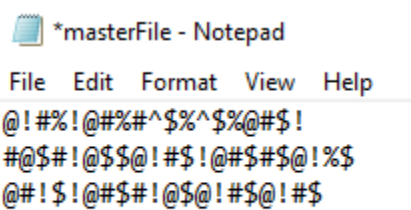
Figure 1. Decrypted password file.

For insertion, I will simply add the new accountName, username, password to the end of the file (after encrypting the username and password).

For searching, I will read the entire file in as a ArrayList<String> and search for the accountName, then return that index and the indexes of the that accountName's username and password.

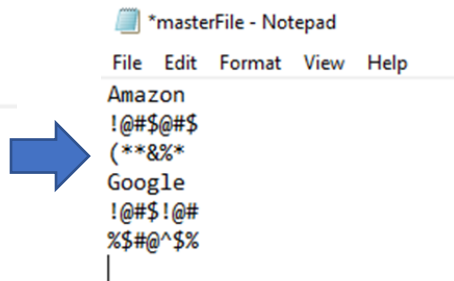
For deletion, I will search the ArrayList<String> and get the indexes of the accountName, username and password. And call a deletion method on them. ArrayList<String>'s .remove(index). Then clear the password file and rewrite the ArrayList<String> to the file.

A graphical representation of the passwordfile (masterFile) is as follows:



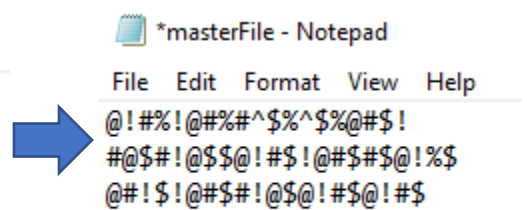
```
*masterFile - Notepad
File Edit Format View Help
@!#%!@#%#^$%^$#@#$!
#@$#!@$$$@!#$!@#$$$@!%$
@#!$!@#$$!@$@!#$@!#$
```

Figure 2. Encrypted password file.



```
*masterFile - Notepad
File Edit Format View Help
Amazon
!@#$$@#$
(**&%*
Google
!@#$$!@#
%$#@^$%
|
```

Figure 3. Decrypted password file.
(Run: insertion, search, and deletion
on file here)



```
*masterFile - Notepad
File Edit Format View Help
@!#%!@#%#^$%^$#@#$!
#@$#!@$$$@!#$!@#$$$@!%$
@#!$!@#$$!@$@!#$@!#$
```

Figure 4. Re-encrypted password file after
functions are called.

Question 5 (5 points): What is the considered threat model of your password designed in comparison

with Question 1? If you can support the full threat model of Question 1, then explicitly mention how you are protecting them.

I can support the full threat model of Question 1, by two main factors. One, by having a strong password generator that generates cryptographically strong passwords. Two, by using a secure encryption algorithm for both my master-password-file and my username and passwords. This should ensure a strong defense against any brute-force attack.

Question 6 (7+8 = 15 points): What security properties from Question 2 can your design fulfill? For each such property, clearly explain how your design achieves it.

1. Password generation, as stated above, creates a 10-character secure password.

```
public static String createSecurePassword(){
    //Ciataion I referenced this code while creating my password creator: https://mkyong.com/java/java-password-generator-example/
    //https://crypto.stackexchange.com/questions/41436/is-deriving-the-iv-from-the-password-secure

    final String CHAR_LOWERCASE = "abcdefghijklmnopqrstuvwxyz";
    final String CHAR_UPPERCASE = CHAR_LOWERCASE.toUpperCase();
    final String DIGIT = "0123456789";
    final String SPECIAL_SYMBOLS = "!@#&()-[{}];',?/*~$^+=<>";
    final String ALL_CHARACTERS = CHAR_LOWERCASE + CHAR_UPPERCASE + DIGIT + SPECIAL_SYMBOLS;

    //Using this to append characters for my password gen @ the end I will shuffle this to make it random
    StringBuilder password = new StringBuilder(10);

    //get 2 lowercase letters
    password.append(getRandomCharacters(CHAR_LOWERCASE, 2));

    //get 2 uppercase letters
    password.append(getRandomCharacters(CHAR_UPPERCASE, 2));

    //get 2 digits
    password.append(getRandomCharacters(DIGIT, 2));

    //get 2 special symbols
    password.append(getRandomCharacters(SPECIAL_SYMBOLS, 2));

    //get 2 random
    password.append(getRandomCharacters(ALL_CHARACTERS, 2));

    //shuffle characters
    List<String> shufflePassword = Arrays.asList(password.toString().split(""));
    Collections.shuffle(shufflePassword);
    String finalPassword = shufflePassword.stream().collect(Collectors.joining());

    return (finalPassword);
}
```

