International Journal of Humanities Social Science and Management (IJHSSM)
Volume 3, Issue 4, Jul.-Aug., 2023, pp: 385-389 www.ijhssm.org

Hash Function on Cryptography

Suparnesh Bhattacharyya
[1,2]*Faculty Member, Gobardanga Hindu college, Gobardanga, Westbengal, India*

Date of Submission: 24-07-2023 Date of Acceptance: 06-08-2023

Abstract:
Modern cryptography relies heavily on hash functions because they offer efficient and safe ways to transform input data of any size into fixed-size hash values. The fundamental traits and uses of hash functions in cryptographic systems are examined in this abstract. It is very impossible to recover the original data from its hash value since a hash function is a one-way function that creates a unique and irreversible output for each input. Pre-image resistance, second pre-image resistance, and collision resistance are three crucial characteristics of secure hash functions that guarantee the safety of cryptographic systems. They are used in many different applications, including password storing, data integrity checking, and digital signatures. In order to protect confidential information and maintain the security of cryptographic systems in the face of developing computing capabilities and potential assaults, this abstract emphasizes the significance of selecting robust and thoroughly tested hash functions.

Body:

A hash function is a key component of cryptography that transforms input data of any size into a fixed-size output known as a hash value or hash code. A one-way function, hashing is created so that it is computationally impossible to reverse it and retrieve the original input from its hash result. Numerous cryptographic applications, such as digital signatures, message authentication codes (MACs), password storage, and data integrity checking, make extensive use of hash functions.

What Makes a Secure Hash Function?

1. Deterministic: The hash function consistently generates the same output for a given input.

2. Quick Computing: The hash function should compute the hash value for any input quickly and efficiently.

3. Pre-image Resistance: Determining the original input from its hash value should be computationally impossible.

4. Second Pre-image Resistance: It should be computationally impossible to locate an alternative input that yields the same hash result given a certain hash value.

5. Collision Resistance: Finding two different inputs that give the same hash value should be computationally impossible.

6. Avalanche Effect: A slight modification to the input ought to result in a noticeably changed hash value.

Hash Function Examples:

1. SHA-256 (Secure Hash Algorithm 256-bit): This member of the SHA-2 family creates hash values that are 256 bits (32 bytes) long.

1.1. Secure Hash algorithm

The National Institute of Standards and Technology (NIST) in the United States has released a family of cryptographic hash functions under the name "Secure Hash Algorithm" (SHA), which was created by the NSA. The SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 hash function families all include members. Each participant produces hash values of various lengths.

Secure hash algorithms are made to take arbitrary-length input messages and return a fixed-size result, called the hash value or hash code. A secure hash algorithm's essential characteristics include:
Pre-image Resistance: It should be computationally impossible to locate the original input message given a hash value.
Second Pre-image Resistance: Given an input message, finding an alternative input that yields the same hash result should be computationally impossible.
Collision Resistance: Finding two different input messages that create the same hash value should be computationally impossible.
Due to weaknesses discovered in SHA-1, which was once widely utilized, its use has been discontinued in favor of its descendants in the SHA-2 family, particularly SHA-256 and SHA-512. The most recent member of the SHA family, SHA-3, offers an alternate design to SHA-2 and is also regarded as secure.
In the SHA-2 family, SHA-256 produces a hash value that is 256 bits (32 bytes), while SHA-512 produces a hash value that is 512 bits (64 bytes). Numerous cryptographic applications, such as digital signatures, data integrity checks, password storage, and blockchain technology, frequently employ these hash functions.

It's crucial to remember that the security of hash functions may deteriorate over time as cryptographic technologies progress and computing power rises. Utilizing the most recent and secure hash functions is therefore crucial, and security professionals routinely assess and suggest modifications to cryptographic standards to address new threats and weaknesses.

2. The newest member of the SHA family, SHA-3 (Secure Hash Algorithm 3), is intended to offer improved security and performance over SHA-2.

3. MD5 (Message Digest Algorithm 5): An older hashing algorithm that is currently regarded as cryptographically flawed because of flaws.3

3.1. The Message Digest Algorithm 5 (MD5) is a cryptographic hash function that generates a 128-bit hash value from an input message of any length. An explanation of the MD5 algorithm and an illustration of how it produces a hash value for a sample input message are provided below.

Overview of the MD5 algorithm:

To make sure that the message can be broken up into 512-bit blocks, the input message is padded to a length that is a multiple of 512 bits (64 bytes).

Initializing Variables: The initial hash values are four 32-bit words (A, B, C, and D).

Processing Blocks: The padded message is split into 512-bit blocks, and each block is processed via a number of rounds. Each cycle includes a succession of left rotations, modular additions, and bitwise logical operations (AND, OR, XOR).

Final Hash Value: After processing every block, a 128-bit (16-byte) hash value is produced by concatenating the four little-endian, 32-bit words (A, B, C, and D).

MD5 Illustration

Let's compute the MD5 hash for the message "Hello, MD5" in the input.

"Hello, MD5!" is the input message.

The message is padded in the manner shown below:

I'm here, MD5. "Hello, MD5!1" (padding with a 1 bit) was the initial message."48656C6C6F2C204D44352180 00000000" (padded with zeros to a multiple of 512 bits) is written as "1x80" (The padded message is represented in hexadecimal form)

A = 0x67452301 was used to initialize the variables.

D = 0x10325476, B = 0xEFCDAB89, C = 0x98BADCFE, and

Blocks of Processing: The padded message is split into 512-bit blocks and processed in this manner. I'll cut out the specific stages to keep this short.

