Benjamin Dahmen-Hwang
Simon Rubenstein-Salzedo
Euler Circle 2024
19 August 2024
Paper on Cryptographic Hash Functions

# 1 Introduction

Cryptographic hash functions are a way of condensing a large body of data into a much shorter string. The output of a cryptographic hash function is usually a base 256 string whose characters appear to have little to no resemblance to the original string. There is usually not a one-to-one correspondence between the original data and the hash function's output, so the hash function cannot be used to reproduce the original string; however, a good cryptographic hash function will make it challenging for one to find two different strings that correspond to the same hash.

# 2 Cryptographic hash function properties:

A good cryptographic hash function will have the following properties:
The cryptographic hash function should produce fairly short output, regardless of the length of the input string. Depending on the application, this could be anywhere between 8 characters for emails and passwords to several hundred characters for longer bodies of text. It generally should have a fixed length less than a certain value, and it should not take up much space when stored. This makes it hard to find the length of the original string based on the hash function, and it also allows the cryptographic hash to be stored without taking up much data.
It should be hard to find two different strings that produce the same hash. This means that it should be challenging to find two strings such that $h(s_1) = h(s_2)$. In some cases, this would make it significantly more difficult for Eve to find a message she could send without Bob being able to detect it came from Eve.
One small change in the string should produce a significant change in the cryptographic hash. When hashes are being used for data integrity and security purposes, if the cryptographic hash function does not significantly

change when the string slightly changes, a data change between transmission and reception (either by Eve or because of signal interference) may go undetected. Furthermore, if the hash function is being used to represent a password, someone could enter in a wrong password that contains reasonable similarity with the correct one and still be granted login access.
The cryptographic hash function should not change in predictable ways. It should not preserve any arithmetic operation, whether addition or multiplication (for instance, $h(2e)$ should not be equal to $2h(e)$). If the cryptographic hash function did change predictably, the cryptographic hash function would become insecure due to chosen ciphertext attacks, which would give a hacker a much better chance of being able to correlate a password with the given hash value.
Given the hash value, it is very hard to gather any useful information about the characters in the input string. If this was not the case and the cryptographic hash represented a password, information could be gained about the password just by looking at the hash, and this is often undesirable. Furthermore, if Eve could somehow gather information about the hash value of a message, she could get clues about the original message, which would not be good. Even though this is likely to be the case if the first four requirements were met, it is still important to keep in mind when designing cryptographic hash functions.
Lastly, the algorithm should be fairly efficient. Hashes will often have to be computed hundreds to thousands of times in a program, and an inefficient hash function could slow a program down.

# 3 Utilizing hash functions for data integrity and authentication.

As mentioned earlier, cryptographic hash functions have major applications in data integrity, safely storing passwords, and even signing messages. Since one small change in the message should result in a different cryptographic hash function, to verify that data was transmitted correctly, one could send both the cryptographic hash value and the actual message over the channel; this way, if the message was altered in the process for any reason, the received message's hash value would not be equal to the sent hash value. One poten-

tial issue with this is that if Eve could change the data being sent before it reached its destination, she could edit the message and change the cryptographic hash value to match the new message. However, there is a relatively simple solution to this problem; use a cryptographic hash function that is constantly changing with every message sent, and then use a Diffie-Hellman key exchange to establish the numbers used to generate the cryptographic hash function.
First, Alice and Bob should establish a number that is to be used to generate their cryptographic hash function. One way of doing this is a Diffie-Hellman key exchange; Alice's and Bob's private key can be random numbers, and the key that they establish would become the number used to generate the cryptographic hash function. It is very common in practice to encode cryptographic hash functions with a key. Now, when Alice wants to send a message to Bob, she first computes $h(m, k)$, where $k$ is the key, and m is the message. Then, she sends this information publicly. Eve can change this value, but because she does not know what $k$ is, she cannot compute the cryptographic hash value of the message she intends to send. Thus, when Alice sends the message, Bob can tell if Eve manipulated it, as the cryptographic hash value sent through the channel would not match up with the cryptographic hash value of the encrypted message. Since Eve cannot compute the cryptographic hash value of the message she changed, the best thing she could do would be to randomly change the cryptographic hash value that Alice sent and hope it matches up; however, if the cryptographic hash value is sufficiently large, this is impractical. Even if Eve knew what the message was, she still could not find $k$, because finding $k$ given $m$ and $h(m, k)$ should be just as hard as finding $m$ given $h(m, k)$ and k. This is an example of how cryptographic hash functions (combined with the Diffie-Hellman key exchange) can be useful to authenticate data in a more efficient way than the encrypt-then-sign method.

# 4 Methods of hash function computation

Most of the time, a cryptographic hash function should have as little symmetry as possible, while still being quick to compute. A good way to do this is by using a key in combinations with polynomials. First, write the cryptographic hash input value in base 256 for ANSI text (this is equivalent to converting it back into its text state). Then, compute the following:

$h(m,k) = (\sum_{i=0}^{n} m_i k^i)(mod\ p)$
In this formula, $m_i$ is the value of the ith digit, m is the message, $k$ is the key, $n$ is the length of the base 256 string of $m$, and p is the modulus. The value of $p$ should be chosen based on the desired output length of the hash function; for some length $l$, because of the fact that a character of text can be represented in 8 bits, the maximum value of $p$ such that the key length is less than or equal to $l$ is $2^{8l}$.
There exists other methods for computing hash functions, such as MD5 or SHA-256, which operate using several rounds of data splitting (breaking the string into pieces), doing several bitwise, shifting and substitution procedures, and repeating the process many times; however the method shown above is by far the simplest. It is challenging to find a cryptographic procedure that satisfies all the requirements for cryptographic hash functions, and even algorithms like MD5 that were once used are now insecure. This is an interesting problem in cryptography, one that is very important for data security.

