(Approx. 3214 words)

# PASS(word) The Beef, the Hash, the Salt for Einstein, and a Dictionary

By Arthur Gresham, Editor, UCHUG Drive Light

Under the Computer Hood User Group

www.uchug.org

1editor101 (at) uchug.org

**Passwords and Hash, Part 1**

### The Meeting

A plate of food and a book

Description automatically generated with low confidence

At the September 2021 UCHUG General Meeting, the two primary presentation topics were closely related in the area of Security: SMISHING and 2FA. In addition, and coincidentally, October is National Security Awareness Month.

Discussion after the presentations moved into questions concerning the passwords we use on all the sites and services on the internet. Several questions and opinions were shared about managing our passwords, how to build passwords, security of passwords, and the impact of losses from the frequent Data Breaches from small and large businesses, government, and anyone operating a website using Password technologies having sensitive information stolen.

**The Beef**

Text, letter

Description automatically generatedA picture containing food, dish

Description automatically generatedSeveral issues were raised in that meeting questioning the validity of secure password methods, ease or difficulty of hacking into some system with or without an individual's true password, difficulty of 'Cracking' someone's password, or many passwords in a system database. We did not all agree and had some small but lively BEEF.

Obviously, there are many well-trained specialists in this area responsible for creating secure methods and protecting systems from the loss of your password. But many of the methods used are not easy to understand and involve sophisticated mathematical methods. As a result, there are a lot of myths and misconceptions about how passwords are stored by a corporation (or website) and how one could go about getting a password (and in particular, YOUR password) from that collection of thousands. For example, how can it be Cracked? Or Hacked? Or UnEncrypted?

### The Banker

But let's go back to basics in time a little bit. To the good old days when taking your money to the bank was a face-to-face activity. You walked up to the teller (remember when they had live people doing that?) and handed them your bank passbook.

You gave them your money.

They opened up a big ledger book, turned a few pages, and wrote your deposit into your account. How did they know what account to credit the money into? Answer: They looked at your account number in the passbook with your name on it.

How did they know it was really your account? Simple. You have a face, which they recognized (you both were members of the same Grange Hall, or perhaps they had voted for you as Mayor or shopped in your general store). Plus, you had 'the passbook.'

Your USER ID = Account # written in the passbook (physical possession)

Your Password (only you have it) = your face (a Biometric Password-aka Facial Recognition)

The Original Two Factor Authentication (2FA)!

**The Data**

A picture containing text

Description automatically generatedFast forward to today, and no one is storing your information in plain writing in a big ledger book. Instead, your valuable information, pictures, account balance, or credit card number are all stored as ones and zeros on a computer somewhere. Typically, most of this information is organized in huge databases or files. Some parts of it can be in plain language because it is simple information. However, the parts that give it security from theft should be protected somehow.

That is what happens with your password. When you create a password on a website, that password isn't stored verbatim on the website's server. That's because your password could be published and made freely available (Pawned (1) if the server's security were compromised. We call that a 'data breach.'

Instead, your password is put through a process called "hashing," which significantly improves security (provided your password is strong enough). In addition, the database record to access YOUR account will now have:

1. Your USER ID = this could be your email address or other name you use as the first entry in your login
2. Your HASHED Password = you must enter a password to verify that they match
3. Your name or other info, which may be encrypted, or plain text
4. Your account number or other internal ID of your account
5. Other data about your account, such as answers to your security questions, preference settings you have made, or any of the other many things about you that set up your use of that online space.

### 

**The Hash**

More than 50 Hash functions are MD5, SHA-1, SHA-2, SHA-256, NTLM, and LANMAN. (6)

A plate of food

Description automatically generated with medium confidence"*Hashes are the output of a hashing algorithm like MD5 (Message Digest 5) or SHA (Secure Hash Algorithm). These algorithms essentially aim to produce a unique, fixed-length string – the hash value, or "message digest" – for any given piece of data or 'message.'"*

Using a complex algorithm, hacking turns your password (or any other piece of data) into a short string of letters and numbers. (3) It is a short 'indicator' of the original text. [Note that they are not compression functions such as ZIP files that errorlessly retain all the original content. I will discuss this in detail in Part 2.]

If a website or corporation is hacked, the hackers don't get your password. Instead, they just get access to the database with the encrypted "hash" created by your password.