Figure 5. createSecurePassword() function.

```
private static String getRandomCharacters(String inputString, int amountToSendBack){
    try{
        //Code sourced from https://howtodoinjava.com/java8/secure-random-number-generation/
        SecureRandom secureRandomGen = SecureRandom.getInstance("SHA1PRNG","SUN");

        StringBuilder returnStr = new StringBuilder(amountToSendBack);

        for (int i = 0; i < amountToSendBack; i++){
            //get random int between 0 and the length of the strong
            int randomInteger = secureRandomGen.nextInt(inputString.length()-1);

            returnStr.append(inputString.charAt(randomInteger));
        }

        return (returnStr.toString());
    }
    catch (Exception e){
        e.printStackTrace();
    }

    return (null);
}
```

Figure 6. Helper function getRandomCharacters() for createSecurePassword() function.

Figures 5 & 6 show how I generated a secure random password. I first used a string builder to create my password. I then called my helper function `getRandomCharacters()` to pick two truly random indexes (`secureRandomGen`) and return those characters as elements of the password. I then called `getRandomCharacters()` on lowercase, uppercase, digit, special_symbols and appended the results to my string builder. Finally, all these strings were added together and called `getRandomCharacters()` to get 2 completely random characters from all the options. I then shuffled this string and returned it to the user for a secure random password.

I used a secure AES encryption algorithm to secure my master-file and user passwords. Figure 7 & 8, displays the code I used. (I based this code off homework 3. I made modifications to it so that it will encrypt the single input file).

```
// Generate IvParameterSpec object from IV Raw Data
IvParameterSpec ivspec = new IvParameterSpec(iv);

// Generate AES Key through KeyGenerator and SecretKey (handle exception with try-catch)
SecretKey skey = new SecretKeySpec(key, "AES");

//Create a Cipher object with "AES/CBC/PKCS5Padding" (handle exception with try-catch)
Cipher ci = Cipher.getInstance("AES/CBC/PKCS5Padding");

// Initialize the cipher object with the secret key and IV parameter spec obtain before
ci.init(Cipher.ENCRYPT_MODE, skey, ivspec);

//Call the doEncryption function to actually encrypt the file and output the ciphertext in the output file
doEncryptionAndWriteToFile(ci);
```

Figure 7 `setupAndEncrypt()`

```
private static void doEncryptionAndWriteToFile(Cipher cipherContext)
{
    try{

        //File masterFile = new File("C:\\Users\\donwi\\Documents\\GitHub\\Password_Manager\\masterFile.txt");
        File masterFile = new File("masterFile.txt");
        Path path = masterFile.toPath();

        byte[] masterFileRawContent = Files.readAllBytes(path);

        byte[] masterFileEncrypted = cipherContext.doFinal(masterFileRawContent);

        //setting append to false so old data is overwritten (more secure than being deleted)
        FileOutputStream newMasterFile = new FileOutputStream(masterFile, false);

        newMasterFile.write(masterFileEncrypted);
        newMasterFile.close();
    }
}
```

Figure 8 `doEncryptionAndWriteToFile()`

```

public static byte[] getIVFromPassword(String password){
    try{
        //citation: https://howtodoinjava.com/java/java-security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/
        //citation: https://crypto.stackexchange.com/questions/41436/is-deriving-the-iv-from-the-password-secure

        //salt that I am using is going to be the first 16 bytes of the password (only require of a salt is to be globally unique)
        byte[] salt = Arrays.copyOfRange(password.getBytes(), 0, 16);

        PBEKeySpec passwordPBKspec = new PBEKeySpec(password.toCharArray(), salt, 1024, 256);
        SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        byte[] createdPBEKS = skf.generateSecret(passwordPBKspec).getEncoded();

        //IV is the next 16 bytes after key
        byte[] IV = Arrays.copyOfRange(createdPBEKS, 16, 32);

        return IV;
    }
}

```

Figure 9 getIVFromPassword() NOTE: getKeyFromPassword() is the same, it just takes range (16, 32)

In Figure 7, the function setupAndEncrypt() takes input of iv and key generated from the master-password (seen in figure 9). The iv and key are created from the master-password using PBKDF. These are then sent to my setupAndEncrypt() function that creates a cipher that then calls my doEncryptionAndWriteToFile() function, that actually encrypts my file.

Bonus Question (50 points): Implement in any programming language the 7 functionality mentioned above in the Project Description Section according to you design described in Questions 3 and 4.

See README.txt

Works Cited

Pearman, Sarah, et al. "Why People (Don't) Use Password Managers Effectively." *Fifteenth Symposium on Usable Privacy and Security*, vol. 15, 12 Aug. 2019, pp. 319–338.