# 5 Using hash functions for email and password hiding

Cryptographic hash functions are also useful for storing passwords in a secure manner. Let's consider that you want to store your password, but you don't want anyone on the server side to know what your password is. One solution to this problem is to store the password as a cryptographic hash value instead. If the hash function has all of the previously discussed properties, it would be very hard for an attacker to gain any information about the user's password given the cryptographic hash value; thus, a hacker could view the hash values of a password and still not be able to gain much information about the password itself. When a user logs in, the email and password should be POSTed together in an encrypted form to the server. It would not be a good idea to send the cryptographic hash value of each string to the server instead, since if this was done, the person that had access to the cryptographic hash values in the database could send those values to the server instead, bypassing the need to know the email and password to log in. Once the server receives the email and password, the server should decrypt

them and convert each string to a cryptographic hash value. Next, it should look for matches of those values with values in the database. If the email hash value matches up with an email hash value in one of the rows of the database, then the program should check to see if the password hashes match up. If the passwords do not match up, then the server should send a message to the client indicating that his/her password was incorrect. Similarly, if the entire list is scanned and no email matches are found, a message should be sent indicating that the account does not exist.

# 6 Mistaken identity

There is one potential caveat with this method, however. Let's consider that a server stores 50 million emails as hash values. Most likely, each hash value will be distinct; however, there is a chance that two emails could correspond to the same hash value, since a cryptographic hash function is not bijective. If a user tries to create an account, and the email's hash matches an email hash value in the database (despite the original emails not being the same!) the user may not be able to create an account since the server mistakes the emails as being the same. This can cause a user to not be able to create an account, and even if they were allowed to create an account, an ambiguity would be created in the server's database, possibly leading to errors and even allowing a user to log into the wrong account.
There are a couple of ways to reduce the probability of this issue. Firstly, one could store the first few characters of every email before the cryptographic hash value; this would allow the server to verify, with increased reliability, whether an email has already been taken, and it would also reduce the odds of any ambiguities. The first few characters of an email will not be that helpful to anyone trying to hack into the system, as they would still have insufficient information to find the actual email used to log in. Storing the first few characters of an email could also be useful for forms, in situations where the user has to select an email to use; showing only the first few characters allows verification that they are submitting the form under the desired account, without revealing too much valuable information to a hacker. This is much more convenient than having the user manually type out their email every time they want to submit a form, since typos and having another field to fill out can slow down a user significantly.

Another possible method to reduce the probability of mistaken identity would be to increase the length of the output of the hash function; this would increase the number of possible outputs of the hash function, decreasing the probability of an error due to loss of information. Even if the cryptographic hash function's output is only 8 characters long, the probability of such an event randomly occuring is approximately is extremely rare, with a 1 in a quadrillion probability; for this reason, it is usually not a major concern for programmers, and thanks to the hard-to-find-matches property of cryptographic hash functions, it would be hard for a hacker to lock a user out by registering an account whose email hashed to the same value as the user's email, due to the difficulty of finding such an email. Furthermore, a valid email has a specific pattern to it, one that can be verified to be satisfied quickly with regular expressions; a cryptographic hash function's pair solution would most likely not be of any use, since it most likely would not have the format of an email address.

Not only can two emails hash to the same value, but two passwords can hash to the same value as well (in fact, for every password, there are infinite hash matches). At first glance, this may appear to be a major problem, because having multiple passwords for your account should make it easier for a hacker to find the right combination and be granted access. However, the passwords that hash to the same value as your password will have a lot less patterns, will be of the same length, if not longer, and will likely contain characters (such as control characters) that a website could easily search for and invalidate requests made with such characters. Similar to email addresses, despite seeming to be problematic at first glance, cryptographic hash functions work well with passwords, and their loss of information is not much of a problem at all.

# 7 How you can see use of hash functions as you go about your daily life

Due to the irretrivability of an email from its hash, some forms will ask for information that you wouldn't think you'd need to give. For example, in a password reset form, you will often be asked to enter in your email. One may think that this was a security feature, but they actually have to do this, be-

cause the company doesn't actually know what your email is! Instead, they need the hash value to correlate the email with a password, and the only way to figure out which hash value corresponds to your email is for you to give them your email information. This is an example of how you can see the use of cryptographic hash functions without directly interacting with any code!

# 8 Conclusion

Overall, cryptographic hash functions are very useful in cryptography for a variety of reasons. They can be used to verify a message transmitted correctly, and with the help of a Diffie-Hellman key exchange, they can also be used for data authentication. To prevent emails and passwords from being visible to people with access to a company database, cryptographic hash functions can be used to translate emails and passwords into a form that cannot be easily traced back to the original text, but it still can be used to verify whether a given email or password is equivalent to a value in the database (for purposes of logging in and verifying whether a person has already signed up). The five properties that allow cryptographic hash functions to function so well in the aforementioned applications are its short length, the challenge of finding multiple strings corresponding to the same hash, its ability to make one small change in the text noticeable, its asymmetry, and its lack of direct correlation with the input string. Generally, since high-order polynomials modulo n fit these properties quite well, it is not uncommon for cryptographic hash functions to utilize polynomials and keys for their translation of data. With their relative simplicity and wide variety of applications, cryptographic hash functions have massive importance and usefulness in cryptography.