S7 B-TECH COMPUTER SCIENCE AND ENGINEERING
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

PREPARED BY
DR SUREKHA MARIAM VARGHESE

MODULE 4

- Module-4 (Message Integrity and Authentication)

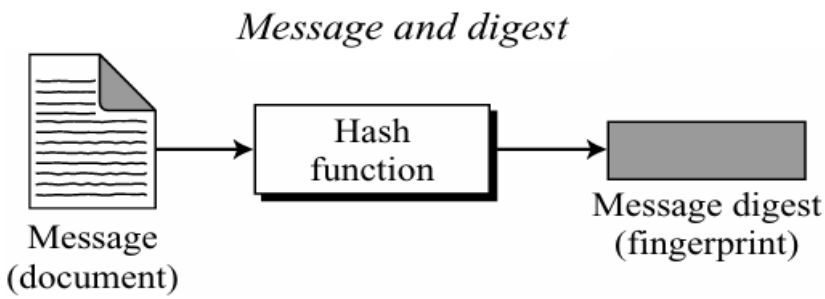
Hash functions – Security requirements, Secure Hash Algorithm (SHA-512). Message Authentication Code (MAC) – Requirements, Uses, Hash-based MAC (HMAC), Cipher-based MAC (CMAC). Digital signatures – Attacks, Forgeries, Requirements, Direct vs Arbitrated digital signatures, RSA digital signature, ElGamal digital signature, Digital Signature Standard (DSS).

MESSAGE INTEGRITY

- The cryptography systems that we have studied so far provide secrecy, or confidentiality, but not integrity
- However, there are occasions where we may not even need secrecy but instead must have integrity
- For example, Alice may write a will to distribute her estate upon her death. The will does not need to be encrypted. After her death, anyone can examine the will. The integrity of the will, however, needs to be preserved. Alice does not want the contents of the will to be changed.

MESSAGE AND MESSAGE DIGEST

- The electronic equivalent of the document and fingerprint
- To preserve the integrity of a message, the message is passed through an algorithm called a cryptographic hash function. The function creates a compressed image of the message that can be used like a fingerprint.



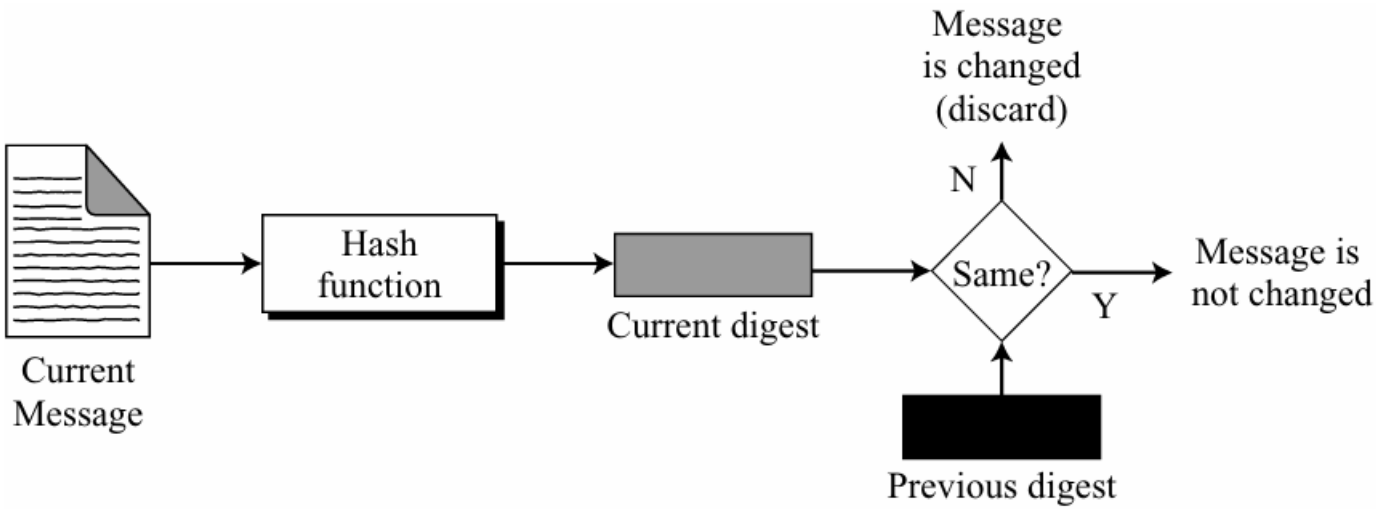
- Difference
- Document and fingerprint are physically linked together
- The message and message digest can be unlinked (or sent) separately, and, most importantly, the message digest needs to be safe from change

