
HASHWORD - CRYPTOGRAPHIC PASSWORD MANAGER & GENERATOR

Cieara Pfeifer, Jarrett Woo, & Jayden Stearns

CS-47206: Data Security and Privacy
Kent State University

December 10, 2021

Contents

1	Introduction	1
2	Password History	1
3	Password Manager Limitations	1
4	Hashing Background	2
5	Development Process	2
6	Hashword Implementation	2
6.1	Terminology	2
6.2	Functions	3
6.3	System Execution Demonstration	4
7	Conclusion	7
7.1	Future Improvements	7
7.2	Final Reflection	8

1 Introduction

The first interface most users encounter when starting a computer is usually a password field to gain access to the system. This trend follows for any accounts used on the internet. There is a strong emphasis on security when it comes to creating passwords. Not only should a password meet a set of requirements including as length, special character use, no personal information, etc., but sometimes need to be changed frequently to ensure your account remains secure. These passwords should also not be reused for other services, so that if one account is compromised another cannot be accessed as well. This leads to the issue of needing multiple passwords for a variety of sites that should ideally not be written down or saved anywhere. Thankfully, there exist specialized programs and internet browser extensions known as Password Managers that aim to solve these issues for users. However, these software systems are not without their own set of issues and security concerns. We aimed to provide a more secure solution to the problem of password management, keeping in mind especially the privacy fundamentalists for whom standard password managers may not be sufficiently secure.

2 Password History

When passwords were first invented in the 1960s, they were used to meter time on shared mainframe computer systems. Each user needed to input their password to gain access to the part of the system that stored their files. One of the first instances of password hacking was done by a student who wanted more time to use the console resources. He accomplished this by printing out a file that contained all the usernames and passwords on the shared mainframe. This event helped lead to the idea of encrypting passwords. The primary method developed was the process of "hashing": translating passwords into bit sequences using one-way encryption. This irreversible sequence of bits was then stored to the system instead of the plain-text password itself. In 1979, the security of hashing was further improved by adding "salt" to make encryption stronger by the addition of extra characters to the encryption. This process can improve security by ensuring that the same password between users will result in completely different hashed values.

Today, general password security, even with all the new methods of encryption and technology, hasn't notably improved. The most common passwords are variants of 123456, a very weak password due to its simplicity. Additionally, around 66% of people still reuse passwords on multiple sites, despite 91% of people claiming to know that they should not. When users use common and repeated passwords, it makes it easier for attacks by malicious actors to be much more successful, and much more quickly. [1]

3 Password Manager Limitations

The average user has around 191 passwords. [1] As such, proper management of this information can be understandably difficult. As introduced above, password manager programs can help with this issue, storing the password data on the system, removing the need for the user to memorize many passwords in order to keep unique passwords between services. For example, many popular internet browsers are equipped with password management, allowing for auto-filling password information for web services. However, this convenience comes with its own security risk. To fill these password fields automatically, the password must be stored in plain-text on the system. Within seconds of access to a system, a malicious actor could view the username and password for every service using this password management. The breach of the single system's security resulted in the loss of confidentiality for all services whose passwords were stored within.

There also exists a more complicated form of password management: programs that generate their own strong passwords for use with individual services. These programs can ensure that individual service passwords are unique and much more secure than most human-created passwords. However, these password generators have their own limitations. While the passwords are incredibly strong, they are generated with essentially no relationship to the user. As such, if the user needed to use these passwords on a system that does not have their specific individual passwords stored, the process can be long and convoluted, typing the seemingly random series of characters into the password field. Even if the password manager program is installed on the

system, the exact password information itself is still required for proper ease-of-use, which in many cases is not possible due to security concerns.

4 Hashing Background

As previously mentioned, Hashing is a method of one-way encryption, meaning that once the data is encrypted it should no longer be able to be retrieved. However, if the same input is put in multiple times the same hashed output will always be produced. If there is even a minor change to the input (such as a single additional letter) the hash output will change significantly. All that is needed to encrypt is the input data; no key is necessary, unlike other forms of encryption. As such, hashing is neither symmetric or asymmetric. [2]

Essentially all widely-supported hashing algorithms produce a fixed-length output sequence regardless of the length of the input. For example, inputs of 6 or 20 characters will both produce outputs of length 256 bits when using the SHA-256 algorithm. [2] The primary benefit of hashing for our implementation is that, once hashed, the original input data cannot be recovered. Additionally, the key space of the output sequence is large enough that it provides a decent amount of security against traditional attacks against passwords, especially including Brute Force Attacks.

