Message Integrity

Message integrity: The process of checking the message's authenticity.

Message Authentication: An information security property that indicates that a message has not been modified while in transit (data integrity) and that the receiving party can verify the
source of the message. (or) Data is prone to various attacks. One of these attacks includes message authentication. This threat arises when the user does not have any information about the originator of the message. Message authentication can be achieved using cryptographic methods which further make use of keys.

Authentication requirements:

Revelation: It means releasing the content of the message to someone who does not have an appropriate cryptographic key.

Analysis of Traffic: Determination of the pattern of traffic through the duration of connection and frequency of connections between different parties.

Modification in the sequence: Changing the order of messages between parties. This includes insertion, deletion, and reordering of messages.

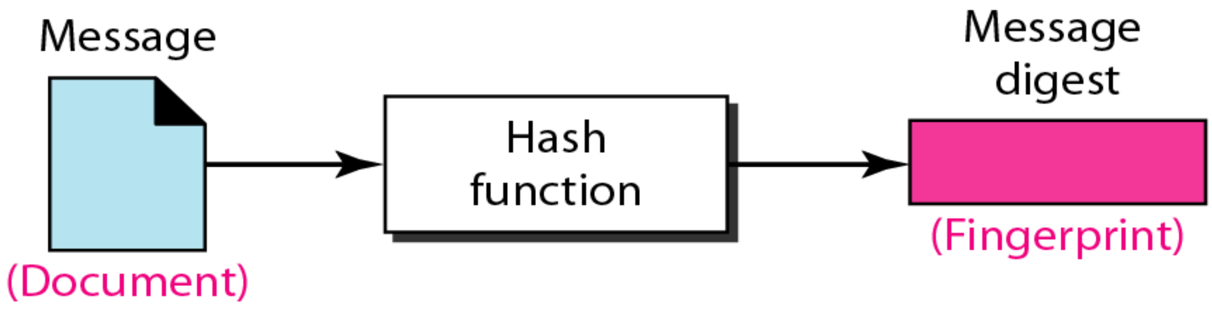
Source Refusal: When the source denies being the originator of a message.

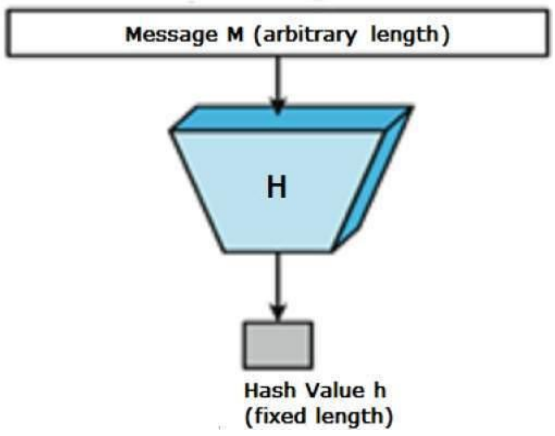
Destination refusal: When the receiver of the message
denies the reception.

Hash function:

Mathematical function used in cryptography which typically take inputs of variable lengths to return outputs of fixed length.

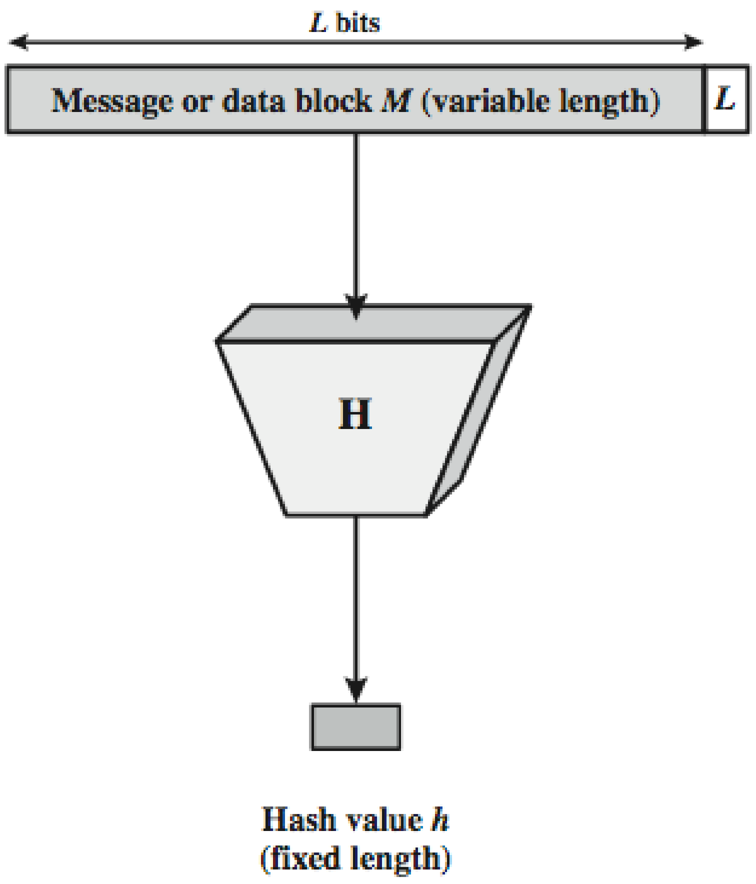
A cryptographic hash function combines the message-passing capabilities of hash functions with security properties.