DOCUMENT AND FINGERPRINT

- One way to preserve the integrity of a document is through the use of a fingerprint. If Alice needs to be sure that the contents of her document will not be changed, she can put her fingerprint at the bottom of the document. Eve cannot modify the contents of this document or create a false document because she cannot forge Alice's fingerprint
- To ensure that the document has not been changed, Alice's fingerprint on the document can be compared to Alice's fingerprint on file. If they are not the same, the document is not from Alice

CHECKING INTEGRITY

- To check the integrity of a message, or document, we run the cryptographic hash function again and compare the new message digest with the previous one. If both are the same, we are sure that the original message has not been changed

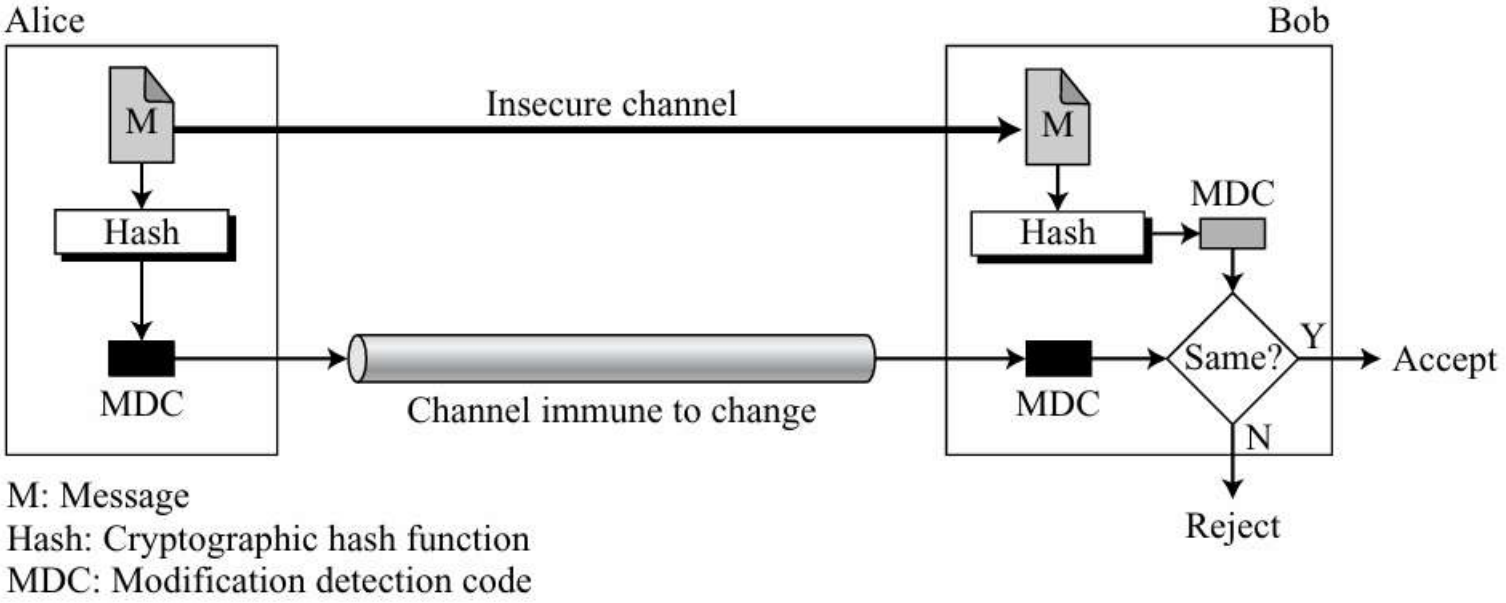


MESSAGE AUTHENTICATION

- A message digest guarantees the integrity of a message. It guarantees that the message has not been changed.
