Cryptographic Hash Functions

Grant Haataja

April 30, 2019

Contents

Introduction

Hash functions are an important facet of cybersecurity. Although the name may sound dry and complicated to those unfamiliar with hash functions, there are many interesting properties and uses for hashes, and they affect everyone who uses the internet.

This paper will discuss cryptographic hash functions and many of the intriguing aspects of their nature. Two famous and widely-used hash functions will be analyzed in detail, namely MD5 and SHA-256.

A hash function is an algorithm that takes an input of arbitrary length and maps it through a series of transformations to a string of fixed length. The purpose of this is to verify a given input without needing to store that input. The fixed length property is useful for storage and consistency, as well as file size reduction. It is important to understand what hash functions are and how they are used in everyday security so one can make good decisions about their personal digital protection.

Requirements of Hash Functions

There are several criteria that must be met for a hash function to be useful [1]. They must be computationally efficient, which means the algorithm should run quickly on a computer without requiring a lot of memory or processing power.

They must have strong pre-image resistance, which means that given the output of a hash function, there should be no way to figure out the input that generated that particular output. Also, hash functions have to be deterministic, meaning anytime a certain input is put into a hash function, the output should always be the same. In order to be secure, hash functions must have strong collision resistance, which means there should be no reasonable way to find multiple inputs that generate the same output. And finally, the output of a hash function should have the appearance of complete randomness.

How do Hash Functions Work?

Hash functions take an input of any size and go through a series of steps to change it into something theoretically unrecognizable to the original message. The output of a hash function is commonly referred to as a *digest*.

A hash function can be as simple as

$$f(x) = x \mod 7.$$

Then any input could be converted into a decimal number and sent through the function. For example, consider the input: "Message". If we apply the standard conversion from letters to numbers, (A $\rightarrow$ 1,B $\rightarrow$ 2,...,Z $\rightarrow$ 26), "Message" becomes 13 5 18 18 1 7 5. Then, we take it as the number 1,351,818,175 and send it through the function, giving

$$f(1351818175) = 1351818175 \mod 7 = 1.$$

Now, there are only 7 possible outputs for this hash function, making it totally useless for verification, since there are so many collisions. It would take fractions of a second to find a collision for this algorithm with a

computer. But it does illustrate 2 important characteristics of hash functions.

First, no matter what the size of the input is, the output will always be of fixed length, in this case one digit. Second, if one was given the output of the above message, (1), and told to reverse it to find the original input, there is almost no chance that anyone could do that.

Uses of Cryptographic Hash Functions

There are many important uses of cryptographic hash functions, including file verification, password protection, and bitcoin mining and creation of addresses.

When a user goes to download a file from the internet, if the website is responsible and secure, there will be a hash of the file stored along with it, and before the file begins downloading, the content is hashed again to verify that it matches the key. This is to ensure that an attacker has not modified or replaced the file with malicious content.

Password protection is one of the most critical uses of cryptographic hash functions. Consider a website that stores users' usernames and passwords in plain text. If a hacker gained access to the database, they would own the credentials to all users of that service. Since many people use the same password for all accounts, the attacker could pivot those credentials for such users to gain access to their emails, credit cards, and online banking accounts. If instead a website stores the hashed version of users' passwords, an attacker could not reverse the hashes to discover the actual passwords of the users, and thus the hashes would be nearly useless to them.

There is one significant issue with this approach, however. Many users use simple passwords, and thus it is not uncommon for many users to have the same, simple password. If, for example, 20 users in a database have the password "password123", the deterministic nature of a hash function would generate the same digest for each of those 20 users. If an attacker noticed 20 of the same hash digest, it wouldn't take long to run a password list through hash function generators online to find which password those 20 users had, thereby owning their credentials.

To get around this security issue, responsible websites use a technique called "salting". Using hash and salt, when a user creates an account and types a password in for the first time, the website uses a hashing algorithm, but also adds a random string of characters to the user-entered string, thus ensuring that even if hundreds of users enter the same exact password, each of their hashes will be different. The salt is usually stored right next to the salted hash in the database. Then every time the user enters their password to log in, the salt is added to whatever they entered and that result is hashed. If it matches the stored digest, they successfully login to their account. This way, if a hacker gains access to a database, every hashed password will look different, and the only way they can match a password to a specific hash is by running a brute force attack.

