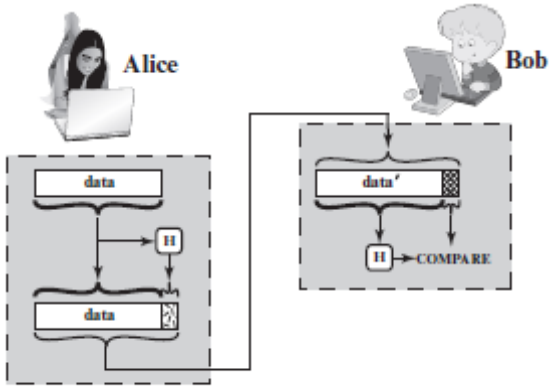
Unit 4: Cryptographic Hash Functions and Digital Signatures

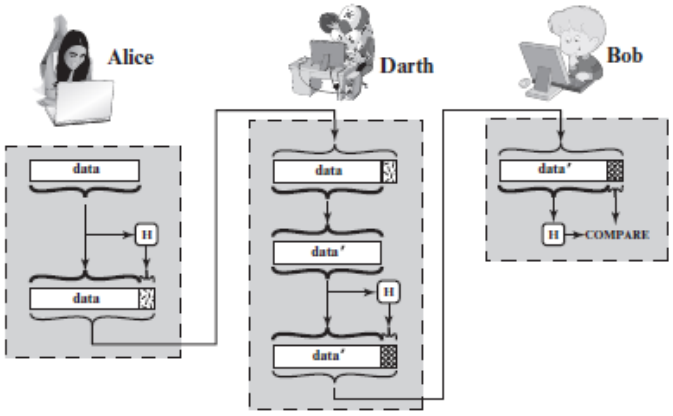
Message Authentication:

- Message authentication is a mechanism or service used to verify the integrity of a message.
- Message authentication assures that data received are exactly as sent (i.e., there is no modification, insertion, deletion, or replay).
- In many cases, there is a requirement that the authentication mechanism assures that purported identity of the sender is valid.
- When a hash function is used to provide message authentication, the hash function value is often referred to as a message digest.
- The essence of the use of a hash function for message integrity is as follows.
    - The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message. The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value.
    - If there is a mismatch, the receiver knows that the message (or possibly the hash value) has been altered.



(a) Use of hash function to check data integrity

    - The hash value must be transmitted in a secure fashion. That is, the hash value must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver. This type of attack is shown in Figure (b).



(b) Man-in-the-middle attack

- In this example, Alice transmits a data block and attaches a hash value. Darth intercepts the message, alters or replaces the data block, and calculates and attaches a new hash value. Bob receives the altered data with the new hash value and does not detect the change. To prevent this attack, the hash value generated by Alice must be protected.

Figure below illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows.

a) The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.
b) Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.
c) It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S. A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B possesses S, it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.
d) Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.

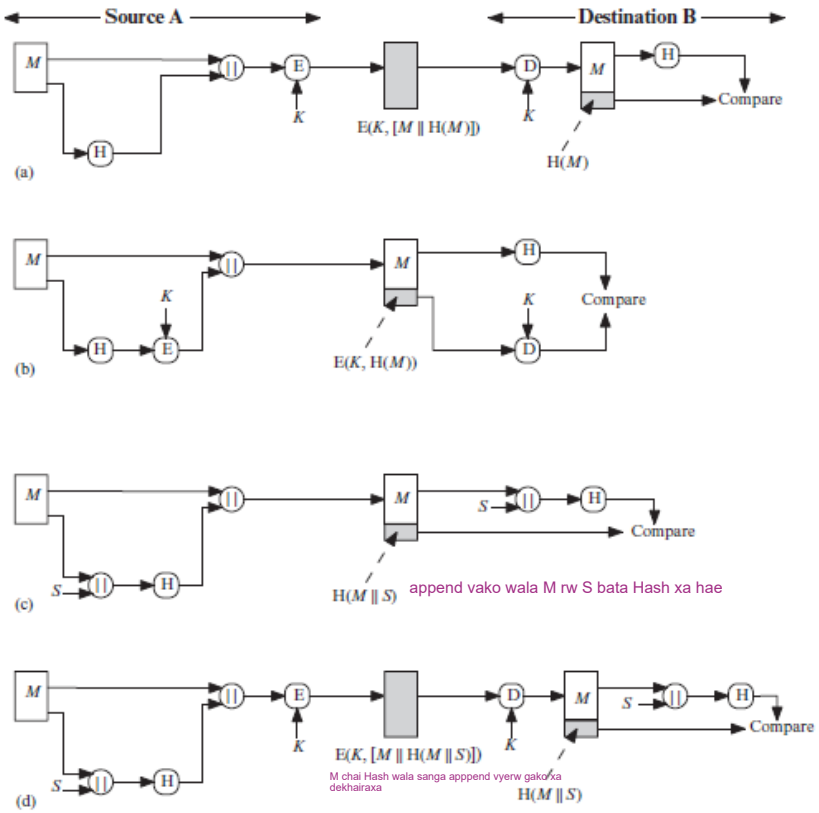


Figure 11.3 Simplified Examples of the Use of a Hash Function for Message Authentication

When confidentiality is not required, method (b) has an advantage over methods (a) and (d), which encrypts the entire message, in that less computation is required. Nevertheless, there has been growing interest in techniques that avoid encryption.

Several reasons for this interest are pointed out as:

- Encryption software is relatively slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
- Encryption hardware costs are not negligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.
- Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
- Encryption algorithms may be covered by patents, and there is a cost associated with licensing their use.

Message Authentication Functions:

Any message authentication mechanism has two levels of functionality. At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message.

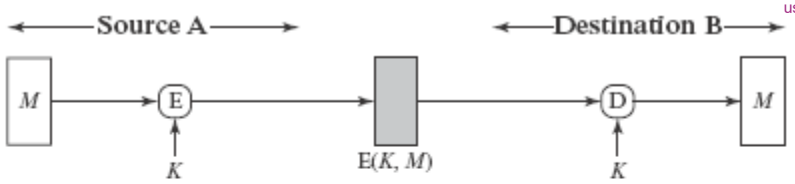
There are three types of functions that may be used to produce an authenticator.

- Hash function: A function that maps a message of any length into a fixed- length hash value, which serves as the authenticator.
- Message encryption: The ciphertext of the entire message serves as its authenticator.
- Message authentication code (MAC): A function of the message and a secret key that produces a fixed-length value that serves as the authenticator.

