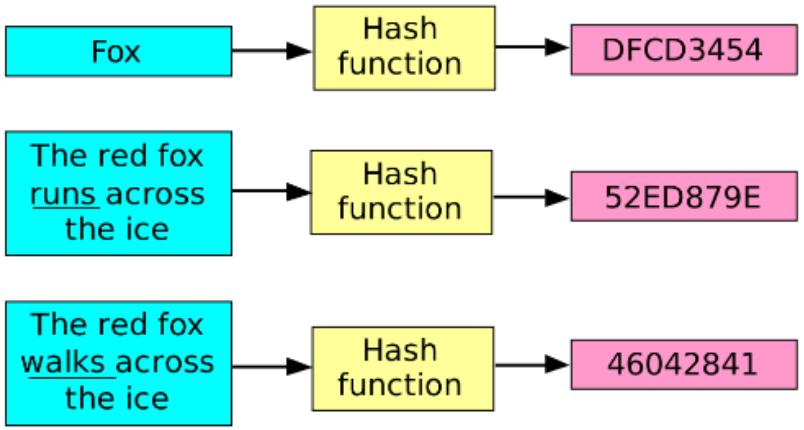
Secure Hash Algorithms

Secure Hash Algorithms, also known as SHA, are a family of cryptographic functions designed to keep data secured. It works by transforming the data using a hash function: an algorithm that consists of bitwise operations, modular additions, and compression functions. The hash function then produces a fixed-size string that looks nothing like the original. These algorithms are designed to be one-way functions, meaning that once they're transformed into their respective hash values, it's virtually impossible to transform them back into the original data. A few algorithms of interest are SHA-1, SHA-2, and SHA-3, each of which was successively designed with increasingly stronger encryption in response to hacker attacks. SHA-0, for instance, is now obsolete due to the widely exposed vulnerabilities.

A common application of SHA is to encrypting passwords, as the server side only needs to keep track of a specific user's hash value, rather than the actual password. This is helpful in case an attacker hacks the database, as they will only find the hashed functions and not the actual passwords, so if they were to input the hashed value as a password, the hash function will convert it into another string and subsequently deny access. Additionally, SHAs exhibit the avalanche effect, where the modification of very few letters being encrypted causes a big change in output; or conversely, drastically different strings produce similar hash values. This effect causes hash values to not give any information regarding the input string, such as its original length. In addition, SHAs are also used to detect the tampering of data by attackers, where if a text file is slightly changed and barely noticeable, the modified file's hash value will be different than the original file's hash value, and the tampering will be rather noticeable.



A small tweak in the original data produces a drastically different encrypted output. This is called the avalanche effect [1] .

Contents

SHA Characteristics

Cryptographic hash functions are utilized in order to keep data secured by providing three fundamental safety characteristics: pre-image resistance, second pre-image resistance, and collision resistance.

The cornerstone of cryptographic security lies in the provision of **pre-image resistance**, which makes it hard and time-consuming for an attacker to find an original message, m, given the respective hash value, h_m. This security is provided by the nature of one-way functions, which is a key component of SHA. Pre-image resistance is necessary to ward off brute force attacks from powerful machines.

EXAMPLE

One-way Function

Alice and Bob are pen pals who share their thoughts via mail. When Alice visited Bob, she gave him a phone book of her city. In order to keep their messages safe from intruders, Alice tells Bob that she will encrypt the message. She tells Bob that he will find a bunch of numbers on every letter, and each sequence of numbers represents a phone number. Bob's job is to find the phone number in the book and write down the first letter of the person's last name. With this function, Bob is to decrypt the entire message.

To decrypt the message, Bob has to read the entire phone book to find all the numbers on the letter, whereas Alice can quickly find the letters and their respective phone numbers in order to encrypt her message. For this reason, before Bob is able to decrypt the message by hand, Alice can re-hash the message and keep the data secure. This makes Alice's algorithm a one-way function[2].

The second safety characteristic is called **second pre-image resistance**, granted by SHA when a message is known, m_1, yet it's hard to find another message, m_2, that hashes to the same value: $H_{m_1} = H_{m_2}$. Without this characteristic, two different passwords would yield the same hash value, deeming the original password unnecessary in order to access secured data.