- A message digest, however, does not authenticate the sender of the message. When Alice sends a message to Bob, Bob needs to know if the message is coming from Alice. To provide message authentication, Alice needs to provide proof that it is Alice sending the message and not an impostor. A message digest per se cannot provide such a proof. The digest created by a cryptographic hash function is normally called a modification detection code (MDC). The code can detect any modification in the message. What we need for message authentication (data origin authentication) is a message authentication code

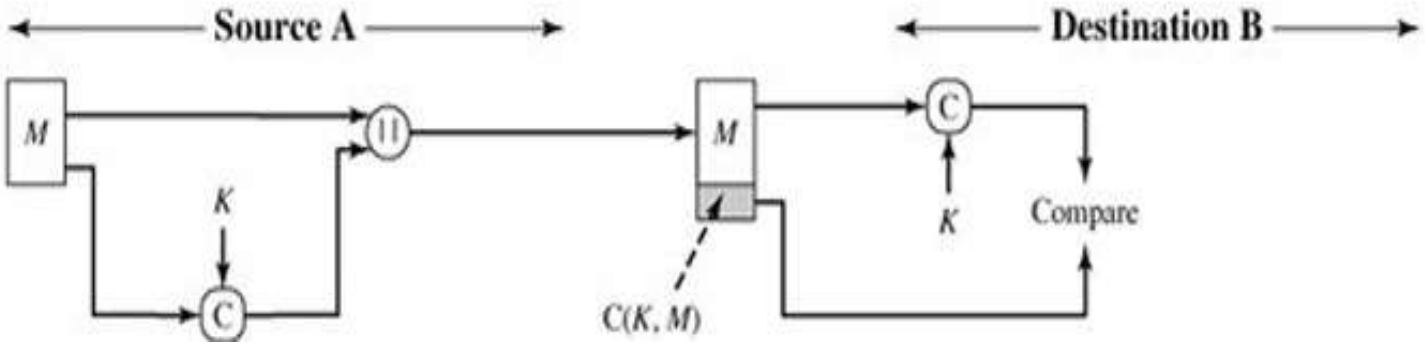
MESSAGE DETECTION CODE

- If Alice needs to send a message to Bob and be sure that the message will not change during transmission, Alice can create a message digest, MDC, and send both the message and the MDC to Bob. Bob can create a new MDC from the message and compare the received MDC and the new MDC. If they are the same, the message has not been changed



MESSAGE AUTHENTICATION CODE

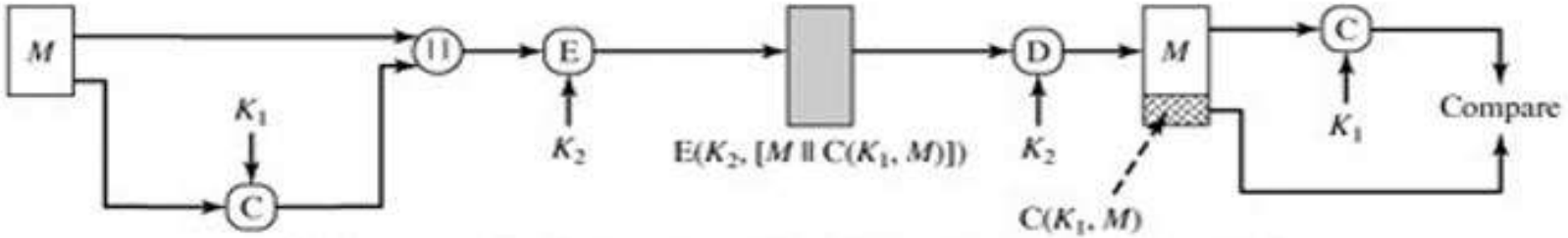
- MAC= C(K, M), where M = input message C = MAC function K = shared secret key MAC = message authentication code