Cryptographic hash function:

- condenses arbitrary message to fixed size h=H(M)



- want a cryptographic hash function
- computationally infeasible to find data mapping to specific hash (one-way property)
- computationally infeasible to find two data to same hash (collision-free property)

Hash Function Applications

Used Alone

Fingerprint -- file integrity verification, public key fingerprint

Password storage (one-way encryption)

Combined with encryption functions

Message Authentication Code (MAC):

protects both a message's integrity as well as its authenticity

Digital signature

Hand-written signature is a method to prove that a paper file is signed by us and not by someone else. It can prove this and the current hand-written signature is compared with one or more of the earlier handwritten signatures.

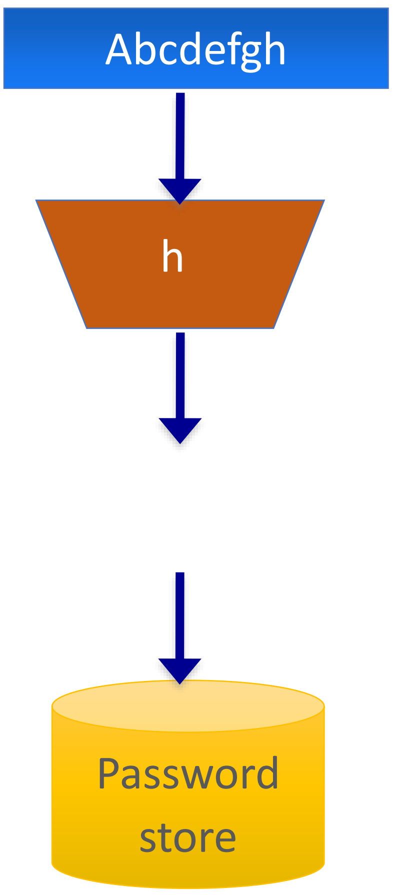
If there is a match then the recipient of the files can securely accept that the files could not have been endorsed by someone else. In case it is the first time, it can have to prove the identity by means of some recognition card, and necessarily by being physically present to sign the files.

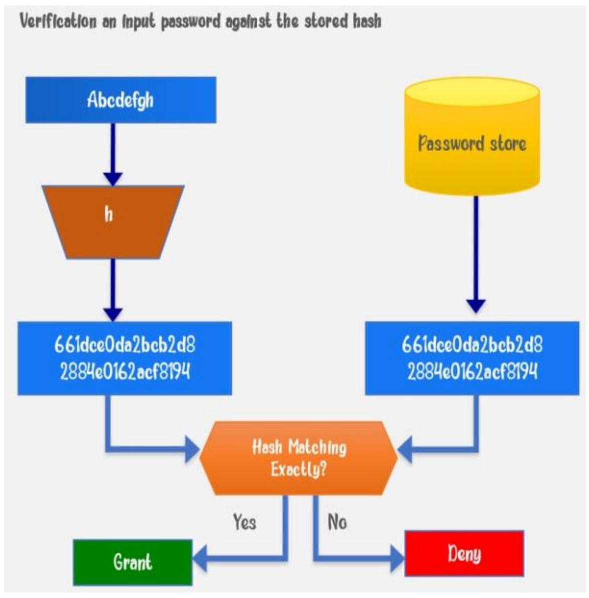
Ensuring Nonrepudiation:

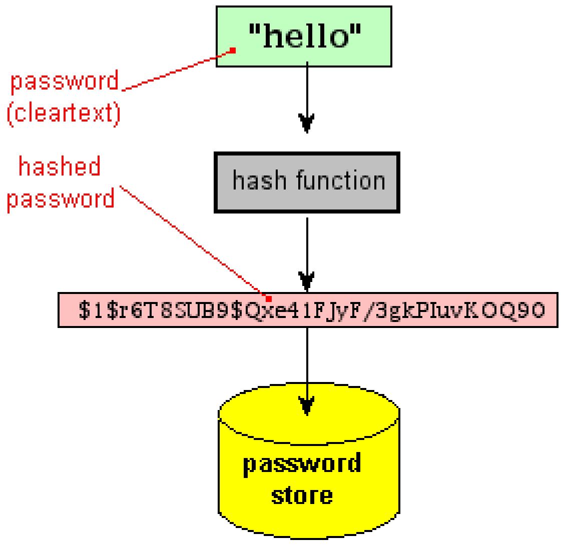
Encrypt hash with private (signing) key and verify with public (verification) key

