International Journal of Scientific Research in Science, Engineering and Technology
Print ISSN: 2395-1990 | Online ISSN : 2394-4099 (www.ijsrset.com)
doi : https://doi.org/10.32628/IJSRSET

Integrating Hashing with Encoding to Eliminate Password Managers

Aditya Ray[1], Aman Roy[2]
*[1]Student of Divya Bhaskar Public School, India
[2]Department of Computer Engineering, GHRCEM, India

ABSTRACT

Article Info
Volume 7 Issue 5
Page Number: 361-367
Publication Issue :
September-October-2020

Article History
Accepted : 15 Oct 2020
Published : 23 Oct 2020

The Internet has created a utopia which enables us to access any information. However, This boon comes with a curse. Everyone is prone to attack. Textual passwords remain one of the most common authentication methods [1]. For being safe online, using strong and different passwords for each website is the way to go. As easy as it looks, in the real world, it becomes very unmanageable. Most people do not follow this norm. Few people who do, end up using password managers who have their own issues. The solution which we came up with uses hashing and encoding to generate passwords. Hashing will be done using the Secure Hash Algorithm(SHA-256). As far as the encoding is concerned, we are going to design our own procedure for producing a pseudo-random combination of letters. There are three counterparts for generating the secret key. One is the website/app name, which is variable. Remaining two is constant, i.e. password and key length all the time. In this algorithm, the secrecy of password is only essential. This method will allow the user to eliminate the role of password managers and also need not to worry about the strength of the password. This should be used as an ideal way of keeping track of passwords.

Keywords: Password, Hashing, Encoding, SHA-256, Password managers

I. INTRODUCTION

Computer passwords have been used for authentication for a long time. In the year 1961, the password was used in a Compatible Time-Sharing System for the first time [2][3]. As time progressed, it grew tremendously, and with that, it became complicated day by day. At first, the passwords were saved in the system as plaintext, which was not safe at all and later got replaced with keeping the password in a hashed format. Now the narrative of *"passwords are dead"* is reaching its new heights, and people are looking for alternatives of authentication.

The use of passwords for online authentication is a great choice or not is out of this paper's scope. The fact is, according to Michael B., in 2014, an average person online has 40 login credentials [4]. In 2020, it would have been much more than that six years down the line.

Technology has grown at lightning speed but not humans. They still do not follow good practices for being safe online. Each person on the internet signs up for many websites. The websites use some login credentials for verification purposes. Generally, people make two mistakes while choosing a password -

1) Keeping a weak password.
2) Repeated use of the identical password everywhere.

A weak password is problematic because it can be broken using various techniques such as - dictionary attack or rainbow tables [5]. Using the same password for every website is not an optimal solution even though our fragile mind tends to go for that. It is risky because if one of our accounts gets hacked by someone, the attacker can use the same login credentials to access other websites. Being safe online can be possible only if we try to avoid the mistakes mentioned above. To use strong and non-identical passwords for each website is impractical and not usually practiced. For making it possible for the masses, password managers are used. They help users to keep a strong password without any headache of remembering it all. We just need to remember one master password which will be used to access all other passwords. However, being so accessible to use, it holds few threats that need to be addressed.

Problems imposed by password managers

Password managers can be widely categorized into two groups, i.e. offline and online.

As far as Offline Password Managers are concerned, it feels safe to have all the data locked inside one's own local computer. There is no trust involved in third parties. However, an offline system does not allow people to sync all the data to multiple devices which makes it impractical.

On the other hand, Online Password managers are good with syncing the passwords to multiple devices, but that feature comes with a price. Users have to trust the internet company in the middle who is making this possible. Also, there is a chance of a data breach because all the data is on the cloud.

Either way we have to compromise on one thing or the other.

Password Generator as an alternative

Storing login credentials either on a local machine or cloud has their own demerits. The better way of managing passwords can be achieved using password generators. Password Generators are a special kind of algorithm which produces a strong password. The solution of enabling users to follow good practices while choosing a password can be achieved using it.