MD5 (Message Digest Algorithm 5)

One of the previously most widely-used hashing algorithms is the Message Digest Algorithm 5, (MD5). This 128-bit hash was created by Ronald Rivest in 1991 as a replacement for the MD4 algorithm which was found to be insecure shortly after Rivest published it in 1990.

MD5 was designed as a cryptographic hash function, but has been found to contain extensive security vulnerabilities. Collision resistance was broken in 2004 after 2^{21} hashes, with the attack only taking one hour to complete.

MD5 is still used extensively, but should no longer be implemented for any sensitive data verification, especially not for passwords or secure keys. [4]. Many common passwords can be found by typing their hash digests into Google search.

SHA-256 (Secure Hashing Algorithm)

The current most commonly-used cryptographic hashing algorithm is the Secure Hashing Algorithm 2. There are several different algorithms in this family, but the most common one is the 256-bit SHA-256. Developed in 2001 by the NSA as part of the SHA-2 set, this is the most trusted and widely used hash algorithm in use currently. SHA-3 is the newest Secure Hashing Algorithm, and is slowly taking over and replacing SHA-256, but that will take time. [5].

Collision resistance has not been fully broken on SHA-256 yet, the best public attacks currently break 46 out of 64 rounds. For preimage resistance, 52 out of 64 rounds break it. SHA-256 can also be vulnerable to length extension attacks, meaning the security of the hash algorithm is decreasing over time. For optimal security, it is recommended to switch to the 512-bit SHA-3 algorithm.

Hashing Examples

Consider the message "Hashing is a secure way to store passwords, change my mind". Below are shown the MD5 and SHA-256 hashes of this message.

MD5: b30d5d9bc11f392daa4950be13d35106

SHA-256: a1b196e3e571170f5111c9f0058d20cc4761546468fbe6bdc16f2663f6096e7a

Now, the comma is removed from the message and the following is hashed: "Hashing is a secure way to store passwords change my mind". Observe how the hash digests have completely changed, all from the removal of a single comma.

MD5: 91a6526d84520f2aad7f25116880d4e9

SHA-256: 7809dace6f2004d72611a4a59f2d6fa5d2ab0307c2b7de741b86779275681a5b

This is the power of hash functions. This also illustrates how using a salt for users' passwords on a website protects the randomness of the hashes even if multiple users have the same password, since appending even a single character to the user-entered string changes the output of the hash function completely.

Another word on collisions. Any well-developed hashing algorithm should have high collision resistance, but it is important to note that it is theoretically impossible to make a hash function that is completely collision-free. This is due to the fact that a hash function returns a fixed-length string. So, for a 128-bit hash, the output is always 32 characters. Possible characters include the 26 letters and 0-9, so a total of 36 possible characters. This gives 36^{32} possible hashes, which is 6.334×10^{49}. This is an astronomically high number, so breaking a hash of this size with brute force is infeasible, but given that there is a near-infinite number of inputs, there will always be possible collisions. [2].

Conclusion

Hashing should be used over encryption in any case where the verifying party does not need to know the actual data. Because hashing always yields a fixed-length string, it is portable and easy to store digests. Hashing is one of the most secure ways to store sensitive data for verification purposes, as long as the hashes are salted. Unlike encryption, no key is required for verification, since hashing algorithms are not secret due to their one-way nature.

There are a few downfalls of hash functions, however. Hashing cannot be used to send messages or any data that the receiver needs to reverse back to its original form. Unsalted hashes stored in a database can become a security risk if an attacker gains access and finds multiple occurrences of a specific hash digest.

Overall, hash functions are an efficient and elegant way to store data and verify files and passwords. If hashing can be used instead of encryption, it will likely be a more secure option, assuming proper techniques are followed.

References

[1] Daniel. (2018, December 03). Cryptographic Hash Functions Explained: A Beginner's Guide. Retrieved from https://komodoplatform.com/cryptographic-hash-function/

[2] Tutorialspoint.com. (2019). Cryptography Hash functions. Retrieved from https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm

[3] Lyons, J. (2012). MD5 Hash. Retrieved from http://practicalcryptography.com/hashes/md5-hash/

[4] Holbreich, A. (2016, November 01). Cryptographic Hash Functions. Retrieved from http://alexander.holbreich.org/cryptographic-hash-functions/

[5] Mikoss, I. (2019). The In's and Outs of Cryptographic Hash Functions (Blockgeek's Guide). Retrieved from https://blockgeeks.com/guides/cryptographic-hash-functions/