Password storage

Store Hashing Password




Verification an input password against the stored hash
Abcdefgh
Password store
h
661dce0da2bcb2d8 2884e0162acf8194
661dce0da2bcb2d8 2884e0162acf8194
Hash Matching Exactly?
Yes
No
Grant
Deny

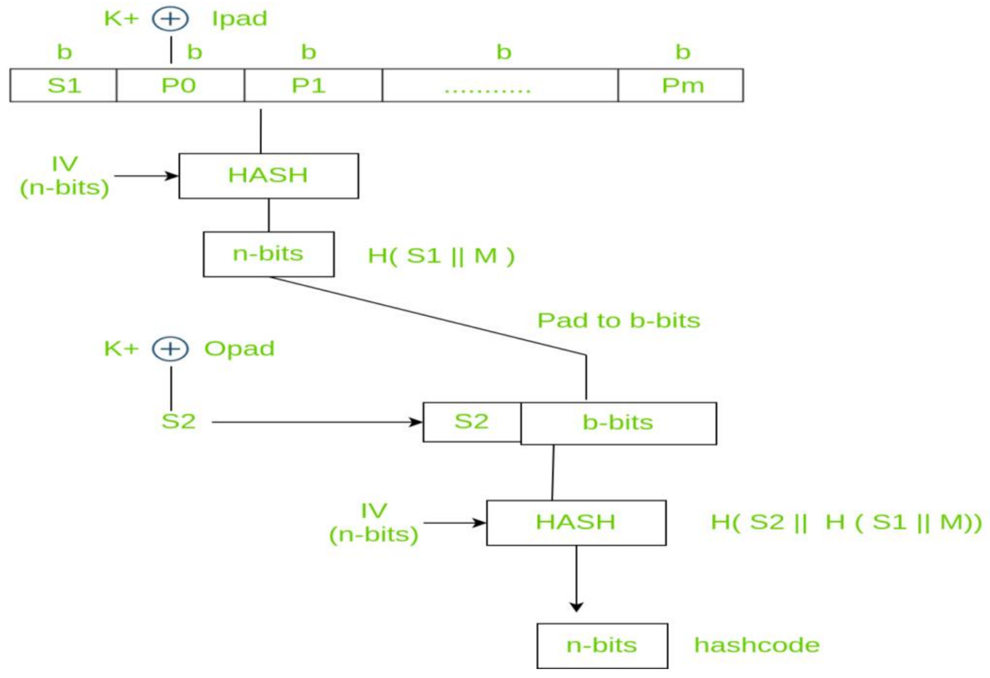


Cryptographic Hash Functions Based MACs

HMAC (Hash-Based Message Authentication Code) is a cryptographic technique that ensures data integrity and authenticity using a hash function and a secret key. Unlike approaches based on signatures and asymmetric cryptography. Checking data integrity is necessary for the parties involved in communication. HTTPS, SFTP, FTPS, and other transfer protocols use HMAC. The cryptographic hash function may be MD-5, SHA-1, or SHA-256. Digital signatures are nearly similar to HMACs i.e. they both employ a hash function and a shared key. The difference lies in the keys i.e. HMAC uses a symmetric key(same copy) while Signatures uses an asymmetric (two different keys).

```
HMAC = hash Func (secret key + message)
```

There are three types of authentication functions. They are message encryption, message authentication code, and hash functions. The major difference between MAC and hash (HMAC here) is the dependence of a key. In HMAC we have to apply the hash function along with a key on the plain text. The hash function will be applied to the plain text message. But before applying, we have to compute S bits and then append it to plain text and after that apply the hash function. For generating those S bits we make use of a key that is shared between the sender and receiver.



SECURE HASH ALGORITHM

- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
- standard is FIPS 180-1 1995, also Internet RFC3174
- based on design of MD4 with key differences
- produces 160-bit hash values
- recent 2005 results on security of SHA-1 have raised concerns on its use in future applications

SHA-512 algorithm

SHA-512 is a hashing algorithm that performs a hashing function on some data given to it.

Hashing algorithms are used in many things such as internet security, digital certificates and even blockchains. Since hashing algorithms play such a vital role in digital security and cryptography, this is an easy-to-understand walkthrough, with some basic and simple maths along with some diagrams, for a hashing algorithm called SHA-512. It's part of a group of hashing algorithms called SHA-2 which includes SHA-256 as well which is used in the bitcoin blockchain for hashing.