Instead of generating a completely random password, the idea is to generate passwords according to the data passed by the user. Also, the data provided by the user will not be different each time except on field (website/app name). Only one password will be used to generate all the passwords, i.e. now the user is not saving any kind of data anywhere. All the passwords are getting generated on the go. The algorithm can be implemented across different platforms without any inconsistency. Even if any individual generated password gets disclosed, the attacker should not be able to recover the original secret key which is used to generate all the passwords.

II. PROPOSED WORK

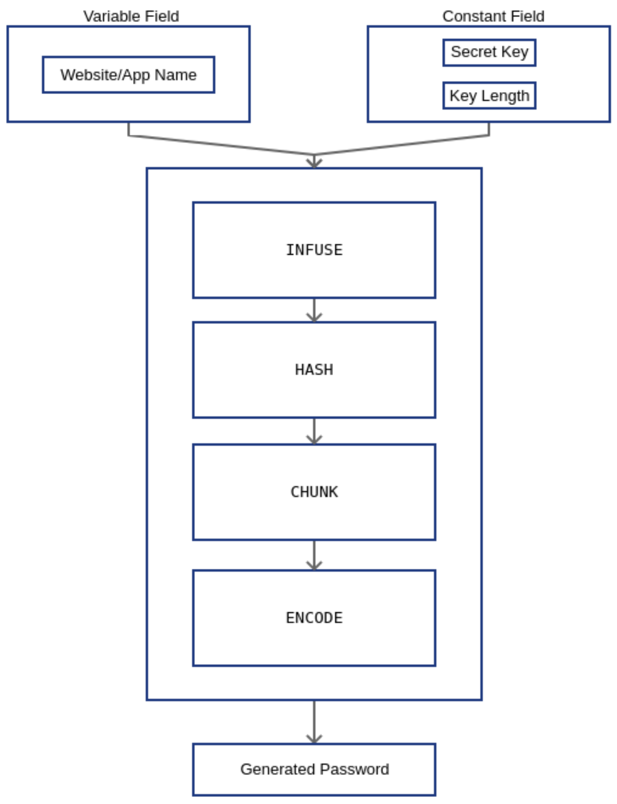

Figure 1: Complete Diagram of the proposed system for password generation

As shown in Fig. 1, the input consists of three counterparts. The feeded data goes into a series of operations which at the end gives a very strong password. The details of how exactly all the operations work under the hood will be discussed later into this section.

Input Data

The algorithm takes three inputs to compute the secret key. All the three counterparts can be categorized into two parts, i.e. variable and constant.

- ✓ Variable field - Only one field is going to change, which is the website/app name. The user does not need to remember it as it will be known by default explicitly.
- ✓ Constant field - Two fields will be constant for all the websites/app, i.e. secret key and key length. The key length is advised to be less than or equal to 20.

The generated password depends on all three components. Secrecy of secret keys is very crucial. Key length may or maynot be kept secret. Unlike these two website/app names will always be kept open. It is not advisable to keep it secret.

Procedure

After getting all the three inputs, the algorithm processes the input and gives the generated password as a result. The process works in four stages –

- ✓ Infuse
- ✓ Hash
- ✓ Chunk
- ✓ Encode

Stage - I [Infuse]

In this very first stage, the app name and password gets infused together. For doing that, in both the strings, one character is taken alternatively until one string is out of range. If still some part of a string is remaining then that will be reversed and added to the final result.

Let X be the website/app name, and Y be the password. Where $X, Y \in U$, $X \neq \phi$ and $Y \neq \phi$. $f(X,Y)$will be used to compute the final infused text if length of X and $Y$is equal.

if $X.len = Y.len$**, then -**

$$f(X,Y) = \sum_{k=0}^{X.len-1} (X_k + Y_k)$$

If $X$and $Y$are not equal, then using X and Y, first the largest string M and length of shortest string n is computed. $f(X,Y,M,n)$will be used to generate the infused text.

if $X.len \neq Y.len$**, then -**

$$largest\ string\ (M) = max(X,Y)$$
$$length\ of\ the\ shortest\ string\ (n) = min(X.len, Y.len)$$

$$f(X, Y, M, n) = \sum_{k=0}^{n-1} (X_k + Y_k) + \sum_{j = M.len-1}^{n} M_j$$

Here, it is assumed that the string starts from index 0.