The last safety characteristic is collision resistance, which is provided by algorithms that make it extremely hard for an attacker to find two completely different messages that hash to the same hash value: $H_{m_1} = H_{m_2}$. In order to provide this characteristic, there must be a similar number of possible inputs to possible outputs, as more inputs than outputs, by the pigeonhole principle, will definitively incur potential collisions. For this reason, collision resistance is necessary, as it implies that finding two inputs that hash to the same hash value is extremely difficult. Without collision resistance, digital signatures can be compromised as finding two messages that produce the same hash value may make users believe two documents were signed by two different people when one person was able to produce a different document with the same hash value.

Recent cryptographic functions have stronger security characteristics to block off recently developed techniques such as length extension attacks, where given a hash value, $hash(m)$, and the length of the original message, m, an attacker can find a message, m', and calculate the hash value of the concatenation of the original message and the new message: $hash\ (m||m')$.

As a general guideline, a hash function should be as seemingly random as possible while still being deterministic and fast to compute.

SHA-1

Secure Hash Algorithm 1, or SHA-1, was developed in 1993 by the U.S. government's standards agency National Institute of Standards and Technology (NIST). It is widely used in security applications and protocols, including TLS, SSL, PGP, SSH, IPsec, and S/MIME.

SHA-1 works by feeding a message as a bit string of length less than 2^{64} bits, and producing a 160-bit hash value known as a *message digest*. Note that the message below is represented in hexadecimal notation for compactness.

There are two methods to encrypt messages using SHA-1. Although one of the methods saves the processing of sixty-four 32-bit words, it is more complex and time-consuming to execute, so the simple method is shown in the example below. At the end of the execution, the algorithm outputs blocks of 16 words, where each word is made up of 16 bits, for a total of 256 bits.

Pseudocode

Suppose the message 'abc' were to be encoded using SHA-1, with the message 'abc' in binary being

$$01100001\ 01100010\ 01100011$$

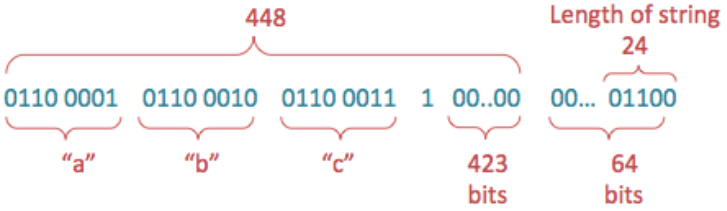
and that in hex being

$$616263.$$

1) The first step is to initialize five random strings of hex characters that will serve as part of the hash function (shown in hex):

$$H_0 = 67DE2A01$$
$$H_1 = BB03E28C$$
$$H_2 = 011EF1DC$$
$$H_3 = 9293E9E2$$
$$H_4 = CDEF23A9.$$

2) The message is then padded by appending a 1, followed by enough 0s until the message is 448 bits. The length of the message represented by 64 bits is then added to the end, producing a message that is 512 bits long:



Padding of string "abc" in bits, finalized by the length of the string, which is 24 bits.

3) The padded input obtained above, M, is then divided into 512-bit chunks, and each chunk is further divided into sixteen 32-bit words, $W_0 \ldots W_{15}$. In the case of 'abc', there's only one chunk, as the message is less than 512-bits total.

4) For each chunk, begin the 80 iterations, i, necessary for hashing (80 is the determined number for SHA-1), and execute the following steps on each chunk, M_n :

- For iterations 16 through 79, where $16 \leq i \leq 79$, perform the following operation:

$$W(i) = S^1\big(W(i-3) \oplus W(i-8) \oplus W(i-14) \oplus W(i-16)\big),$$

where XOR, or $\oplus$, is represented by the following comparison of inputs x and y :

x	y	Output
0	0	0

1	0	1
0	1	1
1	1	0

- For example, when i is 16, the words chosen are $W(13), W(8), W(2), W(0)$, and the output is a new word, $W(16)$, so performing the XOR, or $\oplus$, operation on those words will give this:

$W(0)$	$01100001\ 01100010\ 01100011\ 10000000$
$W(2)$	$00000000\ 00000000\ 00000000\ 00000000$
$W(8)$	$00000000\ 00000000\ 00000000\ 00000000$
$W(13)$	$00000000\ 00000000\ 00000000\ 00000000$
	$\oplus$
$W(16)$	$01100001\ 01100010\ 01100011\ 10000000$