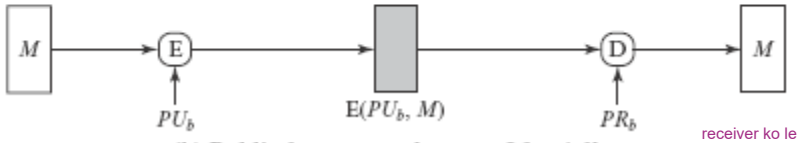
Message Encryption:

Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryption schemes.

signature xaenw kinaki tesko lagi sender ko private key use vakae huan prxa



(a) Symmetric encryption: confidentiality and authentication

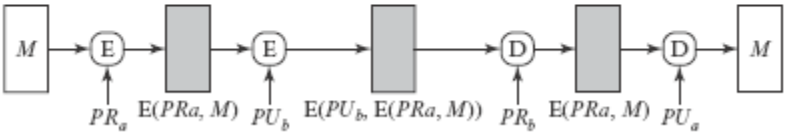


(b) Public-key encryption: confidentiality

receiver ko le garxa vne confidentiality provide garxa receiver samma napugi disclose nahune vyo eti sama ki sender lai ni receiver ko private key thahaxaenw

eha sender ko use vairaxa aba sender le afno private key le garesi signature vyo ani uskae public key le kholxa vnesi authentication vyo sender le nae ho vnne vyo ni tw pathako

(c) Public-key encryption: authentication and signature



(d) Public-key encryption: confidentiality, authentication, and signature

Message Authentication Code (MAC):

A authentication technique that involves the use of a secret key to generate a small fixed-size block of data that is appended to the message is known as Message Authentication Code (MAC). This technique assumes that two communicating parties, say A and B, share a common secret key K. When A has a message to send to B, it calculates the MAC as a function of the message and the key:

MAC = C (K, M)

where

M = input message

C = MAC function

K = shared secret key

MAC = message authentication code

The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC.

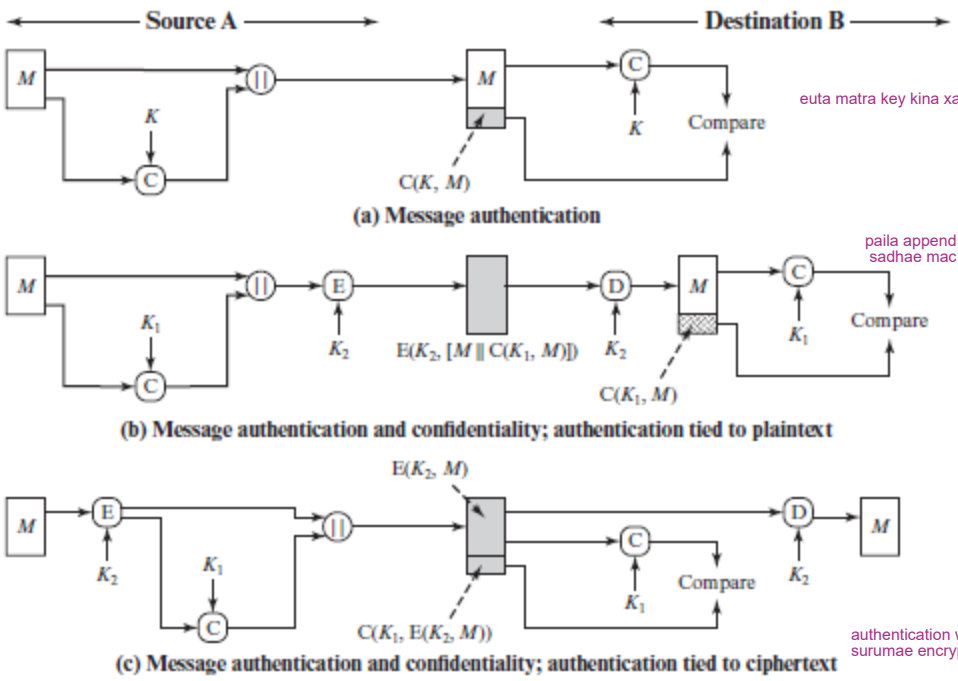


Figure 12.4 Basic Uses of Message Authentication code (MAC)

euta matra key kina xa vnda encryption nae vako xanew

paila append ani encryption
sadhae mac function ma k1 nae use vako xa

tyo encryption gareko wala lai
decrypt garesi msg auxa
ani tehi encrypt gareko cipher text batae
ho mac code generate gareko so tes
batae mac code grne

authentication wala kaam cipher text ma hunxa vnyaxa so
surumae encryption

If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then:

- The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.
- The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
- If the message includes a sequence number (such as is used with TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must be for decryption. In general, the MAC function is a many-to-one function. The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys.

Cryptographic Hash Function:

- A cryptographic hash function (CHF) is a hash function that is suitable for use in cryptography.
- It is a mathematical algorithm that maps data of arbitrary size (often called the "message") to a bit string of a fixed size (the "hash value", "hash", or "message digest") and is a one-way function, that is, a function which is practically infeasible to invert.
- Ideally, the only way to find a message that produces a given hash is to attempt a brute-force search of possible inputs to see if they produce a match, or use a table of matched hashes.
- Cryptographic hash functions are a basic tool of modern cryptography.

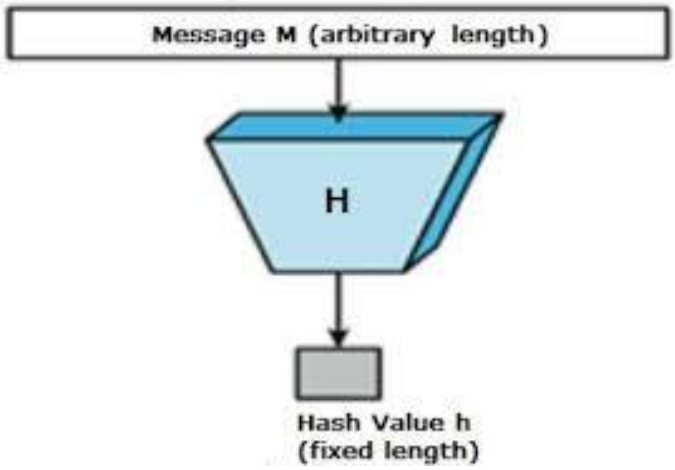
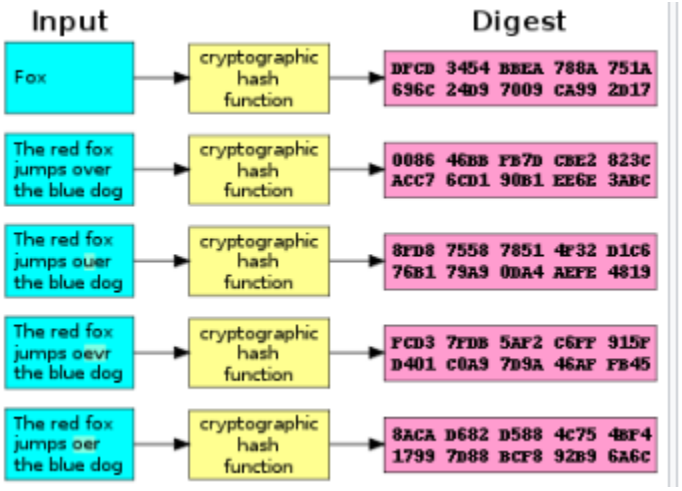


Figure: Cryptographic Hash Function; h = H(M)

Properties of Hash Function: book page no 5

The ideal cryptographic hash function has the following main properties:

- it is deterministic, meaning that the same message always results in the same hash
- it is quick to compute the hash value for any given message
- it is infeasible to generate a message that yields a given hash value (Pre-Image Resistant)
- it is infeasible to find two different messages with the same hash value (Collision Resistance)
- A small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value (avalanche effect)



Applications of Hash Function:

Given that the hash depends on the input to the hash function and will change with the input hash functions are used:

- to ensure that messages have not been tampered with (message authentication, digital signatures, checksums) and