SHA-512 does its work in a few stages. These stages go as follows:

1. Input formatting
2. Hash buffer initialization
3. Message Processing
4. Output

Input Formatting:

SHA-512 can't actually hash a message input of any size, i.e. it has an input size limit. The entire formatted message has basically three parts: the original message, padding bits, size of original message. And this should all have a combined size of a whole multiple of 1024 bits. This is because the formatted message will be processed as blocks of 1024 bits each, so each bock should have 1024 bits to work with

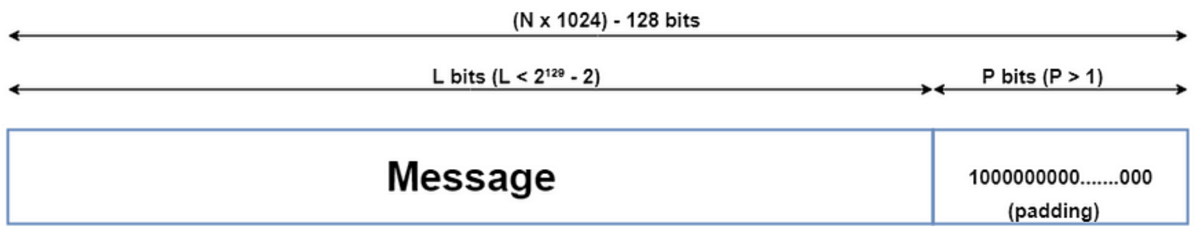
L bits
Message

Padding bits

The input message is taken and some padding bits are appended to it in order to get it to the desired length. The bits that are used for padding are simply '0' bits with a leading '1' (100000…000). Also, according to the algorithm, padding *needs* to be done, even if it is by one bit. So a single padding bit would only be a '1'.

The total size should be equal to 128 bits short of a multiple of 1024 since the goal is to have the formatted message size as a multiple of 1024 bits (N x 1024).

<pic: msg + pad>



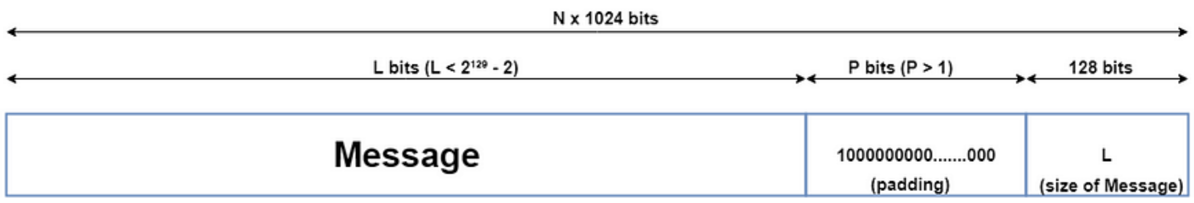
Message with padding

Size of original message

The size of the original message given to the algorithm is appended. This size value needs to be represented in 128 bits and is the only reason that the SHA-512 has a limitation for its input message.

Since the size of the original message needs to be represented in 128 bits and the largest number that can be represented using 128 bits is (2^{128}-1), the message size can be at most (2^{128}-1) bits; and also taking into consideration the necessary single padding bit, the maximum size for the original message would then be (2^{128}-2). Even though this limit exists, it doesn't actually cause a problem since the actual limit is so high (2^{128}-2 = 340,282,366,920,938,463,463,374,607,431,768,211,454 bits).

<pic: message + pad +size>   message with padding and size



Now that the padding bits and the size of the message have been appended, we are left with the completely formatted input for the SHA-512 algorithm.