For e.g., if website/app name(X) = "facebook"
and password(Y) = "pass"
Using X and Y, largest string (M)= "facebook"
length of shortest string (n)= 4
therefore, $f(X, Y, M, n)$ = "fpaacseskoob"

Stage - II [Hash]

The value generated after the infusion of the website/app name and password will be hashed now. For hashing purposes, Secure Hash Algorithm(SHA)-256 will be used. A hash function takes an arbitrary length of data as an input, but the output will always be fixed. There are few properties that a typical hash function fulfills [6]. They are -

- **Fast Computation** - If H(x) is the hashing function, it should not take a long time to output the result.
- **One way** - If H(x) has computed the hash of X as Y, i.e. H(x) = Y. If anyone knows Y's value, it should be infeasible to find out x with the help of Y.
- **Avalanche Effect** - A small change in input will completely change the output.
- **Deterministic** - Identical hash will be generated for the same input every time.
- **Negligible collision** - There should be very less chances of two different input data producing the
- same hash value. i.e. $H(x_1) = Y_1$and $H(x_2)\, Y_2$, if $Y_1 = Y_2$then $x_1 \neq x_2$or vice versa.

SHA-256 fulfills all the properties listed above [7][8]. It outputs a string of fixed length of 64 characters[9]. In SHA-256, 256 represents bits of the output string.

Size of one character = 4 bits.
No. of characters in output string = 64 characters.
No. of bits used = 64*4 = 256 bits.

So, the string derived from the last stage will be hashed using SHA-256. For example -

$$SHA256(\text{"}fpaacseskoob\text{"}) = aaefe\ldots(54\, char)\ldots.bba37$$

Stage - III [Chunk]

This step is just a bridge between the previous and the next step. It prepares the hashed value to be prepared for encoding.

The process converts the hashed string into a list of similar size of chunks. Before chunking, the last 4 characters of the hashed value are dropped. The size of the chunks is determined by - $floor(60/keyLength)$.

The number of chunks will be equal to the key length.

Let's say if the key length is 8, the size of each chunk will be $floor(60/8) = 7$. There will be 8 chunks, and each chunk will consist of 7 characters. The total character used will be 8*7 = 56 characters. In hash value, there are 64 characters, in which 56 characters are used. The remaining 8 characters(64 char - 56 char) will be dropped.

For example, the hashed value derived from the previous stage, if the key length is 8, can be chunked as - ['aaefedc', '69d578b', '095cf6c', '8e934e6', '01d78f2', 'fbb57b0', '18cd75a', '3912c63'].

The remaining part of the hashed value ('da0bba37') will be dropped.

Stage - IV [Encode]

This is the last piece of the whole puzzle. Here the chunked hash value will be encoded. Each chunk will give a single character.

For doing the encoding, a list of shuffled characters will be used. The list of shuffled characters that will be used in the process is not chosen at random without any reason. It is derived after shuffling letters randomly each time and running some tests. This shuffled list of characters was giving us better results (See Section III).

Here is the shuffled character list -

```
'!', ';', 'k', 'a', 'i', 'n', '$', '>', 'q', '2', 'Z',
't', "'", '[', 'w', 'd', 'W', 'z', '*', '^', 'C', '5',
'T', 'F', '-', '{', 'I', 'g', 'Q', 'L', ':', '~', '0',
'8', 'l', '#', '=', 'o', 'b', 'j', 'r', '&', '@', 'u',
'3', 'X', 'x', ')', ']', 'A', 'e', 'U', 'D', ',', '`',
'G', '6', 'R', 'J', '/', '}', 'M', 'h', 'O', '"', '<',
'm', '1', '9', 'p', '%', '?', 's', 'c', 'Y', 'v',
'(', '\', 'y', '4', 'V', 'B', '+', '_', 'E', 'f', 'S',
'H', '.', '|', 'K', '7', 'P', 'N'
```

Figure 2: Shuffled characters to be used for encoding

The shuffled character list consists of all the alphabets small, along with capital. Apart from that, it also has all the digits and special characters. As far as the length of this list is concerned, it is 94, to be precise.

