The evolution of hash functions

Hash functions are important controls that are part of every company's security strategy. Hashing is widely used for authentication and **non-repudiation**, the concept that the authenticity of information can't be denied.

Previously, you learned that **hash functions** are algorithms that produce a code that can't be decrypted. Hash functions convert information into a unique value that can then be used to determine its integrity. In this reading, you'll learn about the origins of hash functions and how they've changed over time.

Origins of hashing

Hash functions have been around since the early days of computing. They were originally created as a way to quickly search for data. Since the beginning, these algorithms have been designed to represent data of any size as small, fixed-size values, or digests. Using a hash table, which is a data structure that's used to store and reference hash values, these small values became a more secure and efficient way for computers to reference data.

One of the earliest hash functions is Message Digest 5, more commonly known as MD5. Professor Ronald Rivest of the Massachusetts Institute of Technology (MIT) developed MD5 in the early 1990s as a way to verify that a file sent over a network matched its source file.

Whether it's used to convert a single email or the source code of an application, MD5 works by converting data into a 128-bit value. You might recall that a **bit** is the smallest unit of data measurement on a computer. Bits can either be a 0 or 1. In a computer, bits represent user input in a way that computers can interpret. In a hash table, this appears as a string of 32 characters. Altering anything in the source file generates an entirely new hash value.

Generally, the longer the hash value, the more secure it is. It wasn't long after MD5's creation that security practitioners discovered 128-bit digests resulted in a major vulnerability.

Here is an example of how plaintext gets turned into hash values:

Hash collisions

One of the flaws in MD5 happens to be a characteristic of all hash functions. Hash algorithms map any input, regardless of its length, into a fixed-size value of letters and numbers. What's the problem with that? Although there are an infinite amount of possible inputs, there's only a finite set of available outputs!

MD5 values are limited to 32 characters in length. Due to the limited output size, the algorithm is considered to be vulnerable to **hash collision**, an instance when different inputs produce the same hash

value. Because hashes are used for authentication, a hash collision is similar to copying someone's identity. Attackers can carry out collision attacks to fraudulently impersonate authentic data.

Next-generation hashing

To avoid the risk of hash collisions, functions that generated longer values were needed. MD5's shortcomings gave way to a new group of functions known as the Secure Hashing Algorithms, or SHAs.

The National Institute of Standards and Technology (NIST) approves each of these algorithms. Numbers besides each SHA function indicate the size of its hash value in bits. Except for SHA-1, which produces a 160-bit digest, these algorithms are considered to be collision-resistant. However, that doesn't make them invulnerable to other exploits.

Five functions make up the SHA family of algorithms:

- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512

Secure password storage

Passwords are typically stored in a database where they are mapped to a username. The server receives a request for authentication that contains the credentials supplied by the user. It then looks up the username in the database and compares it with the password that was provided and verifies that it matches before granting them access.

This is a safe system unless an attacker gains access to the user database. If passwords are stored in plaintext, then an attacker can steal that information and use it to access company resources. Hashing adds an additional layer of security. Because hash values can't be reversed, an attacker would not be able to steal someone's login credentials if they managed to gain access to the database.

Rainbow tables

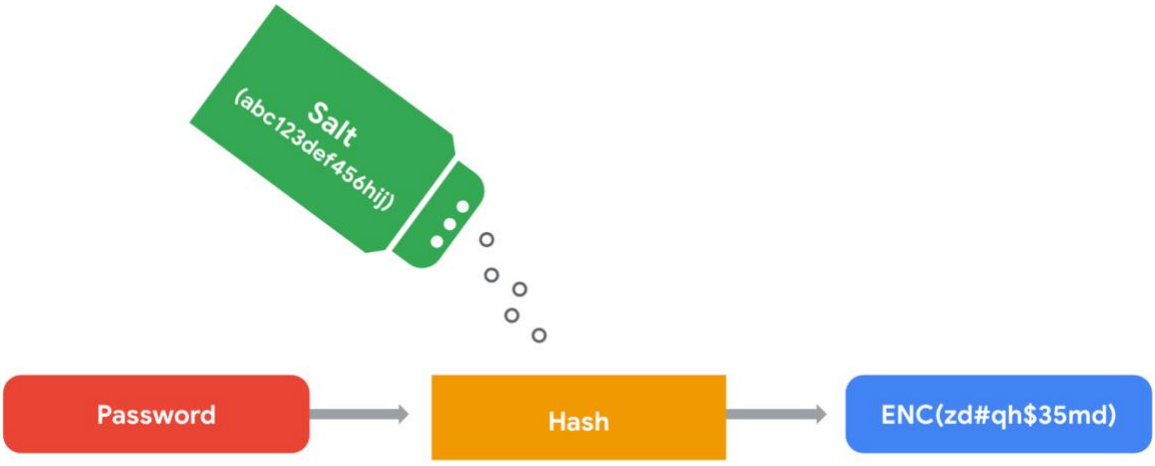
A **rainbow table** is a file of pre-generated hash values and their associated plaintext. They're like dictionaries of weak passwords. Attackers capable of obtaining an organization's password database can use a rainbow table to compare them against all possible values.

Adding some "salt"

Functions with larger digests are less vulnerable to collision and rainbow table attacks. But as you're learning, no security control is perfect.

Salting is an additional safeguard that's used to strengthen hash functions. A *salt* is a random string of characters that's added to data before it's hashed. The additional characters produce a more unique hash value, making salted data resilient to rainbow table attacks.

For example, a database containing passwords might have several hashed entries for the password "password." If those passwords were all salted, each entry would be completely different. That means an attacker using a rainbow table would be unable to find matching values for "password" in the database.



For this reason, salting has become increasingly common when storing passwords and other types of sensitive data. The length and uniqueness of a salt is important. Similar to hash values, the longer and more complex a salt is, the harder it is to crack.

Key takeaways

Security professionals often use hashing as a tool to validate the integrity of program files, documents, and other types of data. Another way it's used is to reduce the chances of a data breach. As you've learned, not all hashing functions provide the same level of protection. Rainbow table attacks are more likely to work against algorithms that generate shorter keys, like MD5. Many small- and medium-sized businesses still rely on MD5 to secure sensitive data. Knowing about alternative algorithms and salting better prepares you to make impactful security recommendations.