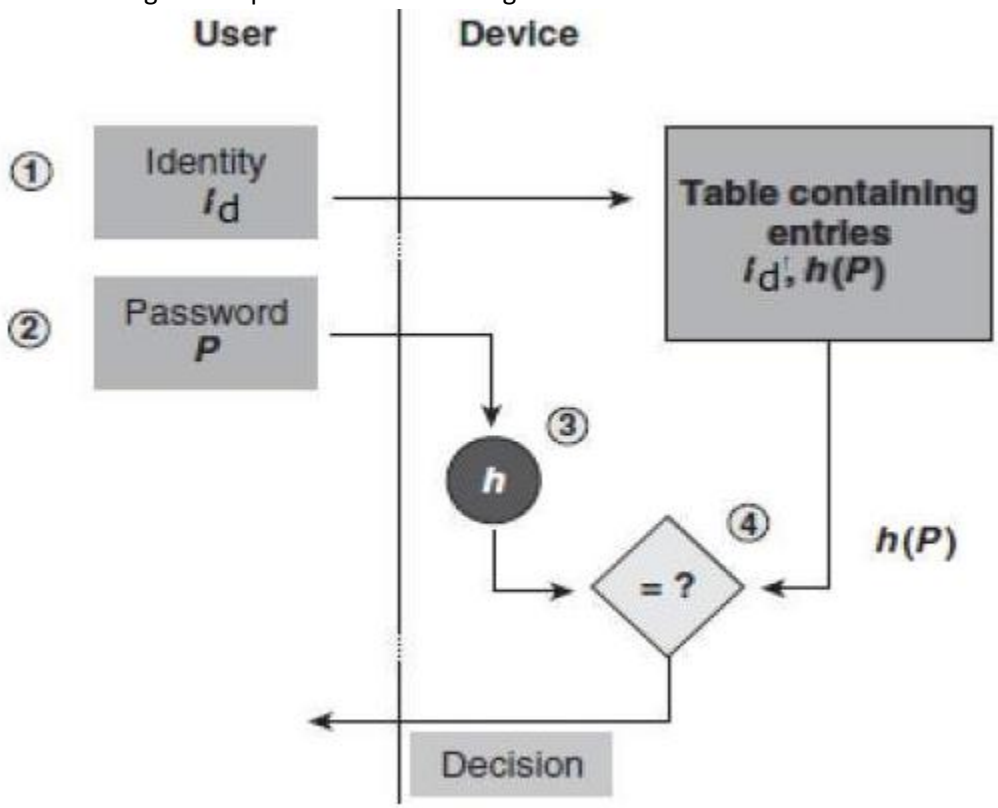
- to check for equality while preserving secrecy / efficiently (for example, checking if password is correct without storing the actual password, or checking for duplicates in lists of large items).
- They are also used as proof-of-work (for example, in cryptocurrencies like bitcoin), error-correcting codes, randomization and to make cryptographic algorithms more efficient.

There are two direct applications of hash function based on its cryptographic properties.

Password Storage

Hash functions provide protection to password storage.

- Instead of storing password in clear, mostly all logon processes store the hash values of passwords in the file.
- The Password file consists of a table of pairs which are in the form (user id, h(P)).
- The process of logon is depicted in the following illustration:



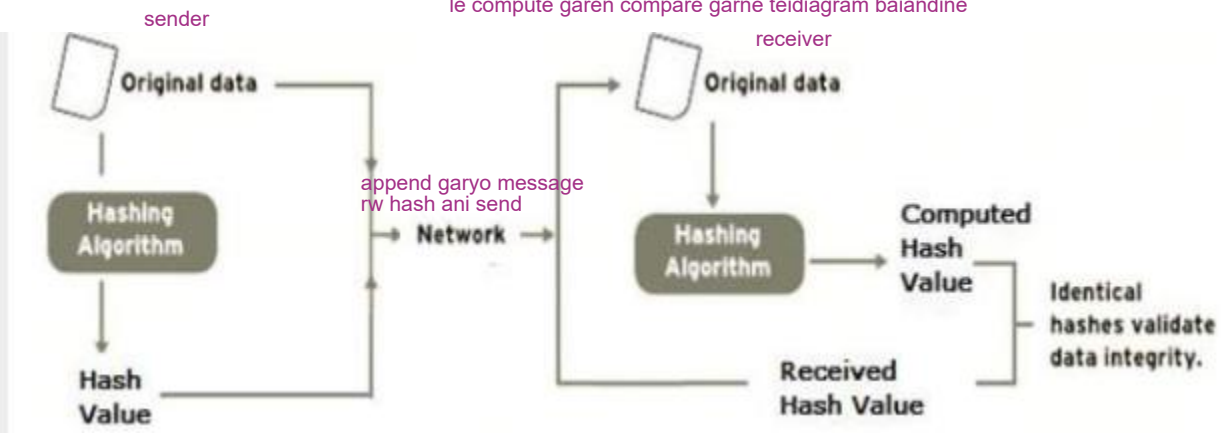
- An intruder can only see the hashes of passwords, even if he accessed the password. He can neither logon using hash nor can he derive the password from hash value since hash function possesses the property of pre-image resistance.

Data Integrity Check

Data integrity check is a most common application of the hash functions. It is used to generate the checksums on data files. This application provides assurance to the user about correctness of the data.

The process is depicted in the following illustration:

tyo agadi ko original data ma hash fxn ani append send ani receiver le compute garen compare garne teidiagram baiandine