DEFINITION

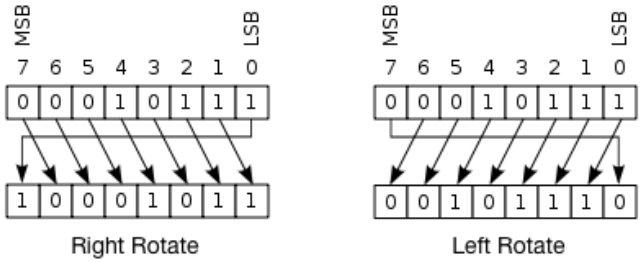
Circular Shift Operation

Now, the circular shift operation $S^n(X)$ on the word X by n bits, n being an integer between 0 and 32, is defined by

$$S^n(X) = (X << n) \quad \textbf{OR} \quad (X >> 32 - n),$$

where $X << n$ is the **left-shift** operation, obtained by discarding the leftmost n bits of X and padding the result with n zeroes on the right.

$X >> 32 - n$ is the **right-shift** operation obtained by discarding the rightmost n bits of X and padding the result with n zeroes on the left. Thus $S^n(X)$ is equivalent to a circular shift of X by n positions, and in this case the circular left-shift is used. [3]



So, a left shift $S^n\big(W(i)\big)$, where $W(i)$ is 10010, would produce 01001, as the rightmost bit 0 is shifted to the left side of the string. Therefore, $W(16)$ would end up being

$$11000010\ 11000100\ 11000111\ 000000000.$$

5) Now, store the hash values defined in step 1 in the following variables:

$$\begin{aligned} A &= H_0 \\ B &= H_1 \\ C &= H_2 \\ D &= H_3 \\ E &= H_4. \end{aligned}$$

6) For 80 iterations, where $0 \leq i \leq 79$, compute

$$TEMP = S^5 * (A) + f(i; B, C, D) + E + W(i) + K(i).$$

See below for details on the logical function, f, and on the values of $K(i)$. Reassign the following variables:

$$\begin{aligned} E &= D \\ D &= C \\ C &= S^{30}(B) \\ B &= A \\ A &= TEMP. \end{aligned}$$

7) Store the result of the chunk's hash to the overall hash value of all chunks, as shown below, and proceed to execute the next chunk:

$$\begin{aligned} H_0 &= H_0 + A \\ H_1 &= H_1 + B \\ H_2 &= H_2 + C \\ H_3 &= H_3 + D \\ H_4 &= H_4 + E. \end{aligned}$$

8) As a final step, when all the chunks have been processed, the message digest is represented as the 160-bit string comprised of the **OR** logical operator, $\vee$, of the 5 hashed values:

$$HH = S^{128}(H_0) \ \vee \ S^{96}(H_1) \ \vee \ S^{64}(H_2) \ \vee \ S^{32}(H_3) \ \vee \ H_4.$$

So, the string 'abc' becomes represented by a hash value akin to **a9993e364706816aba3e25717850c26c9cd0d89d**.

If the string changed to 'abcd', for instance, the hashed value would be drastically different so attackers cannot tell that it is similar to the original message. The hash value for 'abcd' is **81fe8bfe87576c3ecb22426f8e57847382917acf**.

DEFINITION

Functions used in the algorithm

A sequence of logical functions are used in SHA-1, depending on the value of i, where $0 \leq i \leq 79$, and on three 32-bit words B, C, and D, in order to produce a 32-bit output. The following equations describe the logical functions, where $\neg$ is the logical *NOT*, $\vee$ is the logical *OR*, $\wedge$ is the logical *AND*, and $\oplus$ is the logical XOR:

$$\begin{aligned}
f(i;B,C,D) &= (B \wedge C) \vee \left((\neg B) \wedge D\right) && \text{for } 0 \geq i \geq 19 \\
f(i;B,C,D) &= B \oplus C \oplus D && \text{for } 20 \geq i \geq 39 \\
f(i;B,C,D) &= (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) && \text{for } 40 \geq i \geq 59 \\
f(i;B,C,D) &= B \oplus C \oplus D && \text{for } 60 \geq i \geq 79.
\end{aligned}$$