5 Development Process

Hashword is implemented using Python version 3.9.9. Notable Python libraries used include hashlib, getpass, and pyperclip. Python was chosen to be our language of choice because of its improved readability, familiarity between members of our group, and the wide variety of libraries to use as a resource. The hashlib library implements a wide variety of hashing algorithms and functionality. The getpass library includes functionality to privately and securely retrieve password input from the user via the command line. The pyperclip library allows for direct communication with the user's clipboard, for the purposes of added security of not displaying the resulting service password, but simply copying it to the user's clipboard.

GitHub was used as our version control system to collaborate on our code, which allowed us the following functionality: developing the system remotely from our local systems; creating issues to assign to members to distribute work; and keep track of any bugs and development progress.

6 Hashword Implementation

6.1 Terminology

Listed below are common terms and their definitions as used below to describe the implementation of the Hashword system.

- **Concatenation** - The process of appending textual information to the end of another string for the purposes of producing a new string. For example, concatenating the string "Hello,_" and the string "World!" will yield the string "Hello,_World!". Phrased another way, "World!" is appended to the end of string "Hello,_".
- **Current Directory (*also known as Current Folder*)** - The directory (folder) in which the main executable of the Hashword program is stored. This is considered only for the purposes of file path checking during the execution of the Hashword program.
- **Master Password** - The single password for use in the Hashword program. A hash of the master password is stored within the Service Information File. The plain-text of the Master Password

is concatenated with the service name, then hashed to produce the password to be used with the individual service in question.

- **Register [verb]** - The process of adding a service and all of its associated details into the Service Information File. Conducted automatically by the program when needed, as determined by user input.
- **Service** - An individual external system which requires a password to access. Examples may include: Amazon, Gmail, Overleaf, etc.
- **Service Information File (*also known as Service Data File*)** - The data file into which details about the individual services are registered and stored. The name of this file (default: ServiceData.dat) is checked for within the current directory. If not found, the Hashword program will create a new file automatically as part of its execution.

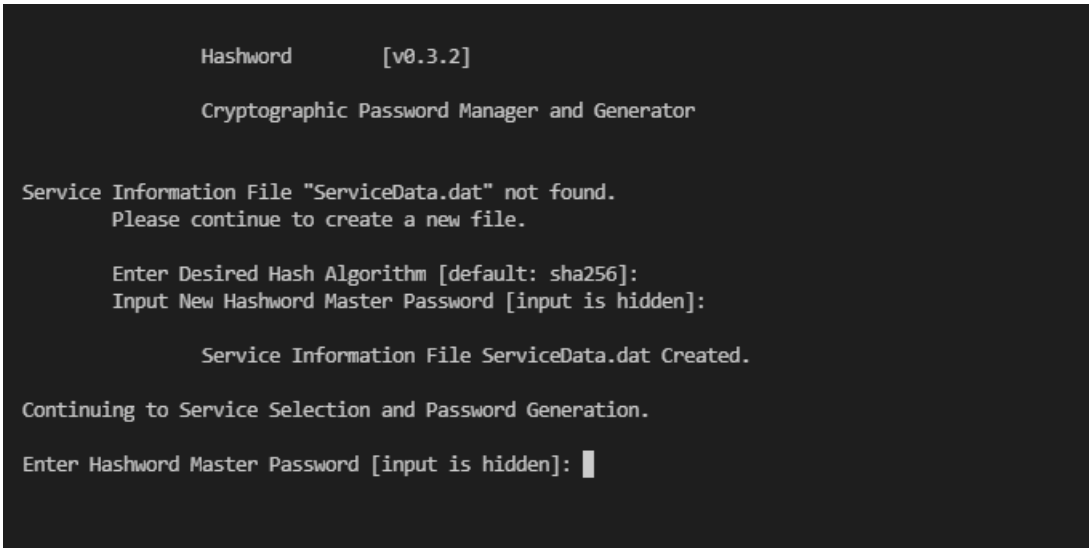
6.2 Functions

The Hashword system has three notable programmer-defined functions as listed below.

