A Cryptographic Introduction to Hashing and Hash Collisions

Last Updated : 10 Sep, 2021

What is hashing?

Hashing is the process of converting any kind of data (usually passwords or installer files) into a fixed-length string. There are multiple types of hashes, but for this article, we will look only at the MD5 hash. MD5 is an example of a hashing method. For example, the MD5 hash of “hello” (without the quotes) is “5d41402abc4b2a76b9719d911017c592” (without the quotes). Similarly, the MD5 hash of “Geeks for Geeks” (without the quotes) is “5ee878924e0cb782e0729066a7d88832” (without the quotes). You’ll notice that even though “Geeks for Geeks” is longer than “hello,” their MD5 hashes are *both the same length*. This is what makes hashes vulnerable, and what we will discuss later in this article.

Why do we use hashes?

Hashes tend to have a few general use cases –

1. Password security
2. Ensuring correct downloads
3. Image integrity verification

In this article, we will be looking at hashing through the lens of password security, as this is its most common use case.

Password Security :

Many times, a website needs to have a login form to authenticate its users. Usually, these websites have a lot of different users and need to store their login information in some kind of database. However, storing passwords as plaintext in a database is incredibly insecure.

For example, let's say a hacker managed to get into the following database of login information stored on a website –

Username	Password
admin	Admin_securep@$$w0rd
johnnyappleseed123	ja.5923!
zadiben	nov151982

From this table, it becomes plainly obvious what each user's password is. Additionally, the hacker just gained access to the administrator account and can control several aspects of the website now! Instead, the website decides to store passwords using their MD5 hashes. Exploring this scenario, let's say the hacker manages to get into the website's new database. This is what they see –

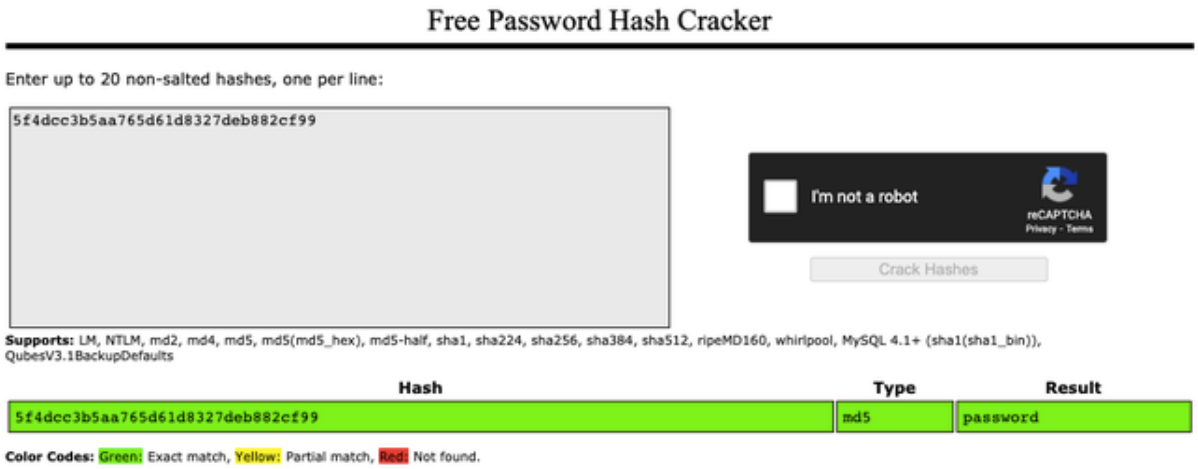
Username	Password Hash
admin	865c5895f347413ca07c81e6c365cb31
johnnyappleseed123	a55805ec1caef94681bb07271659c887
zadiben	ae93717757ecfb103847b6752b88ed36

Now, all the hacker can see is password's hash and *theoretically* cannot log in to any of these accounts. Without the actual password, the hash is (again, theoretically) useless to the hacker. But now, how will website be able to authenticate its users? Instead of checking the user's entered password against the database, the website will check the hash of the password the user entered. If that hash matches hash in database, user is authenticated! This is because the MD5 hash of "Admin_securep@$$w0rd" will always be "865c5895f347413ca07c81e6c365cb31" (more secure hashes like BCrypt use more complex hashing methods as well as more complex methods of comparing two hashes but, for simplicity's sake, we will stick with MD5 for now) and, though you can always go from the password to hash, it is (theoretically) incredibly difficult to go from the hash back to password.
The word "theoretically" has been thrown around a lot just now, and here's why.

Hash cracking :
Hash cracking entails taking a large wordlist or dictionary and hashing each word. Then, you check the hash of each word in the dictionary against the hash you are trying to crack. Once you have found a match, you have found your word! This is why it is not recommended to use common words as your password.
Referring to the example above, the hacker gained access to the website's login database and this was one of the lines in the database –

Username	Password Hash
pharrell157	5f4dcc3b5aa765d61d8327deb882cf99

Because the hacker is not giving up just because the passwords are hashed, they will run this hash through a hash cracker (which check against very commonly-used hashes) like this one. It's a simple matter of typing in hash "5f4dcc3b5aa765d61d8327deb882cf99," solving the reCaptcha, and getting a result –



crackstation.net cracking a common MD5 hash

Now, the hacker knows pharrell157's password is simply just "password" in a couple of seconds using a free online tool.
This is one of the main reasons an MD5 hash is not secure. Because it is an unsalted hash (unlike BCrypt), the same hash results from the same data every time. That is, the MD5 hash of "password" will always be "5f4dcc3b5aa765d61d8327deb882cf99.
In addition to just hash cracking (which is something to which every hash is vulnerable), MD5 is incredibly insecure for another, larger reason.

Hash collisions :

There are infinitely many possible combinations of any number of bits in the world. Therefore, there are infinitely many possible data that can be hashed. Note the definition of a hash above which states that a hash is always fixed-length. For example, the MD5 hash is always 128 bits long

(commonly represented as 16 hexadecimal bytes). Thus, there are 2^128 possible MD5 hashes. While this is an extremely large number, it is certainly finite... though the number of possible passwords that can be hashed is infinite. What this means is that infinitely many different passwords have the same hash. This also means that if a hacker gains access to the MD5 hashes of passwords, they do not necessarily need to find the actual password, but something else which shares that hash. Because of recent innovations in technology, finding collisions in MD5 hashes is all but trivial. For more, see Marc Stevens's project, HashClash, or Corkami's GitHub repo on collisions.

What can we use instead of MD5?

Thankfully, now that MD5 does not provide the same level of security we might hope for, many new hashes have been created which can. For example, the SHA-256 (a.k.a. SHA-2) is more secure because it is 256 bits long instead of 128. Now, most websites use salted hashes like BCrypt which can create different hashes from the same password as long as there is a varying salt (or seed). MD5 hashes may still be used in digital forensics in some cases but it has mostly been succeeded by these more secure alternatives.