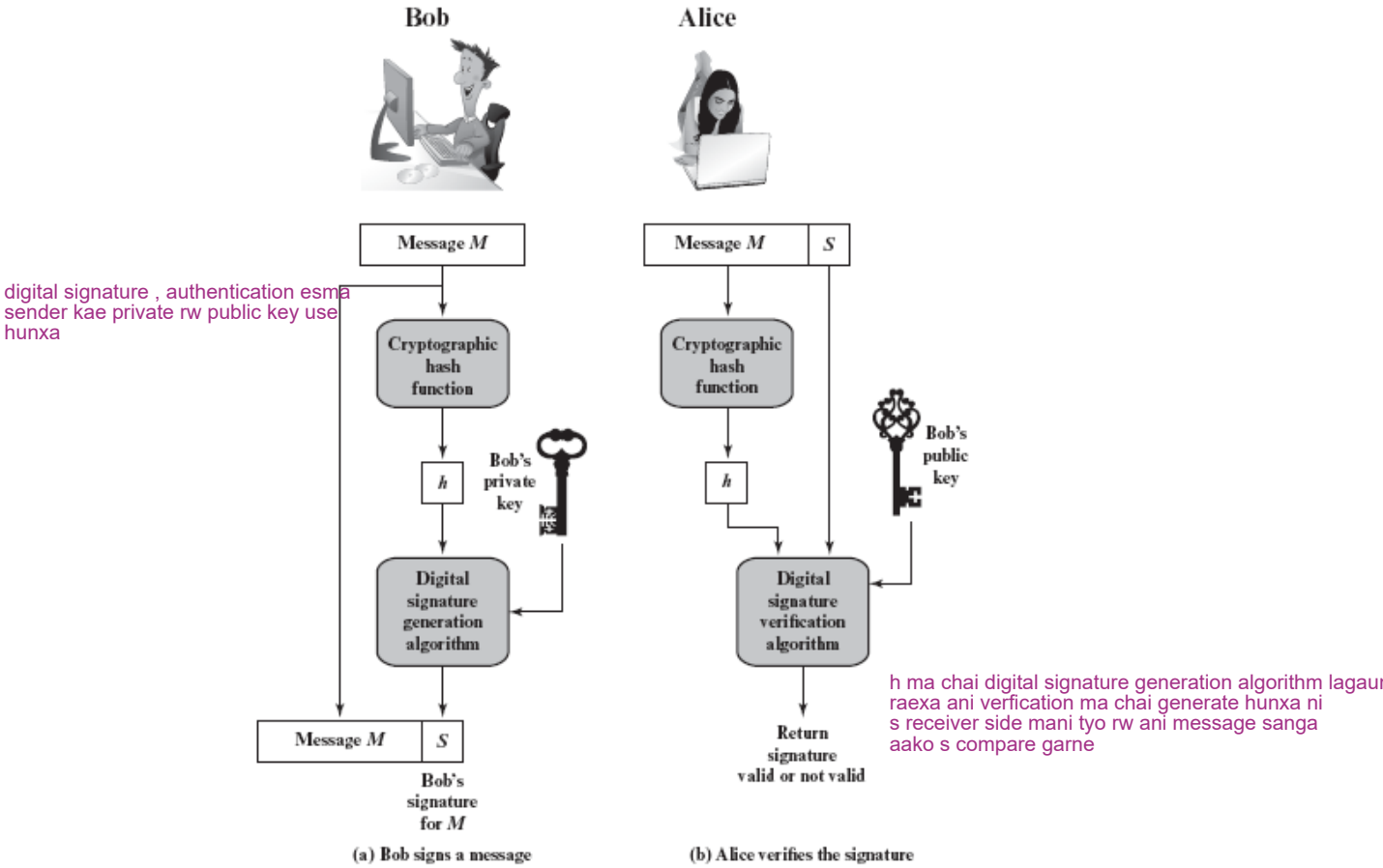
The integrity check helps the user to detect any changes made to original file. It however, does not provide any assurance about originality. The attacker, instead of modifying file data, can change the entire file and compute all together new hash and send to the receiver. This integrity check application is useful only if the user is sure about the originality of file.

Digital Signature:

Digital Signature is a process that guarantees that the contents of a message have not been altered in transit. In other words, a digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document.

As the digital equivalent of a handwritten signature or stamped seal, a digital signature offers far more inherent security, and it is intended to solve the problem of tampering and impersonation in digital communications. Digital signatures can provide the added assurances of evidence of origin, identity and status of an electronic document, transaction or message and can acknowledge informed consent by the signer. Digital signatures are based on public key cryptography. Using a public key algorithm, such as RSA, one can generate two keys that are mathematically linked: one private and one public.

- Suppose that Bob wants to send a message to Alice.
- Although it is not important that the message be kept secret, he wants Alice to be certain that the message is indeed from him.
- For this purpose, Bob uses a secure hash function, such as SHA-512, to generate a hash value for the message. That hash value, together with Bob's private key serves as input to a digital signature generation algorithm, which produces a short block that functions as a digital signature.
- Bob sends the message with the signature attached.
- When Alice receives the message plus signature, she
  (1) calculates a hash value for the message;
  (2) provides the hash value and Bob's public key as inputs to a digital signature verification algorithm.
- If the algorithm returns the result that the signature is valid, Alice is assured that the message must have been signed by Bob. No one else has Bob's private key and therefore no one else could have created a signature that could be verified for this message with Bob's public key.
- In addition, it is impossible to alter the message without access to Bob's private key, so the message is authenticated both in terms of source and in terms of data integrity.



digital signature , authentication esma sender kae private rw public key use hunxa

h ma chai digital signature generation algorithm lagaun raexa ani verfication ma chai generate hunxa ni s receiver side mani tyo rw ani message sanga aako s compare garne

Fig: Simplified Depiction of Essential Elements of Digital Signature Process

Properties:

yo properties sabai arbitrated digital signature snga milxa

- Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. Several forms of dispute between the two parties are possible.
- In situations where there is not complete trust between sender and receiver, something more than authentication is needed. The most attractive solution to this problem is the digital signature. The digital signature must have the following properties:
  - It must verify the author and the date and time of the signature.
  - It must authenticate the contents at the time of the signature.
  - It must be verifiable by third parties, to resolve disputes.
- Thus, the digital signature function includes the authentication function.

Direct Digital Signature

- The term direct digital signature refers to a digital signature scheme that involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source.

- Confidentiality can be provided by encrypting the entire message plus signature with a shared secret key (symmetric encryption). Note that it is important to perform the signature function first and then an outer confidentiality function.
- In case of dispute, some third party must view the message and its signature. If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message.
- However, if the signature is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.

encryption decryption le chai confidentiality provide garxa

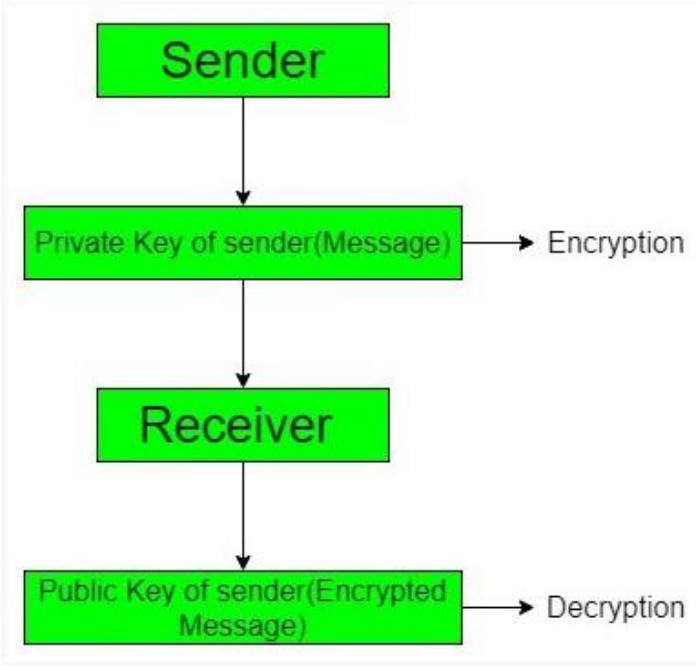


Fig: Direct Digital Signature

The Arbitrated Digital Signature

integrity authenticity check grne tw ho suruma msg thik xa ki xaenw herxa ani time stamp xa ki xaenw herxa ani thik xa ki xanew heresi integrity check hune vyo third pparty involve vyesi deny grna mildenw ani time stamp vyesi pxi pani corrupted huan didenw

- An arbitrated digital signature invites a third party into the process called a "trusted arbiter".
- The Arbitrated Digital Signature includes three parties in which one is sender, second is receiver and the third is arbiter who will become the medium for sending and receiving message between them.
- The role of the trusted arbiter is usually twofold: first this independent third party verifies the integrity of the signed message or data. Second, the trusted arbiter dates, or time-stamps, the document, verifying receipt and the passing on of the signed document to its intended final destination.
- The sender cannot deny sending of the message since it has to go through the arbiter before the recipient sees the message.
- The message is less prone to get corrupted because of timestamp being included by default.