A common hash function is md5(), which returns a 32-character string from any input. Below are a few examples of what a hash looks like:

md5(*helloworld*) = fc5e038d38a57032085441e7fe7010b0

md5(*hell0world*) = 0a123b92f789055b946659e816834465

md5(*g84js;l238fl-242ldfsosd98234*) = 42e7862f4ad5225471866d2023fc4cca#

md5(*helloworld*) = fc5e038d38a57032085441e7fe7010b0

### The Recipe for Hash

From the examples above, notice these things are always true; they are in every recipe:

1. **Small changes matter a lot** – Take a look at examples 1 and 2. Just one digit has been altered, from an "o" to a "0." (OH to ZERO.) This is a very small change, and yet the second output is unrecognizable from the first.

Calendar

Description automatically generated

1. **The output length never changes** – The input in example 3 is considerably longer than the other examples, yet it produces an output of the same length (32 characters). You could input an entire book into the md5() hash function, and you would still get a 32-character string as the output.
2. **Repeatable** – An input will always give the same output when hashed using the same function. If this weren't the case, they would just generate a random output, which would be useless for passwords. (I included the same function in example 1 as example 4 just to see if you were paying attention.)

**Hard to reverse** – Even though a hacker may be able to tell the function used to create a hash, it's impossible to reverse that function and generate the password. In fact, it's so hard that trying millions of combinations to try and produce the same end result (a brute force attack) is typically quicker than the calculations required to reverse the hashing process. (The Humpty Dumpty Rule: You Can't Uncrack the scrambled egg in the HASH, more about that later)

### Einstein Expects Results

Text

Description automatically generated

As mentioned in item 3 above, we expect to get the same results for a given string every time. To get anything different would be crazy. That is what we count on for this concept to work, and we will also see later why it can be dangerous if you use a short password.

So, let's follow the steps in a normal log-in.

**Step 1** – A user visits a new site, fills in a form to create their username, uses a given default, and then creates a password.

**Step 2** – That password is put through a hash function, and the hash is stored in the company database.

**Step 3** – Later, when a user logs in, they enter their password.

**Step 4** – That entered password is run through the same hashing function as was used before.

**Step 5** – The server checks this hash against the one stored for the user in the database.

**Step 6** – If the two hashes match exactly, the user is granted access.

### The Dictionary for Uncracking the Egg

So, if no one can Unscramble the password, how are the criminals actually getting into an account after getting that Data Breach file from the Dark Web? The answer is they probably don't need to Unscramble it. They have a Dictionary. Or several. I am referring to what is properly called a **Hash**

**Table Dictionary** (also known as a [Rainbow Table](https://en.wikipedia.org/wiki/Rainbow_table). (4) I will simply call it - a dictionary.

A picture containing logo

Description automatically generatedWhat is a dictionary, and how does it help? Remember in 'The Hash' that the data for account 1 and account 4 had an identical Password and Hash? THAT is the weakness of a hash code. Anyone can run the hash function on as many words as they want and save the hash values in a database (this becomes their Dictionary). SO, they can save the hash for all the passwords like *'12345'* and 'admin*'* and any other word in a list of well-known, commonly re-used, and very bad passwords.

For example, the MD5 hash for *helloworld* is fc5e038d38a57032085441e7fe7010b0 And, that PW is now in their' dictionary.' When they look for 'fc5e….' in the stolen database, they find it, and it belongs to both user 1 and user 4. Both must have a password of *helloworld*. Almost zero seconds to look through the data and find all the fools who have used *helloworld* as their password. And they are not even breaking a sweat yet.

So, if you are a bad guy, what do you do? You would create huge lists using all the known passwords and their hash. Those lists of words and phrases contain things that have been used most often that will give them the biggest bang for their buck. And, with the hash for each of those passwords, all you need to do is look for them in the stolen database. Just one problem, and it is a big problem; there are a lot of words you must hash. That creates huge files. (You can check any password, plus variations, in a list of 14,344,391 known passwords (6). For example, Google for 'hello kitty' site:https://md5hashonline.com/most-common-passwords/ On page 310 it is word number 30,972 )

### Bigger is Better

And the longer the password lengths get, the huger (more bigger?) that file grows. So, for example, for passwords with just 9 lower case letters (*abcdefghi*), the number of passwords that could be formed is 5,646,683,826,134. But, of course, all those are not words, and as the number of characters (or numbers or symbols) increases, so does the size of the database they have to hash to complete their dictionary. So even if they had a database with all the possible combinations of 9 lower case and 3 uppercase letters, they would have almost 4 x 10**20**passwords. And with no symbols or numbers, it is not even close to being complete. And they would need to buy a lot of big drives and have lots of supercomputers working around the clock.

