CS 331: Computer Security Fundamentals **Fall 2023**

Homework 2 (Individual Programming Assignment)

Total 100 Points

Due date & time: 11:59 pm on October 13, 2023.

Late Policy: You have three extra days in total for all your homeworks and projects. Any portion of a day used counts as one day; that is, you have to use integer number of late days each time. If you exhaust your three late days, any late homework or project will not be graded.

Submission: For the written part of your assignment, you should have your answers in a PDF file. Please do not submit hand-written assignment. For the programming part of the assignment, please just submit your Python files. All the files (Python and PDF) should be zipped and then submitted. For submitting the assignment, you will use Brightspace.

1 Motivation

In the current cyberspace, user authentication is done using passwords. As we will see in class, passwords are very useful and convenient as a user authentication mechanism. However, there is an inherent tussle between the *usability* and *security* of passwords. For security, it is expected that users will generate unexpected or unpredictable passwords. However, unpredictable passwords are often difficult to recall hampering usability severely.

To make matters worse, for better security, there are many advices such as the following: (1) *users should have different passwords for different accounts*; (2) *users should change their passwords at least bi-annually*; (3) *Each password should contain characters from certain group of alphanumeric set and special character set, and cannot have any words from the dictionary.* Compliance with all these advices, however, creates a serious *information overload* on the users. That is, a user has to remember n different passwords (where n can be as large as 106, according to an old survey) of n different accounts where each password has to satisfy some password composition policies and each of these passwords would have to be changed twice every year. Managing this information overload is infeasible, if not impossible. As a result, more often than not, it has been shown that users tend to side with usability over security.

As a convenient mechanism for addressing this information overloading without hampering security, ***password managers*** were proposed. Most of the password managers require the user to remember a single master password that is used to protect all passwords of its other accounts. Essentially, password managers reduces the mental overload of remembering n different passwords for n different accounts, to memorizing or recalling only 1 password and username pair (i.e., the *master username and password*) that is used to authenticate the user to the password manager.

Underlying goal of this project. The overarching goal of this programming project is to implement such a working password manager in `Python` using a cryptographic library called `pycryptodome`. ***You are given some scaffolding or skeleton code which will you will complete in this assignment. After finishing the programming part of the assignment, you will answer some questions about the assignment.***. The other goal of the project is to provide hands-on

training to students regarding how to effectively use cryptographic constructs to achieve better security.

2 Design Principle

Roughly, existing password managers or vaults can be categorized into the following three dimensions: (1) Storing the vault or encrypted password file in the cloud, local storage, or both; (2) Recoverability of the master password; (3) Availability of the autofill functionality.

The password vault or manager you are going to implement will have the following features: (a) the encrypted password vault will be stored only in the local storage; (b) there is no Recoverability of the master passwords (if you forget the master password then you will lose access to the vault); (c) there is no auto-filling feature; (d) one does not need to open an account to use the vault.

To implement such a password vault, one has to consider the following main things: (1) To ensure that the password vault is secure (both confidentiality and integrity), the password vault file will never be stored in local storage in plaintext; (2) how does one store the symmetric keys that are used to encrypt and integrity-protect the password vault file; (3) how does the vault operate without requiring the users to open up an account.

Symmetric Key Storage Problem. Instead of storing the key in the disk, the key will be generated dynamically from the user's master password in memory. For this, you will use the `scrypt` function from the `pycryptodome` library (see `https://pycryptodome.readthedocs.io/en/latest/src/protocol/kdf.html#scrypt`).

User account problem. Instead of using a database where the user password is stored, the password manager takes a different approach because then security boils down to protecting the database where user information is stored. The vault essentially hashes the master user account using the `sha256` function from the `pycryptodome` library (see `https://pycryptodome.readthedocs.io/en/latest/src/hash/sha256.html#sha-256`) and then names the password vault file the resulting digest. *To essentially see whether the current user has an account already, you calculate the SHA256 hash of the master username and see whether there is a file in that name.* If a file is present already, then it suggests that user already has an account. Otherwise, the user will be presented to create a new account by creating the file.

Format of a password vault in plaintext. In plaintext, a password vault file can be viewed as a list, where each element of the list appears in a single line. Each of these lines has the following format: ⟨username : password : domain⟩. As you can see, the username, password, and domain name fields are separated by the ":" character. Hence, username, password, or domain name fields cannot have the ":" character in them.

Before encrypting the plaintext vault file in memory and then storing it in a file, one should ***prepend*** the magic string "10101010101010101020202020202020202020303030303030303030" followed by a new line character to the list of ⟨username : password : domain⟩.