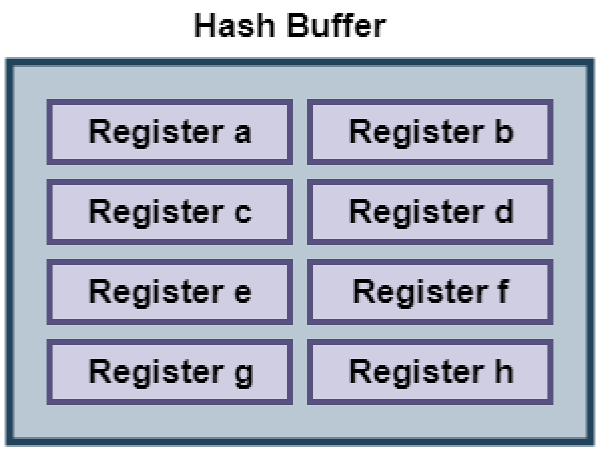
Formatted message

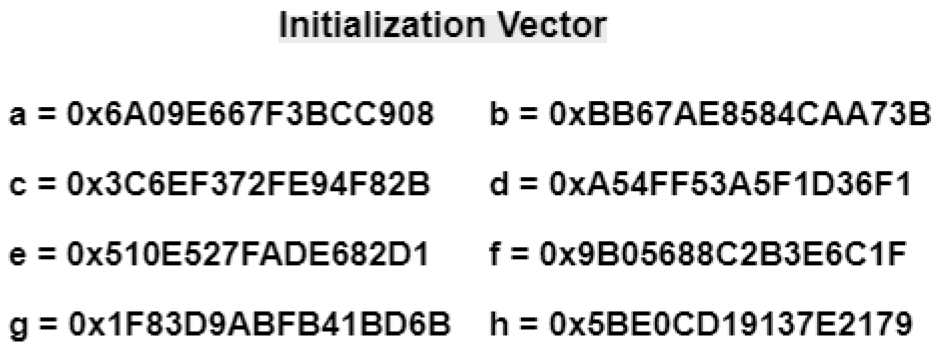
2. Hash buffer initialization:

The algorithm works in a way where it processes each block of 1024 bits from the message using the result from the previous block. Now, this poses a problem for the first 1024 bit block which can’t use the result from any previous processing.

Since each intermediate result needs to be used in processing the next block, it needs to be stored somewhere for later use. This would be done by the *hash buffer*, this would also then hold the final hash digest of the entire processing phase of SHA-512 as the last of these ‘intermediate’ results.

The actual value used is of little consequence, but for those interested, the values used are obtained by taking the first 64 bits of the fractional parts of the square roots of the first 8 prime numbers (2,3,5,7,11,13,17,19). These values are called the Initial Vectors (IV).





Initialization Vector

a = 0x6A09E667F3BCC908	b = 0xBB67AE8584CAA73B
c = 0x3C6EF372FE94F82B	d = 0xA54FF53A5F1D36F1
e = 0x510E527FADE682D1	f = 0x9B05688C2B3E6C1F
g = 0x1F83D9ABFB41BD6B	h = 0x5BE0CD19137E2179

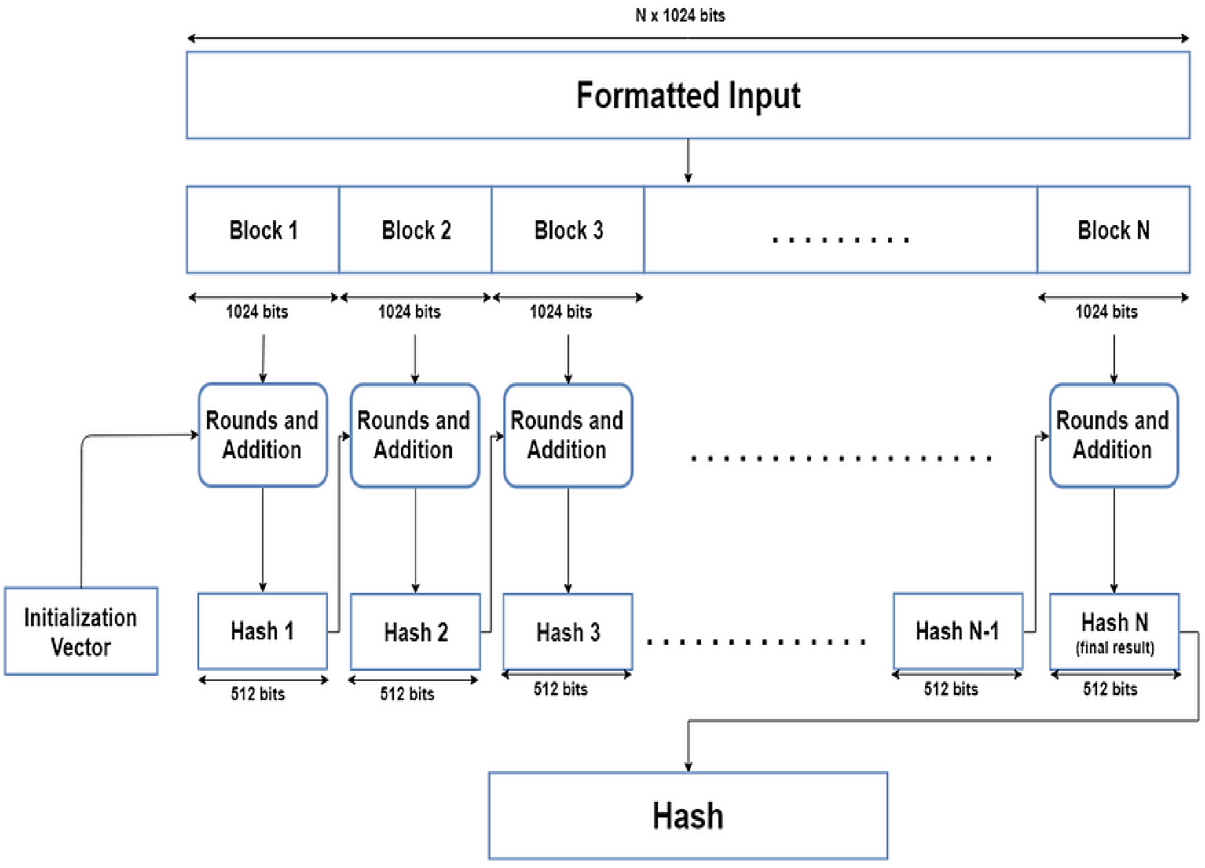
Message Processing:

