Short Title: The Monkey, & the typewriter, More about hashing

Written by Arthur Gresham

CC by SA 4.0

6-14 Sep 2021

##

## **Part 2** of the password discussions

A plain language discussion about using the wrong terms when talking about passwords and hash coding

Terms and concepts include:

Data Breach, Hash Algorithms, Containers, ZIP and Unzipping, RAR, Make, Unmake, KEY, De-Coding, Encryption. Un-Encryption, Cracking, 'indicator' value, SALT, Beavis and Butt-head

This is a follow up to part 1 *PASS(word)* *The Beef, the Hash, the Salt for Einstein, and a Dictionary,* which, if you missed it is here:

[https://docs.google.com/document/d/1nhni0e-gjnb5yZwcmQv6fdINksdqqDJpFVBgHnknY3k/edit#](https://docs.google.com/document/d/1nhni0e-gjnb5yZwcmQv6fdINksdqqDJpFVBgHnknY3k/edit)

Art

(Approx. 2096 words)

**A Monkey and a Typewriter Make a Hash with Shakespeare and a Soccer Ball**

By Arthur Gresham, Editor, UCHUG Drive Light

Under the Computer Hood User Group

www.uchug.org

1editor101 (at) uchug.org

**Passwords and Hash, Part 2**



This discussion is a follow-up to Part 1 ***PASS(word) The Beef, the Hash, the Salt for Einstein, and a Dictionary***, in which I introduced the process of hashing passwords and the concept of Salt.

During a continuing discussion with a friend, while writing part 1, I finally realized that we were looking at the same things and coming to different conclusions regarding passwords. For example, we debated whether passwords stored as a hash code are really easy to un-encrypt (decode/crack/break/hack) or really hard.

##

## **We Both Win**

It turns out we are all using the wrong terms. Yes, the hash code for a short password is of little value because it can be determined quickly. He wins. But it is also a fact that a hash code cannot be un-encrypted. I win. I will demonstrate both of these concepts in this article.

The big problem is because several terms are being incorrectly used for the world of hashing and passwords. Let me explain by using very simple examples from our shared experience.

##

## **The Gorilla in the Room**

****You may recall a theoretical discussion when you were in school. Something about a monkey in a room with a typewriter being able to write the works of Shakespeare if he has enough time to randomly peck the keys. This thought experiment is called the **Infinite monkey theorem** (read about it [in Wikipedia](https://en.wikipedia.org/wiki/Infinite_monkey_theorem) (1) if you have forgotten how it works).

The strings produced by Hash Algorithms look like something you might think was written by that monkey. We expect that most of what that monkey typed is gibberish. Likewise, the hash for a particular input text (or a picture or an entire operating system, library, or simple password) is an' indicator'. This text appears to be pure gibberish. That is because it does not contain anything actually from the input. The key here is 'contain.'

**The Key is the Container**

****Let me illustrate that in a different way. You are all familiar with ZIP or RAR, or other compression functions. You have undoubtedly downloaded some program, text, spreadsheet, or audio file, which was sent to you as a compressed file. When you get the file, you 'unzip' it into a folder, then read, watch, listen to, or somehow use the contents inside that file. The zip was much smaller than the original contents inside of it. Yet, it contains an EXACT duplicate of the original inputs. If it didn't, you would be very upset. Your program would not run, or your audio would not play, or the words in the text would turneenloiariuwwka08qkj k3lksd fjasdhd rhandnt making you very confused and unhappy. This is a two-way process, In and out.

Hash Algorithms are not a zip of the original input. While the Zip file was easy to unzip because it is designed as a two-way process, the Hash is a one-way process. You can MAKE a hash, but you can't UN-MAKE it. It does not 'contain' any information about the input string; it cannot be cracked. Again, this is a one-way process. What goes in can not come out.

As an extreme example, this week, I installed a new version of Linux on the computer I am typing on right now. The download was a 2 Gigabyte file. Part of the install instructions are to compare the SHA-256 Hash (2) of this download with a given 256-byte check value. The SHA-256 Hash from the authorized site must match your value to ensure that yours is a complete, unaltered download. But the SHA-256 Hash does not contain all of Linux Mint 20.2 Cinnamon and all its files. If it did, I could have just downloaded the Hash, UN-MAKE it, and installed it. So, the files aren't contained in the Hash.

Yet if I create the SHA Hash for the string 'A' (that is just the letter A), I will still get a 256-byte hash value. And it certainly does not compare to the contents of my Linux download.

This is because a Hash is only what is called an 'indicator' value.

##

## **The Container has a Key**

Let me give you another example and use a couple of other terms that have been misused in this discussion. Perhaps you remember WW 2 (No, I am not trying to age check you, so put your hearing aids back in your ears and listen up). During the war, the radio became a vital tool for communications. The allies used it to communicate from London to the generals in the field. But they did not simply use plain words to give instructions. Instead, those instructions were processed with machines that scrambled the letters. The messages were 'encrypted.' Headquarters used a KEYCODE to garble the text. That text was sent by radio, and anyone with a receiver could get it. But only our side (mostly) had the matching KEYCODE to UN-Encrypt the message.

With our fast, modern-day computers, perhaps we could now DECODE or CRACK those messages (simple cipher codes), but they did not have the means to do it then, so the messages were secure.

But here again, the messages were designed as Two-way messages, containing the plain text going in and coming out with the same exact text when un-encrypted. If it wasn't exact, it would have been of no use in the war effort.

With Hash Algorithms, there is no Container. There is no KEY. No Unzipping. No Coding-No De-Coding, No Encryption-No Un-Encryption. Because a Hash is only an 'indicator' value.