After the shuffled list is ready, we can proceed to the actual encoding. For deriving one character out of each chunk, the following algorithm will be used -

```
procedure Encode(chl, sl):
  gen_pass ← ""
  for each chunk in chl
    val ← 1
    for each char in chunk
      if char = '0'
        continue
      val ← val * hex2decimal(char) + val
    endfor
    gen_pass ← gen_pass + sl[val mod length(sl)]
  endfor
return gen_pass
```

For the given pseudo code, here is the flowchart -

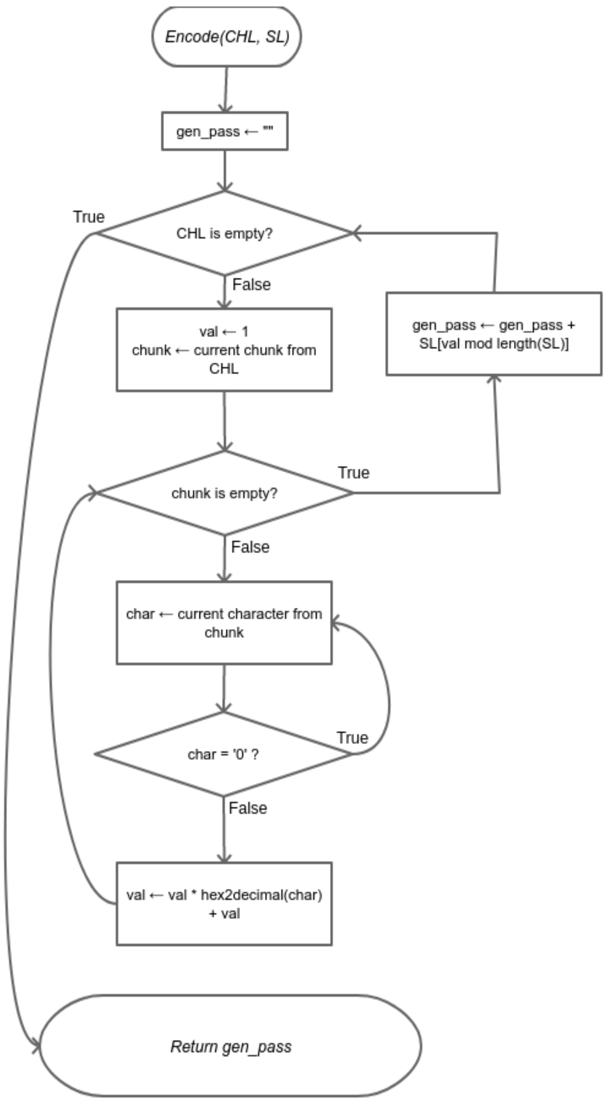
Encode(CHL, SL)
gen_pass ← ""
CHL is empty?
True
False
val ← 1
chunk ← current chunk from CHL
gen_pass ← gen_pass + SL[val mod length(SL)]
chunk is empty?
True
False
char ← current character from chunk
char = '0' ?
True
False
val ← val * hex2decimal(char) + val
Return gen_pass

Figure 3: Flowchart of the encoding function

Note:- Here CHL stands for Chunked Hash list and SL stands for Shuffled Letters.

III. RESULTS AND DISCUSSION

The proposed algorithm will generate passwords of different length according to the instruction given by the user. But, what about the strength of the password? A password is considered strong if it consists of small letters, capital letters, numbers, and special characters.

The generated password strength will be determined between 0 and 1. 0 being the lowest and 1 being the highest. The following flowchart will be used for determining the strength of the generated password.