So, what do they do? They have reasonably sized (but huge) Hash Table Dictionaries, which they can afford to purchase, and have enough disk space to store, to get maybe just those top 5 (or 14 or 600) million common, repeated, very awful, known passwords.

But wait…. There's more. We have only done that for the MD5 function. They still need the time and disk space for the SHA-1, SHA-256, NTLM, and LANMAN. And what about words written in other languages? (*Holamundo* is *helloworld* in Spanish!) More possibilities. Without those, all of the data that was breached is of much less use to them. Unless they want to test words one hash, one at a time, that is called Brute Force. For the next 10 thousand years. (See footnote (7) about Hashcat). It is possible, but…..?)

### To Improve the Hash, Add Salt

A picture containing indoor

Description automatically generated

So, you see the problem here. Einstein told us. Do the same thing, get the same thing. It IS repeatable. Those repeated passwords all had the same repeated hash. How can that be fixed? It is neither impossible nor difficult. It can and should be fixed from two ends of the system.

If the hash is bad, we need to add Salt. But who should add the salt, You, or the Cook? It turns out the Cooks ought to season the hash, but in case they did not, then you should.

### Let the cook add the salt.

In this case, the 'cook' is the guy in the IT shop who wrote the routines that are hashing and saved the password you entered. Salting is adding something to the hash to make it different. For instance, adding the word Salt to *helloworld* and then hashing helloworldSalt OR *Salthelloworld* will generate new, unique hash values. This is good.

Here is how it works. (And I am going to shorten the hash just to make this readable)

If *helloworld* = fc5e0380 then *helloworldSalt* = er8d25a9

Now when they look for fc5e0380 (the word in their standard password list), they will not find it.

The bad guys will have to re-do their entire hash table dictionary if the cook adds the same salt to every word when they hash it. Thus, more time is added, delaying their access, and costing them money.

But the better site managers change the Salt shaker on every item. So the Salt can (must) be different (random) for every single entry in the database. This really disrupts the hacker's day because they must re-hash every standard password with every salt. That is effectively impossible.

Using our example, we could have three customers with the same password but now (salted with 'Salt,' '69b21' and 'pqv42') 1 helloworldSalt = er8d25a9

2 helloworld69b21 = a6d51cbc

3 helloworldpqv42 = f56702622

Now no matter what they have in their dictionary for *helloworld*, they can never find it in the target file.

For more about Salting, plus a very excellent description of the Dictionary process I have described, you should read (5) at

<https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>

**Salt it yourself**

A close-up of a chess board

Description automatically generated with low confidence

But wait, you say. I do not know if that site is using Salt (and no one should ever answer that question about their data). So, what can you, as the customer do? Bring your own Salt!

Salt all your passwords. Use whatever trick works for you to add your special something to EVERY password you create.

Also, if you use a short-length Master password for your Password Manager, Salt it too. (Or better yet, make it a long, easy phrase.) Simply add Salt when you type it in. Now, if anyone finds that sticky note that says your password is 'Arenteyespecial?' they will get nowhere without your special seasoning. (And no, don't write your salt down next to the beef.) And for all the passwords in your Password Manager, store them plain, and add the Salt during your login, and no one would ever know. If your plain character passwords are ever compromised, none of those passwords will work. Frustrating the bad guy, saving your bacon (and everything is better with bacon).

### SO, here are some lessons learned.

* Always Use a Password Manager program or app with a **long** master passphrase
* Create a long and seemingly **random** password for every site (easy to do with most Password Manager programs/apps)
* Change that password periodically
* **Never reuse** that password on other websites
* **Add Salt** (8 t0 12 characters is a good start)

And did I mention….You should always use a Password Manager ‘cuz your memory ain't that great.