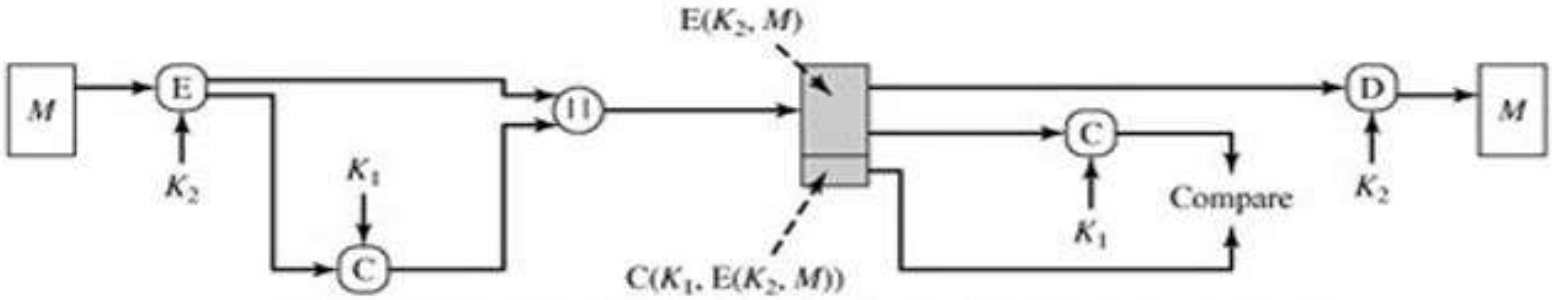
- The process provides authentication but not confidentiality, because the message as a whole is transmitted in the clear

PROVIDING CONFIDENTIALITY

Confidentiality can be provided by performing message encryption either after (b) or before (c) In both these cases, two separate keys are needed, each of which is shared by the sender and the receiver



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

HASH FUNCTIONS

- A hash value h is generated by a function H of the form h = H(M) where M is a variable-length message and H(M) is the fixed-length hash value
- Hash value is appended to the message at the source at a time when the message is assumed or known to be correct.
- Receiver authenticates that message by recomputing hash value
- Not considered to be secret -required to protect hash value
- Purpose is to produce a "fingerprint" of a file, message, or other block of data

Requirements for a Hash Function

- H can be applied to a block of data of any size
- H produces a fixed-length output
- H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical
- For any given value h, it is computationally infeasible to find x such that H(x) = h- referred to as the one-way property
- For any given block x, it is computationally infeasible to find $y \neq x$ such that H(y) = H(x). - referred to as weak collision resistance
- It is computationally infeasible to find any pair (x, y) such tha H(x) = H(y) - referred to as strong collision resistance

SIMPLE HASH FUNCTIONS

- general principles
- Input (message/file/..) viewed as a sequence of n-bit blocks
- Input is processed one block at a time in an iterative fashion to produce an n-bit hash function
- One simplest is bit-by-bit exclusive-OR(XOR) of every block
- Expressed as