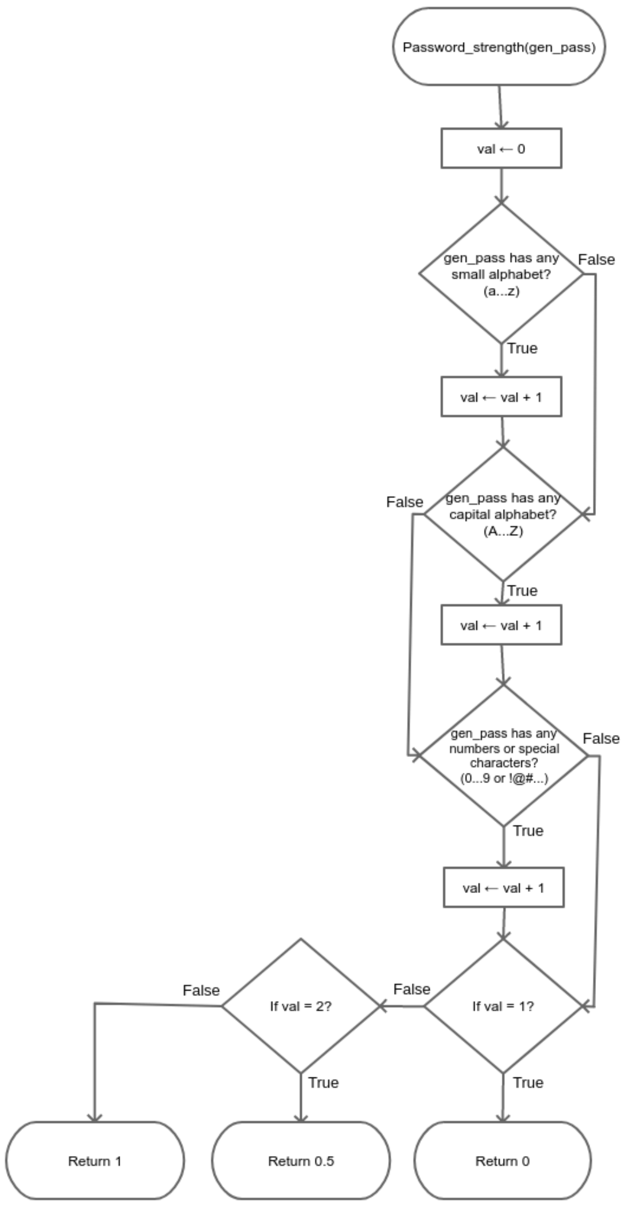
Password_strength(gen_pass)
val ← 0
gen_pass has any small alphabet? (a...z)
False
True
val ← val + 1
gen_pass has any capital alphabet? (A...Z)
False
True
val ← val + 1
gen_pass has any numbers or special characters? (0...9 or !@#...)
False
True
val ← val + 1
If val = 1?
False
True
If val = 2?
False
True
Return 1
Return 0.5
Return 0

Figure 4: Flowchart for getting the strength of a password

In Fig. 4, as shown, if the generated password has small letters, capital letters, and numbers, or special characters, then the strength is 1, i.e., very strong. If only one condition is fulfilled, then it will give 0 as a return value, i.e., very weak. If two of the conditions are satisfied, then 0.5 strength will be returned, which implies a moderate strength.

We have iterated over 100,000 most used passwords on the internet, converted into generated passwords, and checked their strength. The average of all the strengths of the generated password is filled in TABLE I.

TABLE I

PASSWORD STRENGTHS GENERATED BY THE ALGORITHM

Website/App Name	**Key Length**	**Strength**
facebook	6	84.53 %
instagram	6	84.28 %
facebook	8	92.52 %
instagram	8	92.51 %
facebook	10	96.35 %
instagram	10	96.33 %
facebook	12	98.10 %
instagram	12	98.19 %
facebook	16	99.54 %
instagram	16	99.53 %
facebook	20	99.88 %
instagram	20	99.87 %

(100k password list source - https://bit.ly/31H2uKl)

According to TABLE I, the strength largely depends on key length. Change in website/app name has a very insignificant effect. Also, even for a very small key length, the strength of the password is very good. The advisable key length is between 8 to 20. In that range of key length, the password's strength is more than 90 percent, which is a good sign.

