Cryptographic Hash Functions

Grant Haataja

March 21, 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. There will be two famous and widely-used hash functions that 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.

Requirements of Hash Functions

There are several criteria that must be met for a hash function to be useful. 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

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

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. 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. Many common passwords can be found by typing their hash digests into Google search.

SHA-256 (Secure Hashing Algorithm)

256-bit hash Designed by the NSA as part of the SHA-2 set in 2001 Collision resistance hasn't been fully broken yet Most commonly used and trusted hashing algorithm currently, although industry is in the process of switching to SHA-3

Hashing Examples

- We have the message "Hashing is a secure way to store passwords, change my mind".
- MD5: b30d5d9bc11f392daa4950be13d35106
- SHA-256: a1b196e3e571170f5111c9f0058d20cc4761546468fbe6bdc16f2663f6096e7a
- Now we will remove the comma from the message and hash "Hashing is a secure way to store passwords change my mind".
- MD5: 91a6526d84520f2aad7f25116880d4e9
- SHA-256 7809dace6f2004d72611a4a59f2d6fa5d2ab0307c2b7de741b86779275681a5b

Pros and Cons of Hashing

- PROS
  - Portable and easy for storing
  - Simple and fairly secure way to store sensitive data to be used for verification, as long as the hashes are salted
  - Does not require any key for verification
- CONS
  - Cannot be used to send messages, since there is no way to reverse secure hashes
  - Unsalted hashes can become security risks

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/