- **closeAndMoveFile (fileObject)**
 - Can be invoked if an error is found with a service information file.
 - Argument: the file object associated with the erroneous file in question.
 - This function will close the file, save its contents elsewhere, then delete the file. This cleans the program directory of the service information file, to be automatically re-made in the next execution of the program.
- **getMasterPass (masterPasswordPlaintext , hashAlgorithm)**
 - Invoked to handle the hashing and memory management of the plain-text master password entered by the user.
 - Arguments: a string containing the plain-text of the master password and a string containing the name of the hash algorithm to use to produce the resulting hash.
 - Returns: a string containing the hash of the master password using the given hashing algorithm.
 - Care is taken within the function to ensure that no instances of master password plain-text are left remaining in memory. When hashing is completed, the local variable containing the master password plain-text is overwritten and the associated object is deleted from memory.
- **verifyPass (passwordPlaintext , expectedHash , hashAlgorithm)**
 - Invoked to handle comparing the hashing of the given password and comparing the result to the given expected hash value.
 - Arguments: a string containing the plain-text of a password, a string containing the expected hash value for the password, and a string containing the name of the hash algorithm to use to produce the expected hash.
 - Returns: true if the hash of the given plain-text matches the expected hash value, otherwise false.
 - Care is taken within the function to ensure that no instances of the given password plain-text are left remaining in memory. When hashing is completed, the local variable containing the password plain-text is overwritten and the associated object is deleted from memory.

6.3 System Execution Demonstration

When executing Hashword, the first check performed is within the current directory for the existence of a Service Information File. If not located, it will create one, then prompt the user for the desired algorithm they would like to use for their passwords. If unknown, they can enter any invalid algorithm name to receive a complete list of algorithms available, or just press enter to default to SHA-256. After entering the choice of algorithm, the user is then prompted for the master password that will be used for the program. This information will be stored in the created Service Information File for the program to refer to in later executions. The user will then again be prompted to input the same password to login and access the Service Information File.



```
Hashword      [v0.3.2]

Cryptographic Password Manager and Generator

Service Information File "ServiceData.dat" not found.
Please continue to create a new file.

Enter Desired Hash Algorithm [default: sha256]:
Input New Hashword Master Password [input is hidden]:

Service Information File ServiceData.dat Created.

Continuing to Service Selection and Password Generation.

Enter Hashword Master Password [input is hidden]: █
```

Figure 1: New Service Information File initialization

The user will then be prompted for the service they are creating a password for. The program will search the file and check if that service has already been registered (in this case, Facebook). The registration process will begin by asking for the maximum character amount permitted for the service, or simply how long the user wants their password to be if shorter than the maximum length. In this case, a character length of 16 was chosen. The program will update the Service Information File with this information and proceed to copy the password to the user's clipboard for a total of 10 seconds before clearing it, so it cannot be accidentally pasted elsewhere, risking security.

```
Hashword [v0.3.2]

Cryptographic Password Manager and Generator

Service Information File "ServiceData.dat" not found.
Please continue to create a new file.

Enter Desired Hash Algorithm [default: sha256]:
Input New Hashword Master Password [input is hidden]:

Service Information File ServiceData.dat Created.

Continuing to Service Selection and Password Generation.

Enter Hashword Master Password [input is hidden]:

Login Successful

Enter Service Name: Facebook

Service "Facebook" not registered.
Beginning Registration to file "ServiceData.dat".

Enter Maximum Facebook Password Length: 16

New Service "Facebook" registered to Service Information File.

Copied Facebook Password to Clipboard.
```

Figure 2: New service registration sequence

Here is shown the Service Information File after executing the program. The user's choice of hashing algorithm was "sha256" (SHA-256). Following this is the hash of the user's master password. As a hash, this is cryptographically secure to store, because if someone were to copy this and enter it an entirely different string would result since the hash would simply be hashed again. Lastly, the service Facebook was registered with the maximum password length of 16.

```
Hashword > ≡ ServiceData.dat
1 hash:sha256
2 master:5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
3 Facebook:16
4
```