In part 2, titled *'The Monkey & the Typewriter,'* I will teach you how those hash algorithms work, why no one can reverse (un-encrypt, decode, break, crack, hack - call it what you want) a hashed input. And I will even make you smart enough to create an 11-character hash when given a LONG input string. I promise you will never try to reverse a hash again. And I will show you more examples of how the bad guys do their thing to make you think they are 'cracking' your password.

Let me emphasize this about Password Managers. You should NEVER add your salt to the passwords you store i*n* your Password Manager. Just store your passwords as normal text. And when you enter it onto a site, then you add your salt. Then if anyone ever gets one or more, or all of your passwords, it will be of no use to them at all. Carry your own salt. Apply when needed.

### Some helpful sites-footnotes and additional resources

(1) Pawned Passwords are 613,584,246 real-world passwords previously exposed in data breaches. This exposure makes them unsuitable for ongoing use as they are at a much greater risk of being used to take over other accounts. Has YOUR password already been compromised? <https://haveibeenpwned.com/Passwords>

(2) What is Hashing (and how does it work?)

<https://www.sentinelone.com/cybersecurity-101/hashing/>

(3) Extensive quotes at the beginning of this article are from <https://thycotic.com/company/blog/2020/05/07/how-do-passwords-work/>

By [Barbara Hoffman](https://thycotic.com/company/blog/author/barbara-hoffman/) May 7, 2020

(4) Learn about the 7 Ways Hackers Steal Your Passwords. This article and Part 2 only cover methods 2 and 5, Spraying and Brute Force. YOU still must protect yourself against other types such as Phishing and Keyloggers, Local Discovery and of course Extortion

<https://www.sentinelone.com/blog/7-ways-hackers-steal-your-passwords/>

(5) Learn about adding SALT to HASHING from the perspective of those on the inside who create the systems to manage passwords. <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>

(6) A smaller list of 14,344,391 of the most common passwords discovered in various data breaches worldwide (plus some very odd strings!) at <https://md5hashonline.com/most-common-passwords> where you can see the results of the more than 50 hash functions, plus 115 MD5 variations on each. To search for a specific password or hash string, use a site-specific Google search such as this

['hello kitty' site:https://md5hashonline.com/most-common-passwords/](file:///C:\Users\scvju\AppData\Local\‘hello%20kitty’%20site:https:\md5hashonline.com\most-common-passwords\)

(7) Aren't there actual programs that try to 'crack' a single password? Yes, of course. A popular one is Hashcat. How does it work?

<https://www.csoonline.com/article/3542630/hashcat-explained-why-you-might-need-this-password-cracker.html>

**Additional Resources**

A quick evaluation of how secure your password is at

<https://howsecureismypassword.net/>

A couple easier to use websites that will make hash for you at

SHA-256 <https://www.freeformatter.com/sha256-generator.html#ad-output> (has a good tutorial)

MD5 and SHA-1 <https://www.md5hashgenerator.com/>

<https://md5hashonline.com/?s=nothing> Replace 'nothing' with something else

### Photo Credits

1. "HashandSaltandDictionary" by Arthur Gresham is licensed under CC BY-SA 2.0
2. "Corned Beef and Hash" by gozamos is licensed under CC BY-SA 2.0
3. "Freedmen's bank passbook" by Allen Gathman is licensed under CC BY-NC-SA
4. "Geordi & Data" by JD Hancock is licensed under CC BY 2.0
5. "Red Flannel Hash (9)" by Joelk75 is licensed under CC BY 2.0
6. "Betty Crocker's Cookbook" by Patrick Q is licensed under CC BY-NC 2.0
7. "Insanity by Albert Einstein" by Mimsen is licensed under CC BY-SA 2.0
8. "dictionary-1 copy.jpg" by TexasT's is licensed under CC BY-NC-ND 2.0
9. "salt shaker" by TooFarNorth is licensed under CC BY-NC-SA 2.0
10. "Day 76 of 365: kuchenne rewolucje" by Arek Olek (cropped) is licensed under CC BY 2.0

This work by Arthur Gresham is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/#_blank).

As long as you attribute this article, you can use it in part, or whole, for your newsletter, website, or blog.

All photos and images are licensed as noted in the Credits under a [Creative Commons Attribution 2.0 International License.](https://creativecommons.org/licenses/by/2.0/legalcode)

Creative Commons Attribution 2.0 International License. Creative Commons Attribution 2.0 International License.