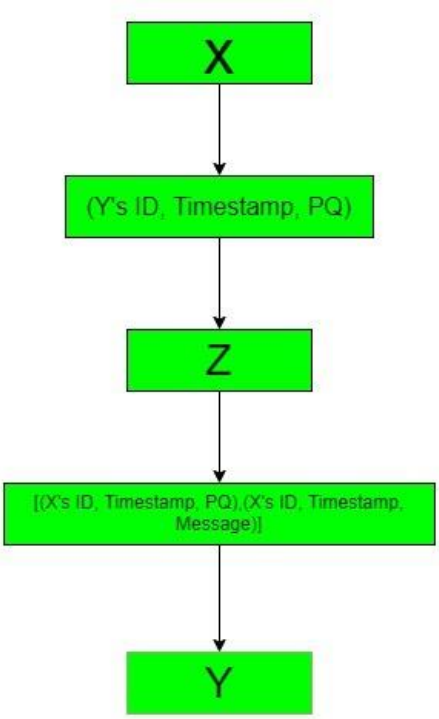


Fig: Arbitrated Digital Signature

S NO.	DIRECT DIGITAL SIGNATURE	ARBITRATED DIGITAL SIGNATURE
1.	It only require the communicating parties.	It requires arbiter along with communicating parties to send or receive messages.
2.	In this the digital signature encrypts the whole plain text with the sending party's private key.	The encrypted message is send by X to arbiter Z with Y's id, timestamp and some random number PQ.
3.	The message is directly transmitted between both parties without any help of a intermediate.	Arbiter is needed to transmit the message.
4.	Timestamp is not maintained by both side.	Timestamp is maintained by all three members by default.
5.	In case the confidentiality is needed the message will be encrypt with receiver's public key or a shared key.	The arbiter provides confidentiality of the message.
6.	Vulnerable to any kind of replay attack.	The timestamp is used to protect the message from any kind of replay attack.

Type text here

confidentiality
le msg disclose huan didenw

repudiation may occur

avoid repudiation

RSA ma chai hash ma encryption garinthyo esma chai signing griraxa

Digital Signature: The DSA Approach

- The DSA uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange but it is a public-key technique.
- The DSA approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number k generated for this particular signature.
- The signature function also depends on the sender's private key (PR_a) and a set of parameters known to a group of communicating principals.
- We can consider this set to constitute a global public key (PU_G). The result is a signature consisting of two components, labeled s and r.
- At the receiving end, the hash code of the incoming message is generated. The hash code and the signature are inputs to a verification function.
- The verification function also depends on the global public key as well as the sender's public key (PU_a), which is paired with the sender's private key.
- The output of the verification function is a value that is equal to the signature component r if the signature is valid.
- The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

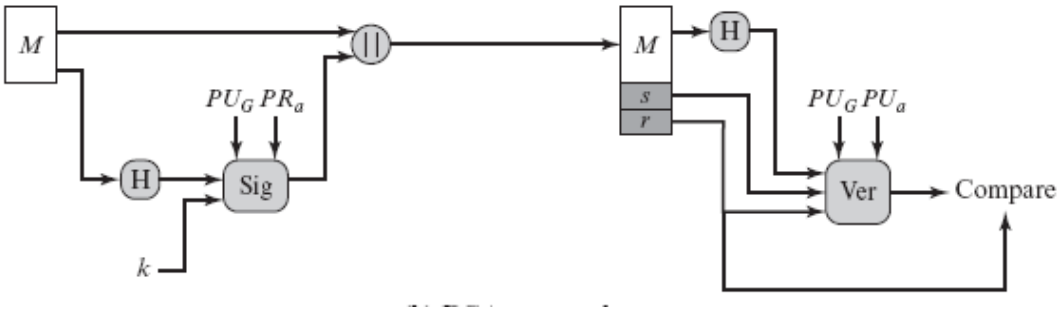


Fig: DSA Approach

Digital Signature: The RSA Approach

- In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature.
- Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key.
- If the calculated hash code matches the decrypted signature, the signature is accepted as valid.
- Because only the sender knows the private key, only the sender could have produced a valid signature.

agadi ko rsa vnda ulto kasari vnda sender side le agadi public key matra use garthyo bt eha tw digital signature ho so sender le private key nae use grxa

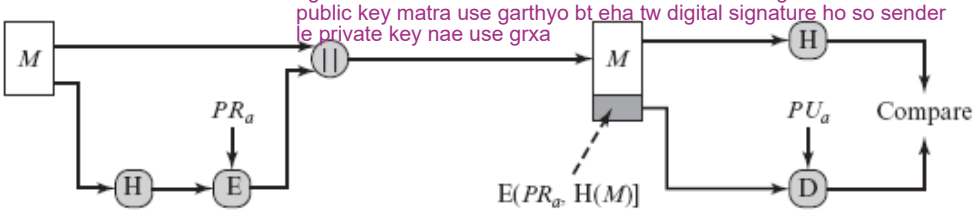
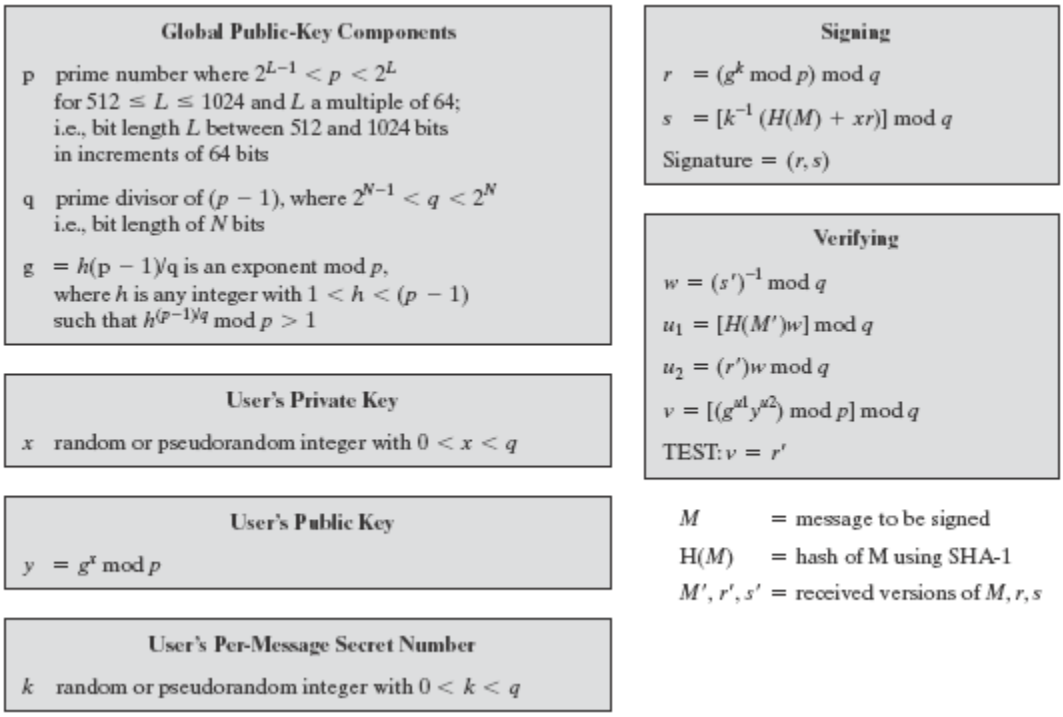


Fig: RSA Approach

The Digital Signature Algorithm (DSA):

DSA is based on the difficulty of computing discrete logarithms and is based on schemes originally presented by Elgamal.