Figure 3: Service Information File

If the user were to login once again successfully, there is no need to create a new Service Information File as it was found in the current directory. Additionally, the service password is just able to be generated and copied to the user's clipboard since its associated service was previously registered.

```
Hashword [v0.3.2]

Cryptographic Password Manager and Generator

Enter Hashword Master Password [input is hidden]:

Login Successful

Enter Service Name: Facebook

Copied Facebook Password to Clipboard.
```

Figure 4: Returning User and Service

If the user were to enter the incorrect password as shown here, it will inform the user of this and prompt them for re-entry of the password. Now, Twitter will be a new service to be registered. Upon reading the existing Service Information File and not finding the service Twitter, the registration process will begin again.

```
Hashword [v0.3.2]

Cryptographic Password Manager and Generator

Enter Hashword Master Password [input is hidden]:

Login Failed
Please try again.

Enter Hashword Master Password [input is hidden]:

Login Successful

Enter Service Name: Twitter

Service "Twitter" not registered.
Beginning Registration to file "ServiceData.dat".

Enter Maximum Twitter Password Length: 16

New Service "Twitter" registered to Service Information File.

Copied Twitter Password to Clipboard.
```

Figure 5: Existing File Service Addition and Password Failure

Here is shown the updated Service Information File with Twitter and its password length max again of 16, for example.


```
Hashword > ServiceData.dat
1 hash:sha256
2 master:5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
3 Facebook:16
4 Twitter:16
5
```

Figure 6: Updated Service Information File

For demonstration purposes, the passwords for both of these services copied to the clipboard were pasted into a separate document. Although both of these services have the same password length and are registered to the same Service Information File (and therefore the same master password), they do not result in the same hash. This is because the service name itself is also concatenated to produce the resulting service information password.

```
Facebook: 92a2e47a60f80d&J
```

```
Twitter: 5ae0fcc3c0306c&J
```

Figure 7: Resultant Service Passwords

7 Conclusion

7.1 Future Improvements

When considering the realistic development for such a software system, we identify the following future improvements that would be useful for this system.

- Security of the resultant service passwords could be improved further by also salting the hashes. This would also add some functionality to the software system: if two different users had the same master password, the salt would result in a completely different service password.
- Tools for system administrator use could be a useful implementation. Specifically, such tools may allow for the Hashword system to be incorporated into an existent data security policy at the company level, for example.
- Password strength can be further improved by implementing a custom hash translation map. In the current implementation, a hexadecimal output is produced and used for the resultant password, resulting in a key space of 16 to the power of the maximum service password length. If a custom output translation was implemented, the number of possible characters may be able to be increased to 32, providing a drastic increase to individual service password complexity.
- For improved user convenience, more information could be stored about the services. Currently, only the name of the service (necessary for concatenation to produce the service password) and the maximum length of passwords (the length to which the hashed password will be truncated before finalization into the service password) are stored.

7.2 Final Reflection

Through this project, we collectively learned an impressive amount about our topic area. Hash algorithms are a very unique type of cryptography, and this served as an opportunity to research further into the fundamental characteristics of this encryption method. Additionally, when considering the importance of hashing in relation to passwords for our project, we discovered interesting information regarding the significance of secure passwords, including an introduction into password strength assessment. Finally, when implementing our project, we gained valuable experience with Python, our development language of choice, through working with the input and output handling and file management in the context of secure programming.

Through the presentations of other class members, we gained valuable insight into many aspects of data security and privacy. Ranging from topics tangentially related to our own, such as symmetric encryption, to topics entirely unique, such as machine learning and steganography, we gained interesting perspective into many facets of modern technology.

References

- [1] Web Desk. (2021, May 18). *The history and future of passwords (and what's next)*. Digital Information World. <https://www.digitalinformationworld.com/2020/05/what-comes-after-passwords-infographic.html>
- [2] Paar, C., & Pelzl, J. (2010). Hash functions. In *Understanding cryptography: A textbook for students and practitioners* (pp. 293-314). Springer. DOI 10.1007/978-3-642-04101-3