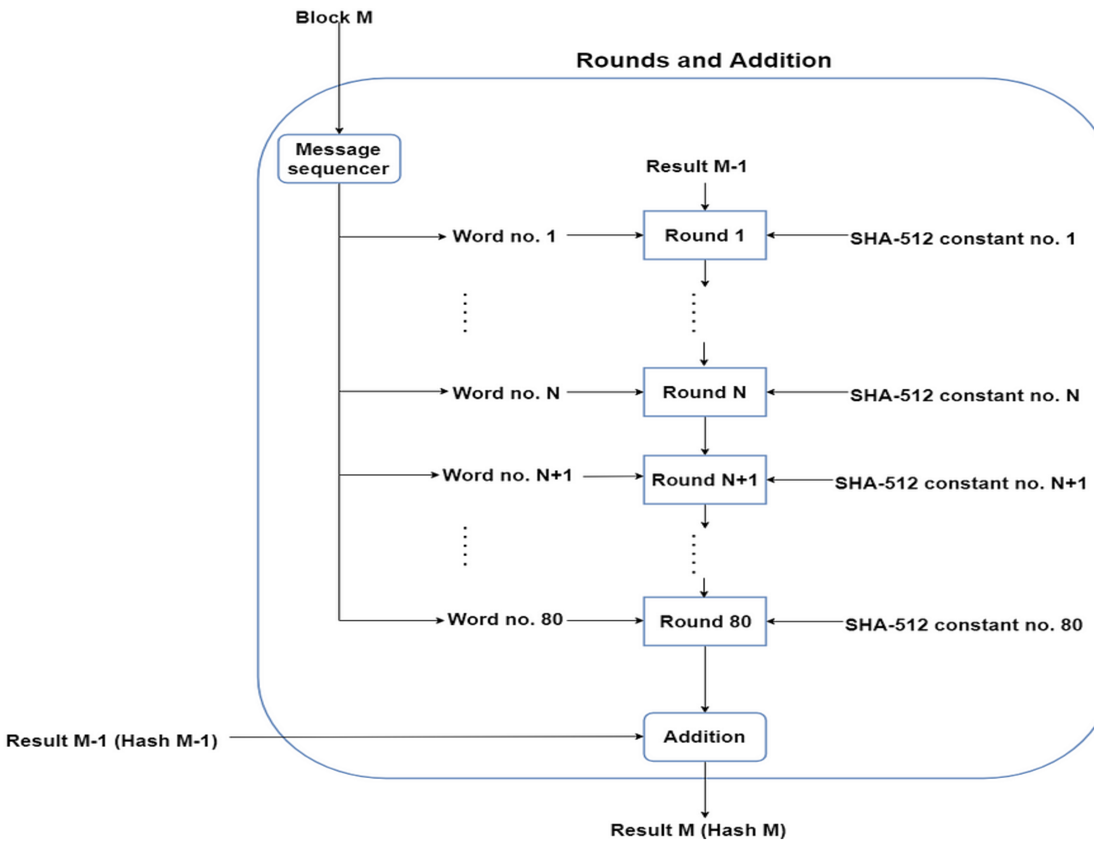
Message processing is done upon the formatted input by taking one block of 1024 bits at a time. The actual processing takes place by using two things: The 1024 bit block, and the result from the previous processing.

This part of the SHA-512 algorithm consists of several 'Rounds' and an addition operation.

<pic: Formatted input 1024 bit blocks; F(M.n ,H.n-1)=H.n>

So, the Message block (1024 bit) is expanded out into 'Words' using a 'message sequencer'. Eighty Words to be precise, each of them having a size of 64 bits.