Figure below summarizes the algorithm.



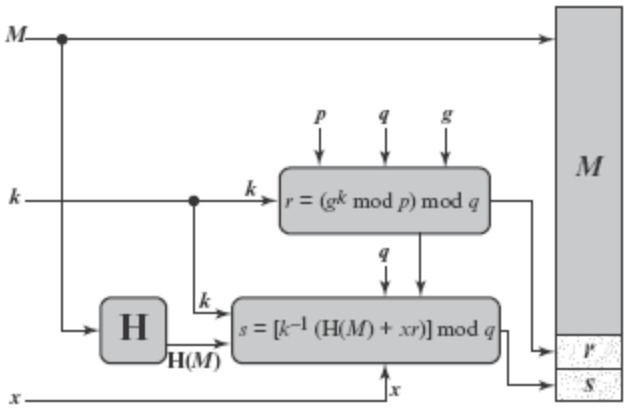
There are three parameters that are public and can be common to a group of users. An N-bit prime number q is chosen. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides (p - 1). Finally, g is chosen to be of the form h(p-1)/q mod p, where h is an integer between 1 and (p - 1) with the restriction that g must be greater than 1.

With these parameters in hand, each user selects a private key and generates a public key. The private key x must be a number from 1 to (q - 1) and should be chosen randomly or pseudorandomly. The public key is calculated from the private key as $y = g^x \bmod p$. The calculation of y given x is relatively straightforward. However, given the public key y, it is believed to be computationally infeasible to determine x, which is the discrete logarithm of y to the base g, mod p.

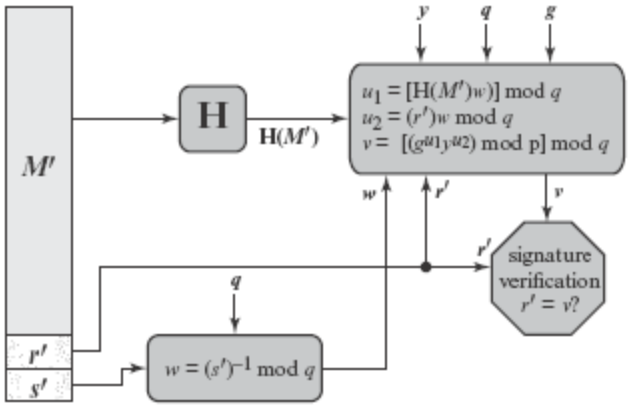
The signature of a message M consists of the pair of numbers r and s, which are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message H(M), and an additional integer k that should be generated randomly or pseudorandomly and be unique for each signing.

Let M, r', and s' be the received versions of M, r, and s, respectively. Verification is performed using the formulas shown in Figure above. The receiver generates a quantity v that is a function of the public key components, the sender's public key, the hash code of the incoming message, and the received versions of r and s. If this quantity matches the r component of the signature, then the signature is validated.

Figure below depicts the functions of signing and verifying.



(a) Signing



(b) Verifying

Note that the test at the end is on the value r, which does not depend on the message at all. Instead, r is a function of k and the three global public-key components. The multiplicative inverse of k (mod q) is passed to a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover r using the incoming message and signature, the public key of the user, and the global public key.

Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover k from r or to recover x from s. Another point worth noting is that the only computationally demanding task in signature generation is the exponential calculation g^k mod p. Because this value does not depend on the message to be signed, it can be computed ahead of time. Indeed, a user could pre-calculate a number of values of r to be used to sign documents as needed. The only other somewhat demanding task is the determination of a multiplicative inverse, k^{-1}. Again, a number of these values can be precalculated.

Message Digest: MD4 Algorithm

MD4 is a message digest algorithm (the fourth in a series) designed by Professor Ronald Rivest of MIT in 1990. It implements a cryptographic hash function for use in message integrity checks. The digest length is 128 bits. The algorithm has influenced later designs, such as the MD5, SHA algorithms.

MD4 was designed to be fast, which meant taking a few risks regarding security. By 1992 weaknesses had been found which led Rivest to produce a strengthened, but slower, version known as MD5. The security of MD4 has been severely compromised. The first full collision attack against MD4 was published in 1995 and several newer attacks have been published since then.

MD4

General	
Designers	Ronald Rivest
First published	October 1990[1]
Series	MD2, MD4, MD5, MD6
Cipher detail	
Digest sizes	128 bits
Block sizes	512 bits
Rounds	3
Best public cryptanalysis	
A collision attack published in 2007 can find collisions for full MD4 in less than 2 hash operations.[2]	

The working of MD-4 algorithm is defined in RFC 1320.

Algorithm:

Let the symbol "+" denote addition of words (i.e., modulo-2^{32} addition). Let (X <<< s) denote the 32-bit value obtained by circularly shifting (rotating) X left by s bit positions. Let ¬X denote the bit-wise complement of X, and let X V Y denote the bit-wise OR of X and Y. Let X ⊕ Y denote the bit-wise XOR of X and Y, and let XY denote the bit-wise AND of X and Y.

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of 8, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

$m_0, m_1 \ldots m_{b-1}$.

The following five steps are performed to compute the message digest of the message.

Step 1. Append padding bits

The message is padded (extended) so that its length (in bits) is congruent to 448, modulo 512. Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended. 512 -64

Step 2. Append Length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.) At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits.

Step 3. Initialize MD Buffer

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Step 4. Process Message in 16-Word Blocks

We first define three auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

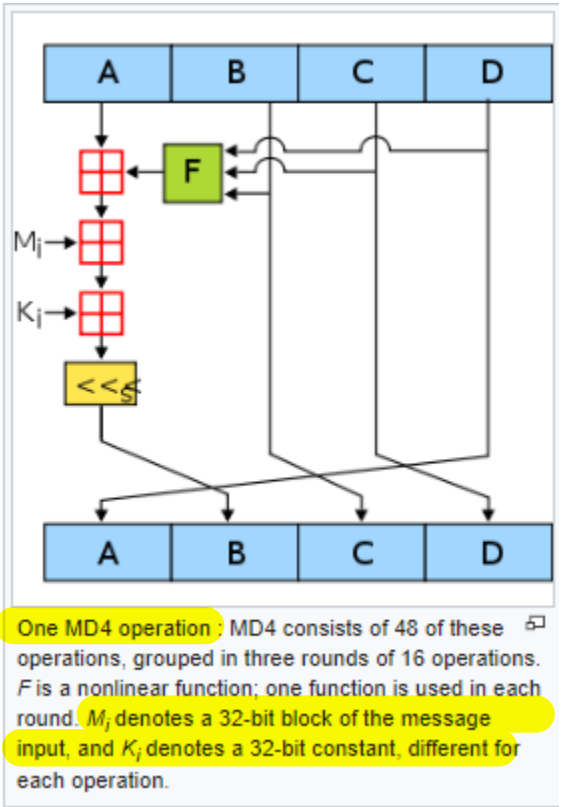
$$
\begin{aligned}
f(X,Y,Z) &= XY \vee (\neg X)Z \\
g(X,Y,Z) &= XY \vee XZ \vee YZ \\
h(X,Y,Z) &= X \oplus Y \oplus Z
\end{aligned}
$$

duita ma or last ma xor

Step 5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

right shift raexa ni tw mathi ko wala so a b ma b c ma...