The final hash value, which is a representation of the 128-bit hash value in hexadecimal form, is "4b5d0953fa38f7c37aade8b9c5a0e6b9" after all blocks have been processed.

Thus, the input message "Hello, MD5!" has an MD5 hash of "4b5d0953fa38f7c37aade8b9c5a0e6b9".

Keep in mind that due to flaws, MD5 is no longer regarded as secure. For applications that require security, it is advised to utilize more reliable hash algorithms like SHA-256 or SHA-3.

4. RACE Integrity Primitives Evaluation Message Digest 160 (RIPEMD-160): This message digest is frequently used in crypto currencies like Bit coin.

4.1. A 160-bit (20-byte) hash value is generated by the cryptographic hash function known as RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest 160). It was created as a component of the RACE Integrity Primitives Evaluation (RIPE) project and is frequently employed in many different applications, particularly those involving crypto currencies like Bit coin.

Overview of the RIPEMD-160 algorithm:

The input message is padded to a multiple of 512 bits (64 bytes), just like other hash algorithms, to make sure it can be separated into 512-bit blocks.

Initializing Variables: The initial hash values for RIPEMD-160 are five 32-bit words (A, B, C, D, and E).

Processing Blocks: The padded message is split into 512-bit blocks, and each block is processed via a number of rounds. These rounds consist of left rotations, modular additions, and bitwise logical operations.

Final Hash Value: Following the completion of all blocks, a 160-bit (20-byte) hash value is created by concatenating the five little-endian, 32-bit words (A, B, C, D, and E).

RIPEMD-160 Illustration

Let's determine the RIPEMD-160 hash for the message "Hello, RIPEMD-160!" that was entered.

"Hello, RIPEMD-160!" is the input message.

The following is the padding for the message: "Hello, RIPEMD-160!" "Hello, RIPEMD-160!1" (original message) "Hello, RIPEMD-160!1x00" (padding with zeros to reach a multiple of 512 bits) "48656C6C6F2C20524950454D442D313 63021800000000000000000" (hexadecimal representation of the padded message)

A = 0x67452301 was used to initialize the variables.

A = 0xC3D2E1F0 B = 0xEFCDAB89 C = 0x98BADCFE D = 0x10325476

Blocks of Processing: The padded message is split into 512-bit blocks and processed in this manner. I'll cut out the specific stages to keep this short.

Final Hash Value: The 160-bit hash value obtained after processing all blocks is "9c5b056b04641533fa2d9fcaf9e66652ff360d5d" in hexadecimal form.

Therefore, "9c5b056b04641533fa2d9fcaf9e66652ff360d5d" is the RIPEMD-160 hash of the input message "Hello, RIPEMD-160!".

RIPEMD-160 is still employed in a number of applications that call for a 160-bit hash value since it is thought to be secure. However, due to their greater bit sizes and resistance to known attacks, SHA-256 and SHA-3 are more frequently advised for new cryptographic applications.

An explanation of the RIPEMD-160 algorithm and an illustration of how it produces a hash value for a sample input message are provided below.

Experimentation:

bring up hashlib

# Enter the message "Hello, RIPEMD-160!" as the message.

# Convert the message's encoding to bytes (UTF-8).

message.encode('utf-8') returns the message's bytes.

ripemd160_hash = hashlib.new('ripemd160'); # Calculate the RIPEMD-160 hash

ripemd160_hash.update(message_bytes)

Hash_value is equal to ripemd160_hash.hexdigest()

"RIPEMD-160 Hash:", "hash_value", print

When you execute this code, it will print the "Hello, RIPEMD-160!" input message's RIPEMD-160 hash. To ensure security and accuracy, it is advised to utilise well-tested cryptographic libraries in real-world applications rather than the Python hashlib library, which is used in the example above for convenience.

Utilization in cryptography

1. **Digital Signatures**: Prior to signing a message with a private key, hash functions are employed to create a fixed-size representation (digest) of the message.

2. **Data Integrity**: Hash functions are used to check the accuracy of data while it is being sent or stored. To verify for any modifications, the recipient might compute the hash of the received data and compare it with the original hash.

3. **Password Storage:** Passwords are safely stored using hash techniques. Only the password's hash is kept on file rather than the original password. When a person tries to log in, their password is hashed and compared against a hash that is already saved.

For cryptographic applications, using a reliable and well-known hash function is crucial to maintaining the system's security and integrity. It is critical to employ the most recent and secure hash functions available because as computing power rises, older hash functions may become vulnerable to attacks.

REFERENCES

[1]. Cybersecurity, Cryptography And Network Security For Beginners: Learn Fast How To Get A Job In Cybersecurity HUGO HOFFMAN **Nov** 2020 · HUGO HOFFMAN · Narrated by Matyas J. and Scott Clem.
[2]. Cryptography and Network Security | 3rd Edition Paperback – 1 January 2015 by Forouzan .
[3]. Handbook of Applied Cryptography, Paul C Van Oorschot, Scoot A Vanstone, A. J. Menezes

[4]. Introduction to Modern Cryptography by Jonathan Katz, Yehuda Lindell
[5]. New Direction on Cryptography, Democratizing Cryptography: The work of Whitfield Diffie and Martin Hellman. August 2022, Page 365-390, https://doi.org/10.1145/3549993.3550007
[6]. R. Merkle, “Secure Communication over an insecure channel” submitted to communications of the ACM [This was subsequently published in volume 21. No .4.pp. 294-299, April.
[7]. A review on steganography and cryptography, Publisher: IEEE, Rina Mishra, Praveen Bhanodiya,
Published in: - 2015 International Conference on Advances Computer Engineering and Application.