**Time to make Hash**

Time for some fun. I want to program your brain. I want you to be my Hash Function Computer. You will have only one job. That is to give me an answer to the question I will ask. Trust me, you have the brainpower to do this.

Here is your input text:

*I am larger than a softball, smaller than a basketball, I am covered with black and white pentagon shapes, and if you kick me into the net, you will score one point What Am I?*

Hint: don't Google it. You will not find the answer...

Just think...

Don't peak…



Time is Money

Got your answer?

**Did you make hash?**

If you said "*soccer ball,*" you are right. Those 11 characters are the hash of that input string. I told you this was easy. BUT if I had said to you "*soccer ball* "at the beginning of this article or in a conversation, what is the chance you would have responded with the exact text -

"*I am larger than a softball, smaller than a basketball I am covered with black and white pentagon shapes, and if you kick me into the net, you will score one point What Am I?*"

But wait. There are many Hash Algorithms and what you just gave me was the American-11 algorithm. What would you have said if you lived in London?

?

Sure, I hope you understand in that part of the world, they have said *FOOTBALL* .

Because that, you see, is the British-8 algorithm. Not to be confused with the Spanish-6 algorithm, which would have said *FúTBOL* . Different Algorithms might produce different lengths. Yet, they are all only 'indicators' of the same exact input. But they do not un-anything any of them. The hash does not contain the input string. So it can't be Cracked.

**A Hard Nut to Crack?**

In the paragraph titled We Both Win, I said, **"**the hash code for a short password is of little value because it can be determined quite easily**."**

While a short input text of a hash code may be determined quite easily, note that I did not say it could be Un-Encrypted or Cracked. For this demonstration, I will be using the Art-4 algorithm. Thus, any input string will generate a 4-character hash (cuz my brain is very small).

You will be playing the part of the internet's bad guys. First, I will show you five input strings (a dictionary of passwords) that have been hashed with the Art4() algorithm. This will represent the bad guys' precomputed Hash Table Dictionary (see part 1 for a description of this).

**Input String ART4() hash**

1. AAAAAAAA = aee9
2. longword = 9546
3. Password = dc647
4. Password99 = e6ab
5. Willam1 = b4b9

{Note all of these passwords have been [Pawned](https://haveibeenpwned.com/Passwords) (3). Someone has actually used them!}

Imagine a bank had a Data Breach (someone inside opened an email and clicked on a "Link." You know the rest of the story!) The bank had saved customers' passwords using my algorithm. Their records are in the database, which was stolen from a bank.

I want YOU to see if you can 'Crack' any of the bank's data and tell me whose password you 'cracked.'

Here is a bit of the data breach file:

**Username Password hash Balance**

Joe 3255 $10,100

Mary 7bb4 $101,000

Beavis 9546 $52.14

Bill 5835 $250,000

(It takes time, but the lookup yields results, look carefully)

So, did you 'Crack' any of the hash values in the bank database? Could you try to log in with the password of any of these victims?

It looks like user Bevis may lose his savings. Maybe his *longword* just was not long enough. But did you actually 'Decrypt,'' Decode,' 'Crack,' 'Hack,' or 'reverse engineer' any of the passwords?

NO. You simply found a value that matches a known hash (you found it in your Hash Table Dictionary), and you 'Guessed' what one of the passwords might be. And you would be exactly correct because, as we learned from Einstein in part 1: "we expect to get the same results for a given string every time. To get anything different would be crazy."

By the way, customer Bill, whose root password is William1, will not be in trouble because he salted his password. So, unless you bad guys hash his actual password ("*William1"*) plus his salt (which is "*PlusPepper*") to get an ART4() hash of 5835, you will not be getting into his account.

And because the bank did not SALT the customer's passwords, a plain language hash dictionary leaves many customers vulnerable for this look-up solution.

Do not let that be you. Use Good Passwords, not common short words or expressions that will be found in the dictionary. And when you do enter or change your password, use SALT if it's valuable, SALT it.



"Don't be a Beavis. Use strong passwords."

 "Yeah, Dude… and Salt it."

## Photo Credits

1. "Picture on Early Office Museum" Author: New York Zoological Society, 1907 Public Domain
2. "Infinite Monkeys 2008" by Simon Greig Photo is licensed under CC BY-NC-SA 2
3. "Glass food storage container with Easy Find Lids" by Rubbermaid Products is licensed under CC BY 2.0
4. "U-505 Enigma Machine (View 4)" by derekbruff is licensed under CC BY-NC 2.0
5. "hourglass" by secubie is licensed under CC BY-NC-SA 2.0
6. "Walnut in nutcracker" by wuestenigel is licensed under CC BY 2.0
7. "Infinite Monkeys 2008" by Simon Greig Photo is licensed under CC BY-NC-SA
8. "Beavis and Butt-head titlecard" fair use by https://en.wikipedia.org/wiki/File:Beavis\_and\_Butt-head\_titlecard.png#filelinks

[CC](https://creativecommons.org/licenses/by/2.0/?ref=ccsearch&atype=rich) creative commons licensed

## References

1. Infinite monkey theorem [Wikipedia](https://en.wikipedia.org/wiki/Infinite_monkey_theorem) article https://en.wikipedia.org/wiki/Infinite\_monkey\_theorem
2. SHA-256 <https://www.freeformatter.com/sha256-generator.html#ad-output> has a good tutorial
3. Pawned Passwords are a dictionary of 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>

*A Monkey and a Typewriter Make a Hash with Shakespeare and a Soccer Ball* by Arthur Gresham is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-sa/4.0/).

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