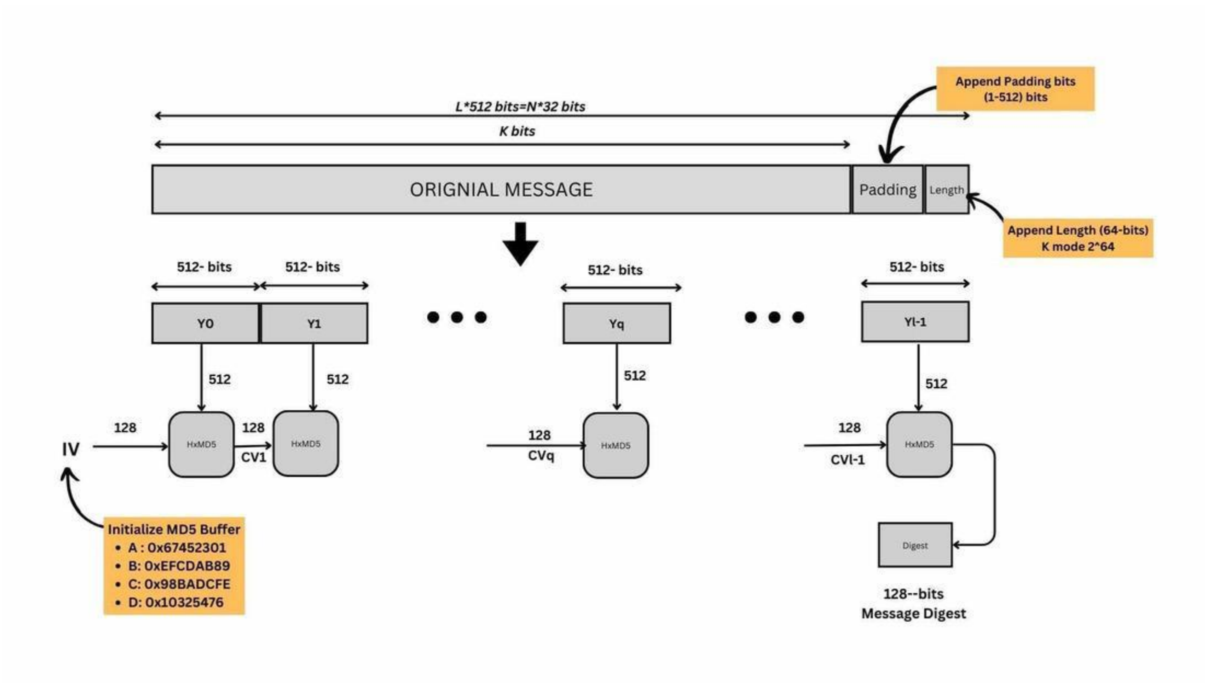


SHA 3 ALGORITHM

The SHA or SHA-3 (Secure Hash Algorithm 3) is known to be the latest member of the SHA family of the secure hash algorithm stands it is published by the NIST on the year 2015. the SHA-3 is different when we look at the internal structure of the SHA-3 because the **MD-5 structure** is different from the SHA-1 and SHA-2. the SHA-3 is designed in order to provide a "random mapping" from a particular string of binary data to a "**message digest**" that is fixed in size, the SHA-3 also helps to achieve more security properties.

MD-5 ALGORITHM

MD5 is a cryptographic hash function algorithm that takes the message as input of any length and changes it into a fixed-length message of 16 bytes. MD5 algorithm stands for the **message-digest algorithm**. MD5 was developed in 1991 by **Ronald Rivest** as an improvement of MD4, with advanced security purposes. The output of MD5 (Digest size) is always **128 bits.**



Overview Of MD5 Algorithm

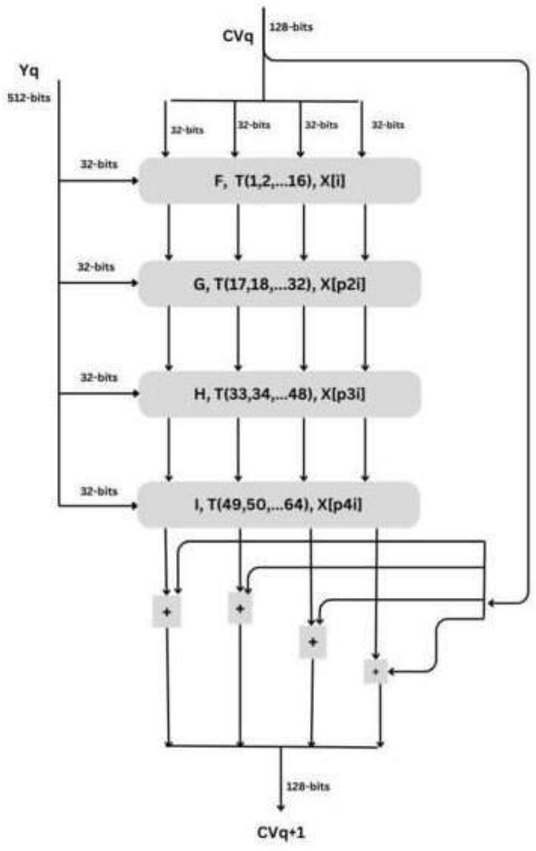
Working of the MD5 Algorithm

MD5 algorithm follows the following steps

1. Append Padding Bits: In the first step, we add padding bits in the original message in such a way that the total length of the message is 64 bits less than the exact multiple of 512.