Additionally, a sequence of constant words, shown in hex below, is used in the formulas:

$$\begin{aligned}
K(i) &= 5A827999, && \text{where } 0 \leq i \leq 19 \\
K(i) &= 6ED9EBA1, && \text{where } 20 \leq i \leq 39 \\
K(i) &= 8F1BBCDC, && \text{where } 40 \leq i \leq 59 \\
K(i) &= CA62C1D6, && \text{where } 60 \leq i \leq 79.
\end{aligned}$$

Albeit SHA-1 is still widely used, cryptanalysts in 2005 were able to find vulnerabilities on this algorithm that detrimentally compromised its security. These vulnerabilities came in the form of an algorithm that speedily finds collisions with different inputs, meaning that two distinct inputs map to the same digest [4].

As of 2010, many organizations have recommended its replacement by SHA-2 or SHA-3. Companies like Microsoft, Google, or Mozilla have announced that their browsers will stop accepting SHA-1 encryption certificates by 2017 [5].

SHA-2

Due to the exposed vulnerabilities of SHA-1, cryptographers modified the algorithm to produce SHA-2, which consists of not one but two hash functions known as SHA-256 and SHA-512, using 32- and 64-bit words, respectively. There are additional truncated versions of these hash functions, known as SHA-224, SHA-384, SHA-512/224, and SHA-512/256, which can be used for either part of the algorithm.

SHA-1 and SHA-2 differ in several ways; mainly, SHA-2 produces 224- or 256-sized digests, whereas SHA-1 produces a 160-bit digest; SHA-2 can also have block sizes that contain 1024 bits, or 512 bits, like SHA-1.

Brute force attacks on SHA-2 are not as effective as they are against SHA-1. A brute force search for finding a message that corresponds to a given digest of length L using brute force would require 2^L evaluations, which makes SHA-2 a lot safer against these kinds of attacks.

Common Attacks

Cryptography wouldn't be as quickly developed if it weren't for the attacks that compromise their effectiveness. One of the most common attacks is known as the primeage attack, where pre-computed tables of solutions are used in a brute-force manner in order to crack passwords. The solution against these kinds of attacks is to compose a hash function that would take an attacker an exorbitant amount of resources, such as millions of dollars or decades of work, to find a message corresponding to a given hash value.

Most attacks penetrating SHA-1 are collision attacks, where a non-sensical message produces the same hash value as the original message. Generally, this takes time proportional to $2^{n/2}$ to complete, where n is the length of the message. This is the reason the message digests have increased in length from 160-bit digests in SHA-1 to 224- or 256-bit digests in SHA-2 [6].

Other attacks exist that attempt to exploit mathematical properties in order to crack hash functions. Amongst these is the birthday attack, where higher likelihood of collisions are found when using random attacks with a fixed number of letter combinations (see the pigeonhole principle), or the rainbow table attack, where a pre-computed hash table is used to reverse a hash function in order to crack passwords.

References

1. media, w. *Hash Function*. Retrieved June 25, 2016, from https://upload.wikimedia.org/wikipedia/commons/thumb/d/da/Hash_function.svg/2000px-Hash_function.svg.png
2. Khovanova, T. *One-way Functions*. Retrieved June 27th, 2016, from http://blog.tanyakhovanova.com/2010/11/one-way-functions/
3. Systems, C. *US Secure Hash Algorithm 1 (SHA1)*. Retrieved September 2001, from https://www.ipa.go.jp/security/rfc/RFC3174EN.html
4. Schneier, B. *Cryptanalysis of SHA-1*. Retrieved June 25th, 2016, from https://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html
5. Forum, G. *Intent to Deprecate: SHA-1 Certificates*. Retrieved June 25th, 2016, from https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/2-R4XziFc7A%5B101-125%5D
6. Morton, B. *Why we Need to Move to SHA-2*. Retrieved January 2014, from https://casecurity.org/2014/01/30/why-we-need-to-move-to-sha-2/

Cite as: Secure Hash Algorithms. *Brilliant.org*. Retrieved 13:19, June 26, 2021, from https://brilliant.org/wiki/secure-hashing-algorithms/