III. LIMITATIONS

The whole system may seem very efficient, but there are few caveats. Some special case scenarios need to be addressed.

The first one is that if the generated password for any particular website is known to someone and we want to change it. We can't get a newly generated password individually. As it is known, the website/app name will be there for that specific website. If you do any change in the secret key then the generated password for other websites will also be affected. The other limitation of the system is that there is a very less probability of getting a weak password, but there is still a chance. If any particular website is insisting on a strong password, then, in that case scenario the user will be trapped. These are the limitations that we are aware of, but there may be more shortcomings of this algorithm, which is still unknown.

IV. CONCLUSION

We all know that passwords are not the best way of authenticating users. Replacement of passwords is important because people are careless while taking care of their login credentials. Still, we cannot deny that most of the websites are using passwords for authentication and will use it for a few more years. Password managers are paving their way in this very niche to solve this problem. Saving login credentials on the disk or in the cloud is problematic.

The solution to this whole system can be fixed with the use of password generators. According to user input, the hash value will be generated. For hashing, SHA-256 is used. It is a trapdoor function (one-way function), so deriving the actual string is quite infeasible. The hashed value contains the characters, which range between *0-9* and *a-f*. Due to this reason, directly using the hashed value as a password is not a good option. This hashed value is later encoded so that the character range can be exceeded.

Despite having few limitations discussed in Section - IV, it can still be used in many case scenarios. The areas in which it is lacking can be later fixed with further improvements.

V. REFERENCES

[1]. Bosnjak, Leon & Brumen, Bostjan. (2019). Rejecting the death of passwords: Advice for the future. Computer Science and Information Systems. 16. 313-332. 10.2298/CSIS180328016B.

[2]. McMillan, R. (2017, June 03). The World's First Computer Password? It Was Useless Too. Retrieved October 21, 2020, from https://www.wired.com/2012/01/computer-password/

[3]. Hunt, T. (2017, August 03). Passwords Evolved: Authentication Guidance for the Modern Era. Retrieved October 21, 2020, from https://www.troyhunt.com/passwords-evolved-authentication-guidance-for-the-modern-era/

[4]. M. Bachmann, "Passwords are Dead: Alternative Authentication Methods," 2014 IEEE Joint Intelligence and Security Informatics Conference, The Hague, 2014, pp. 322-322, doi: 10.1109/JISIC.2014.67.

[5]. Jose, J., Tomy, T. T., Karunakaran, V., Varkey, A., & Nisha, C. A. (2016, March). Securing passwords from dictionary attack with character-tree. In 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET) (pp. 2301-2307). IEEE.

[6]. Pieprzyk, J., & Sadeghiyan, B. (1993). Design of hashing algorithms. Springer-Verlag.

[7]. Gilbert, H., & Handschuh, H. (2003, August). Security analysis of SHA-256 and sisters. In International workshop on selected areas in cryptography (pp. 175-193). Springer, Berlin, Heidelberg.

[8]. Appel, A. W. (2015). Verification of a cryptographic primitive: SHA-256. ACM Transactions on Programming Languages and Systems (TOPLAS), 37(2), 1-31.

[9]. Yoshida, H., & Biryukov, A. (2005, August). Analysis of a SHA-256 variant. In International Workshop on Selected Areas in Cryptography (pp. 245-260). Springer, Berlin, Heidelberg.

Cite this article as :

Aditya Ray, Aman Roy, " Integrating Hashing with Encoding to Eliminate Password Managers, International Journal of Scientific Research in Science, Engineering and Technology(IJSRSET), Print ISSN : 2395-1990, Online ISSN : 2394-4099, Volume 7, Issue 5, pp.361-367, September-October-2020.

Journal URL : http://ijsrset.com/IJSRSET207562