One MD4 operation : MD4 consists of 48 of these operations, grouped in three rounds of 16 operations. F is a nonlinear function; one function is used in each round. M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each operation.

MD5:

The MD5 message-digest algorithm is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities. It can still be used as a checksum to verify data integrity, but only against unintentional corruption.

The MD5 message digest hashing algorithm processes data in 512-bit blocks, broken down into 16 words composed of 32 bits each. The output from MD5 is a 128-bit message digest value.

MD5 was designed by Ronald Rivest in 1991 to replace an earlier hash function MD4, and was specified in 1992 as RFC 1321.

MD5

General	
Designers	Ronald Rivest
First published	April 1992
Series	MD2, MD4, MD5, MD6
Cipher detail	
Digest sizes	128 bit
Block sizes	512 bit
Structure	Merkle–Damgård construction
Rounds	4[1]

Best public cryptanalysis

A 2013 attack by Xie Tao, Fanbao Liu, and Dengguo Feng breaks MD5 collision resistance in 2^{18} time. This attack runs in less than a second on a regular computer.[2] MD5 is prone to length extension attacks.

Working of MD5 Algorithm:

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

m_0, m_1, ... m_{b-1}

The following five steps are performed to compute the message digest of the message:

Step 1. Append Padding Bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512.

Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

Step 2. Append Length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.)

At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits.

Step 3. Initialize MD Buffer

A four-word buffer (A, B, C, D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Step 4. Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

$$
\begin{aligned}
F(B,C,D) &= (B \wedge C) \vee (\neg B \wedge D) \\
G(B,C,D) &= (B \wedge D) \vee (C \wedge \neg D) \\
H(B,C,D) &= B \oplus C \oplus D \\
I(B,C,D) &= C \oplus (B \vee \neg D)
\end{aligned}
$$

Step 5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

Applications:

MD5 digests have been widely used in the software world to provide some assurance that a transferred file has arrived intact. For example, file servers often provide a pre-computed MD5 (known as md5sum) checksum for the files, so that a user can compare the checksum of the downloaded file to it.

As it is easy to generate MD5 collisions, it is possible for the person who created the file to create a second file with the same checksum, so this technique cannot protect against some forms of malicious tampering.

Historically, MD5 has been used to store a one-way hash of a password but NIST does not include MD5 in their list of recommended hashes for password storage.

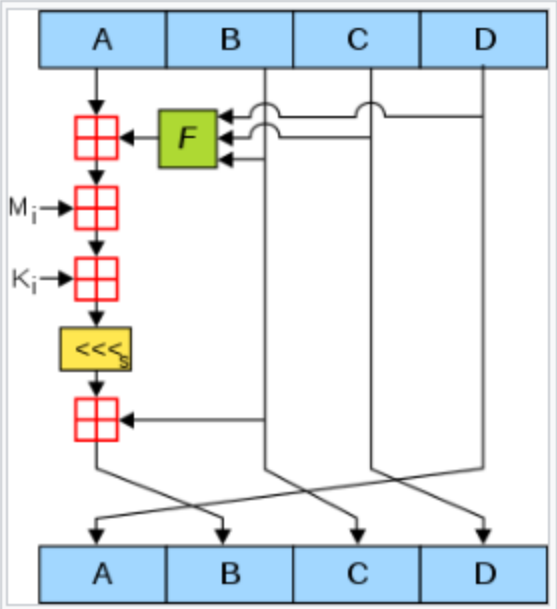


Figure 1. One MD5 operation. MD5 consists of 64 of these operations, grouped in four rounds of 16 operations. F is a nonlinear function; one function is used in each round. M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each operation. $<<<_s$ denotes a left bit rotation by s places; s varies for each operation. $\boxplus$ denotes addition modulo 2^{32}.

Secure Hash Algorithm (SHA):

The Secure Hash Algorithms are a family of cryptographic hash functions published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS), including:

SHA-0: A change applied to the original version of the 160-bit hash function published in 1993 under the name "SHA". It was withdrawn shortly after publication due to an undisclosed "significant flaw" and replaced by the slightly revised version SHA-1.

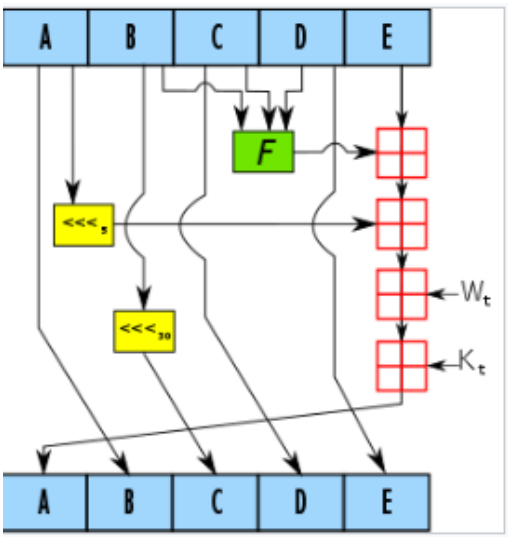
SHA-1: A 160-bit hash function which resembles the earlier MD5 algorithm. This was designed by the National Security Agency (NSA) to be part of the Digital Signature Algorithm. Cryptographic weaknesses were discovered in SHA-1, and the standard was no longer approved for most cryptographic uses after 2010.

SHA-2: A family of two similar hash functions, with different block sizes, known as SHA-256 and SHA-512. They differ in the word size; SHA-256 uses 32-byte words where SHA-512 uses 64-byte words. There are also truncated versions of each standard, known as SHA-224, SHA-384, SHA-512/224 and SHA-512/256. These were also designed by the NSA.

SHA-3: A hash function formerly called Keccak, chosen in 2012 after a public competition among non-NSA designers. It supports the same hash lengths as SHA-2, and its internal structure differs significantly from the rest of the SHA family.

SHA-1:

- In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value known as a message digest – typically rendered as a hexadecimal number, 40 digits long.
- Since 2005 SHA-1 has not been considered secure against well-funded opponents, as of 2010 many organizations have recommended its replacement.
- NIST formally deprecated use of SHA-1 in 2011 and disallowed its use for digital signatures in 2013.
- Since, as of 2020, attacks against SHA-1 are as practical as against MD5.
- It is recommended to remove SHA-1 from products as soon as possible and use instead SHA-256 or SHA-3.
- Replacing SHA-1 is urgent where it's used for signatures.
- All major web browser vendors ceased acceptance of SHA-1 SSL certificates in 2017.
- In February 2017, CWI Amsterdam and Google announced they had performed a collision attack against SHA-1, publishing two dissimilar PDF files which produced the same SHA-1 hash.



One iteration within the SHA-1 compression function:
A, B, C, D and E are 32-bit words of the state;
F is a nonlinear function that varies;
$\lll_n$ denotes a left bit rotation by *n* places;
n varies for each operation;
W_t is the expanded message word of round t;
K_t is the round constant of round t;
$\boxplus$ denotes addition modulo 2^{32}.