The reason for magic string. One may wonder what is the goal of the magic string discussed just above. This is to protect against the following situation. Consider a user who opened an account

(i.e., created a password vault file) with the username "random" and password "doublerandom". Another user came in and chose the username "random" and password "triplerandom". In case of the second user, your password vault program would find a corresponding file because the first user used the same account number. When you try to decrypt the file of the first user using the key derived from the password "triplerandom", obviously the decryption will fail. However, how would one programmatically figure out that the decryption failed. The best way to figure out is that whether the decrypted file has the magic string in the first line. If it has the magic string, then you can safely conclude that the file was successfully decrypted. Otherwise, you can conclude that one of the following situation occurred without being able to tell exactly which one: (1) *the user has forgotten the master password*; (2) *the user currently attempting to log in is a new user.*

3 Cryptography and Authentication Details

Programming Language. For the programming part of things, we will only allow using Python.

Skeleton Code. The Skeleton code for the project can be found in the following Google Colab link: `https://colab.research.google.com/drive/1Q-si56GOlOy0drkiy67vBTprLs85ygWk?usp=sharing`. Please make sure you copy it in your own Colab account and submit the Python files.

External Packages. You cannot use any other packages except the ones that are already included in the Skeleton.

Cryptography library Information. For this assignment we are going to use the `Crypto` or `pycryptodome`. Note that, `pycryptodome` has the same APIs as in the `Crypto` package.

Encryption Key Size. The key size should be 128 bits or 16 bytes.

Cipher Information. You should use `AES GCM` with empty header.

Hash Function. You should use `SHA256`.

Key Derivation Function. You should use the `scrypt` function with the constant salt = "<\n<~\x0e\xeetGR\xfe;\xec_\xfc)8". In addition, you should use the following default arguments: $N = 2 * *14, r = 8, p = 1$.

Password Composition and Size. When the user expects your program to generate a password, then the password should be randomly generated. The length of the password should be 16. Passwords can contain characters from the following three sets: $\{A, B, \cdots, Z\}$; $\{a, b, \cdots, z\}$; $\{0, 1, 2, \cdots, 9\}$.

Encoding. For storing the ciphertext and all the relevant data to file, you can consider encoding it in base64 using UTF-8 encoding.

Notation. The skeleton file is heavily annotated. When writing a function signature, I use the following notation: $\texttt{F} : \texttt{A} \times \texttt{B} \rightarrow \texttt{R}_1 \times \texttt{R}_2$. It signifies that the function F takes two arguments. The first one is of type A and the second one is of type B. It then returns a pair of which the first element is of type R_1 and the second element is of type R_2. An example could be:

`computeMasterKey` : `String` → `A byte string`. This signifies that the `computeMasterKey` function takes a string and returns a string of bytes.

4 Functionality to be Implemented (88 points)

You will implement the following functionality.

Generate an Encryption Key. Generate a Master key using `scrypt` for the vault from the password entered by the user. Function to fill: `computerMasterKey(password)`.

Hashed Username. Use the `hashlib` library with `sha256` to generate a filename from the username input with an encoding('utf-8'). Assigne the value to the variable `hashedusername` in the function `checkVaultExistenceOrCreate`.

Encrypt a File. Use AES GCM to write a code that takes plaintextData and key and encrypts the data. This function outputs nonce, header, ciphertext and tag, encodes the results in base64. Function to fill: `encryptFile(plaintextData,key)`.

Decrypt a File. Use AES GCM write a code that takes an encryptedJson and key and decrypts the data. This function takes nonce, header, ciphertext and tag, and decodes the results from base64. Function to fill: `decryptFile(encryptedJson,key)`.

Generate Password. Implement functionality to generate a password with length 16 consisting of digits, uppercase and lowercase. Function to fill: `generatePassword()`.

Decrypt and Reconstruct Vault. Check if the beginning of the decoded content includes the magic string and if so remove the magic string from the decoded content otherwise throw an error. Function to fill: `decryptAndReconstructVault(hashedusername, password)`.

Add an Account. Create a function that accepts two user inputs; username and domain and generates a random password and stores it in this format [username:password:domain] and adds the result to vault as an entry. Function to fill: `CreatePassword(passwordvault)`.

Add password. Create a function that accepts three user inputs; username, password and domain and stores it is in this format [username:password:domain] and adds the result to vault as an entry. Function to fill: `AddPassword(passwordvault)`.

Delete an Account. Create a function that accepts a domain as an input and deletes the existing password for that domain. Function to fill: `DeletePassword(passwordvault)`.

Update an Account. Create a function that accepts a domain as an input and updates the existing password for that domain. Function to fill: `UpdatePassword(passwordvault)`.

Lookup an Account. Create a function that accepts a domain as an input and displays the existing password for that domain. Function to fill: `LookupPassword(passwordvault)`.

5 Questions to be Answered (12 points)

Problem 1. Write 2 desirable security properties (i.e., one integrity and one confidentiality) of a password manager.

Problem 2. Write 2 potential threat models one should consider when analyzing a password manager.

Problem 3. Analyze the 2 desirable security properties you have written in Problem 1 with respect to one threat model you have described in Problem 2.