Suppose we are given a message of 1000 bits. Now we have to add padding bits to the original message. Here we will add 472 padding bits to the original message. After adding the padding bits the size of the original message/output of the first step will be 1472 i.e. 64 bits less than an exact multiple of 512 (i.e. 512*3 = 1536).

Length(original message + padding bits) = 512 * i – 64 where i = 1,2,3 . . .

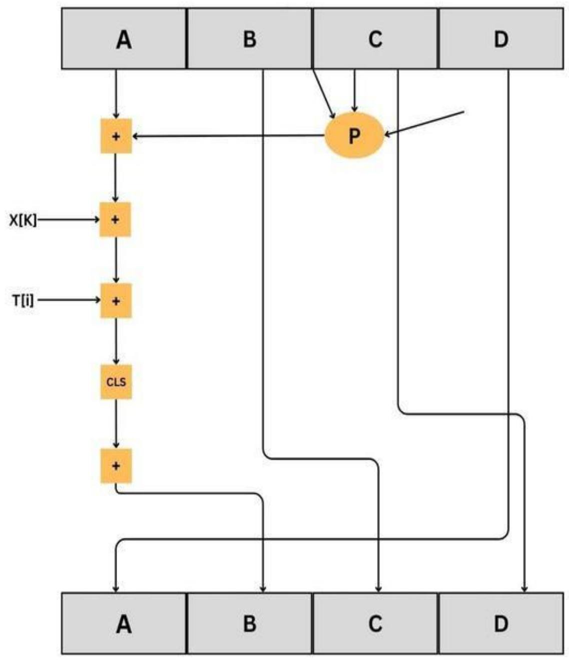


Append Length Bits: In this step, we add the length bit in the output of the first step in such a way that the total number of the bits is the perfect multiple of 512. Simply, here we add the 64-bit as a length bit in the output of the first step.
i.e. output of first step = 512 * n – 64
length bits = 64.

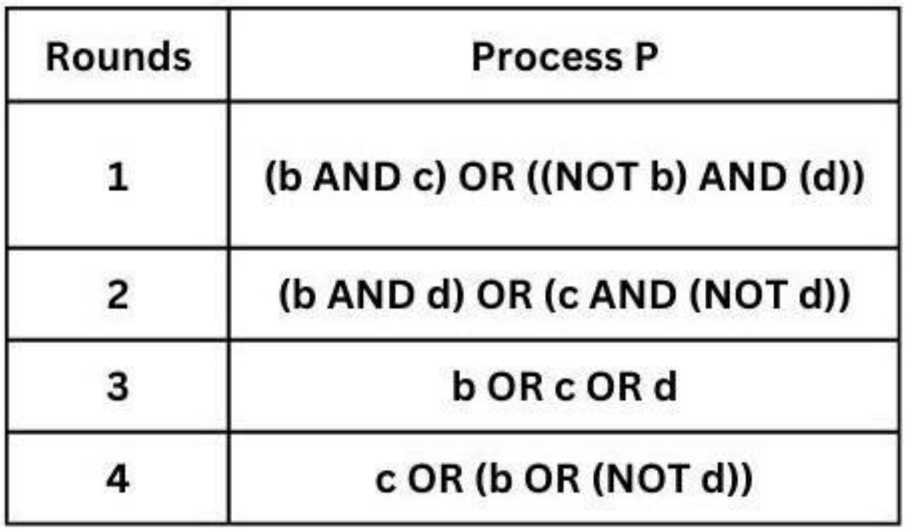
After adding both we will get **512 * n** i.e. the exact multiple of 512.

3. Initialize MD buffer: Here, we use the 4 buffers i.e. A B, C, and D. The size of each buffer is 32 bits.

- A = 0x67425301
- B = 0xEDFCBA45
- C= 0x98CBADFE
- D = 0x13DCE476



Process Each 512-bit Block: This is the most important step of the MD5 algorithm. Here, a total of 64 operations are performed in 4 rounds. In the 1st round, 16 operations will be performed, 2nd round 16 operations will be performed, 3rd round 16 operations will be performed, and in the 4th round, 16 operations will be performed. We apply a different function on each round i.e. for the 1st round we apply the F function, for the 2nd G function, 3rd for the H function, and 4th for the I function.
We perform OR, AND, XOR, and NOT (basically these are logic gates) for calculating functions. We use 3 buffers for each function i.e. B, C, D.



Rounds	Process P
1	(b AND c) OR ((NOT b) AND (d))
2	(b AND d) OR (c AND (NOT d))
3	b OR c OR d
4	c OR (b OR (NOT d))

After applying the function now we perform an operation on each block. For performing operations we need

- add modulo 232

- M[i] – 32 bit message.
- K[i] – 32-bit constant.
- <<<n – Left shift by n bits.