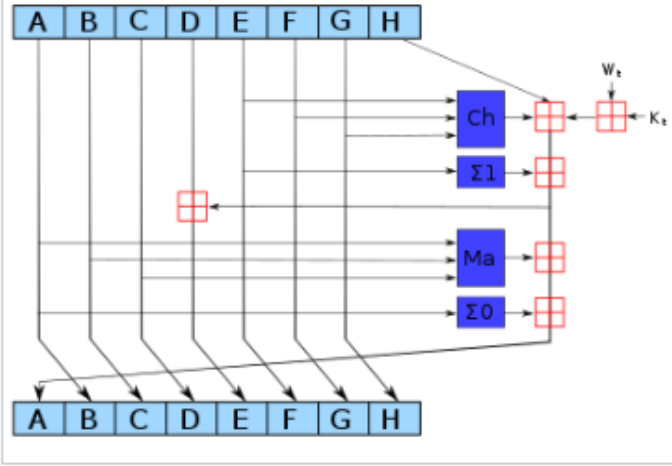
- SHA-1 uses 80 rounds of cryptographic operations to encrypt and secure a data object.
- Some of the protocols that use SHA-1 include:
  - Transport Layer Security (TLS)
  - Secure Sockets Layer (SSL)
  - Pretty Good Privacy (PGP)
  - Secure Shell (SSH)

- Secure/Multipurpose Internet Mail Extensions (S/MIME)
- Internet Protocol Security (IPSec)

SHA 2:

- SHA-2 is a family of hashing algorithms to replace the SHA-1 algorithm. SHA-2 features a higher level of security than its predecessor. It was designed through The National Institute of Standards and Technology (NIST) and the National Security Agency (NSA).
- One of the major benefits of using SHA-2 is that it addresses some weaknesses in the SHA-1 hashing algorithm.
- One of the drawbacks with SHA-2 is that there are some older applications and operating systems that do not support it. Compatibility problems are the main reason why SHA-2 algorithms have not been adopted more rapidly.
- The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.

SHA-256 and SHA-512 are novel hash functions computed with 32-bit and 64-bit words, respectively. They use different shift amounts and additive constants, but their structures are otherwise virtually identical, differing only in the number of rounds (46 and 160 respectively).



One iteration in a SHA-2 family compression function. The blue components perform the following operations:

$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$
$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$
$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$
$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$

The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256.
The red ⊞ is addition modulo 2^{32} for SHA-256, or 2^{64} for SHA-512.

Applications of SHA-2:

- The SHA-2 hash function is implemented in some widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec.
- Used for verifying the transactions. E.g. In cryptocurrency like Bitcoin

Comparison of SHA Parameters:

Algorithm	Message Size	Block Size	Word Size	Message Digest Size
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Note: All sizes are measured in bits.

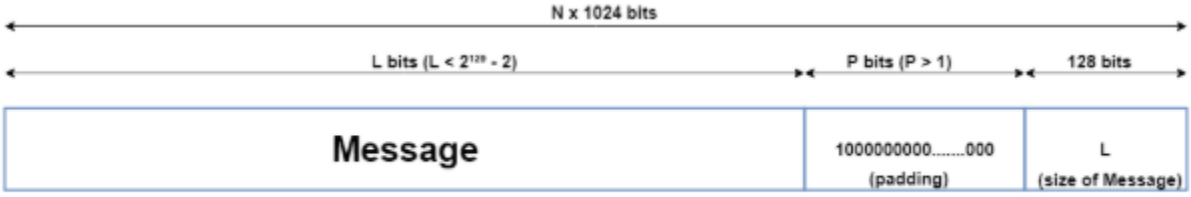
SHA-512:

SHA-512 works in four stages as follows:

- Input formatting
- Hash buffer initialization
- Message Processing
- 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.



Message with padding and size

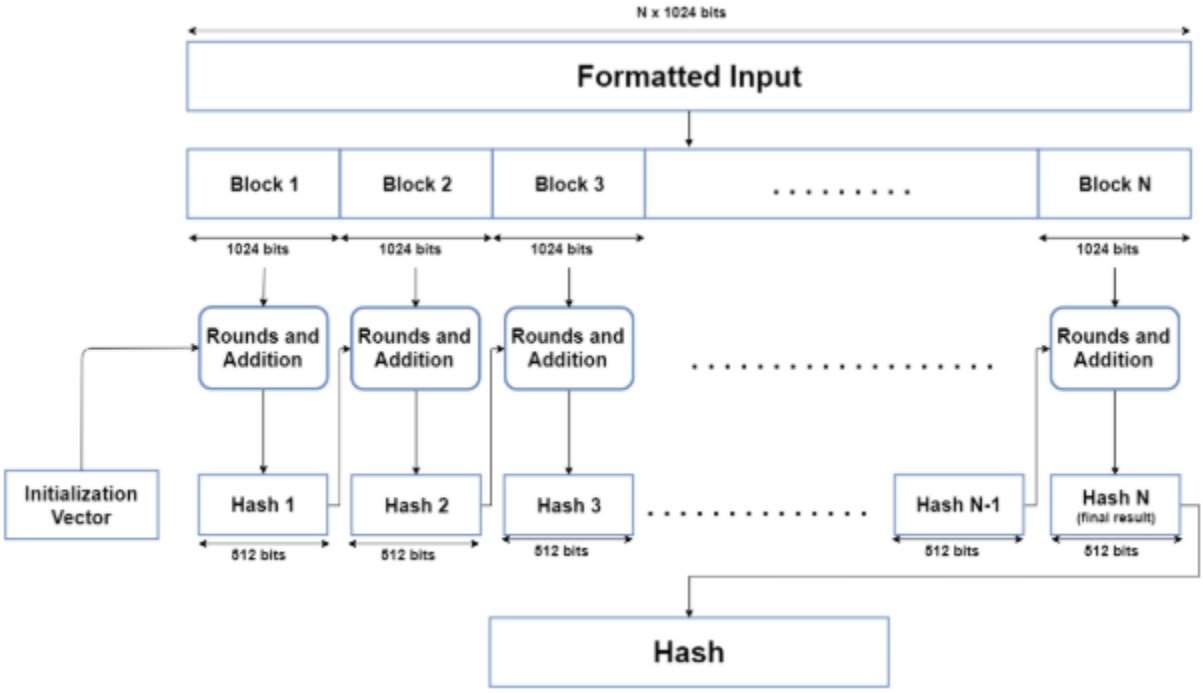
Hash buffer initialization:

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. This problem can be solved by using a default value to be used for the first block in order to start off the process.

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 Addition' operation. The main part of the message processing phase may be considered to be the Rounds. Each round takes 3 things: one Word, the output of the previous Round, and a SHA-512 constant. SHA-512 constants are predetermined values, each of whom is used for each Round in the message processing phase.



Output:

After every block of 1024 bit goes through the message processing phase, i.e. the last iteration of the phase, we get the final 512-bit Hash value of our original message. So, the intermediate results are all used from each block for processing the next block. And when the final 1024-bit block has finished being processed, we have with us the final result of the SHA-512 algorithm for our original message.