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

where C_i = ith bit of the hash code, $1 \leq i \leq n$

m = number of n-bit blocks in the input

b_{ij} = ith bit in jth block

$\oplus$ = XOR operation

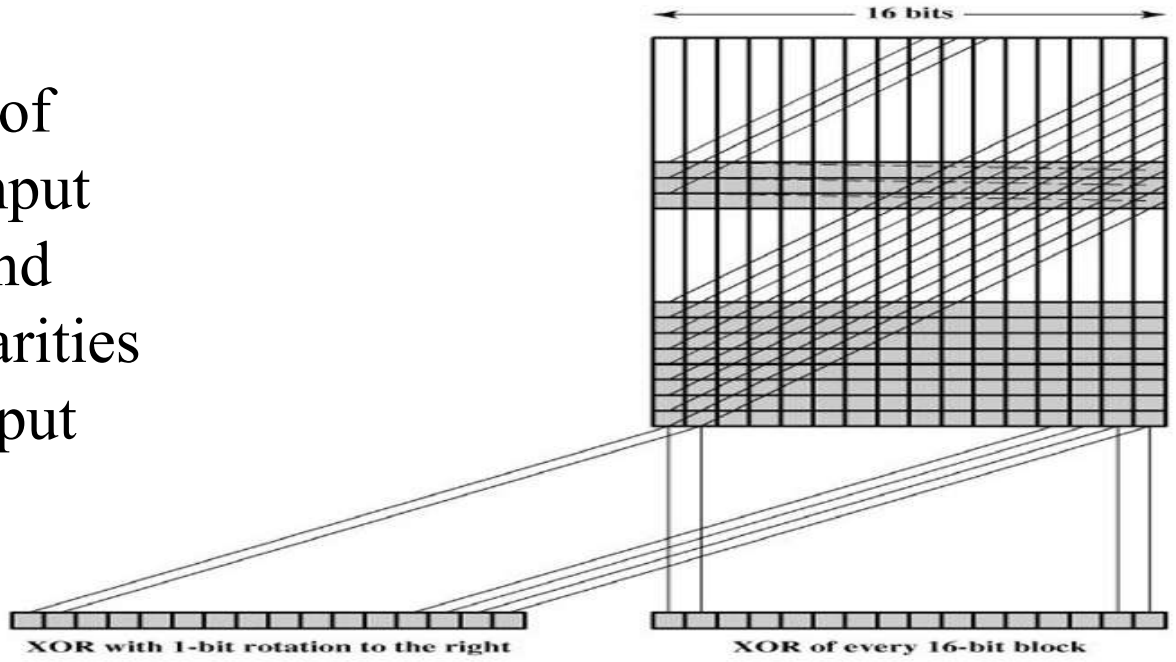
IMPROVEMENT

- It is reasonably effective for random data as a data integrity check. Each n-bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is 2^n. With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2^{128}, the hash function on this type of data has an effectiveness of 2^{112}.
- A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed

ROTATED XOR

- Initially set the n-bit hash value to zero
- Process each successive n-bit block of data as follows:
  - Rotate the current hash value to the left by one bit
  - XOR the block into the hash value

This has the effect of "randomizing" the input more completely and overcoming any regularities that appear in the input

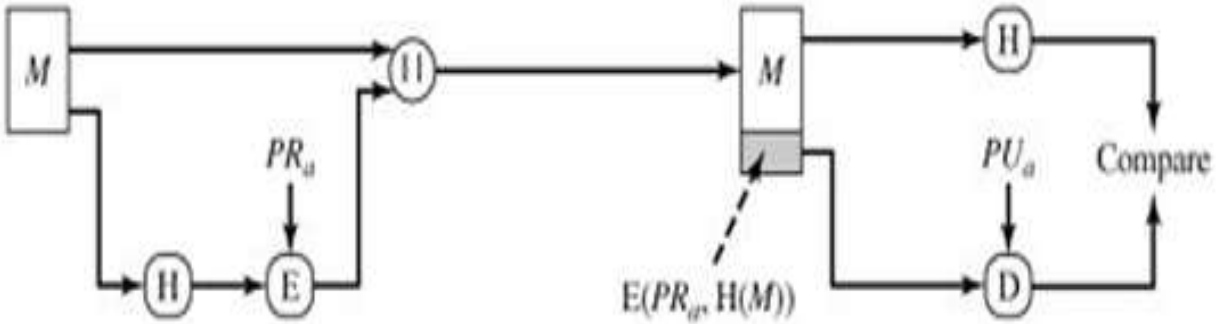


BIRTHDAY ATTACKS

- A **birthday attack** is a type of cryptographic attack that exploits the mathematics behind the birthday problem in probability theory. Here's a breakdown of what it entails:
- **Birthday Problem**
- The birthday problem states that in a group of 23 people, there's a better than even chance that two people will share the same birthday. This counterintuitive result arises because the number of possible pairs of people is much larger than the number of days in a year.

A STRATEGY FOR ATTACK

- The source, A, is prepared to "sign" a message by appending the appropriate m-bit hash code and encrypting that hash code with A's private key



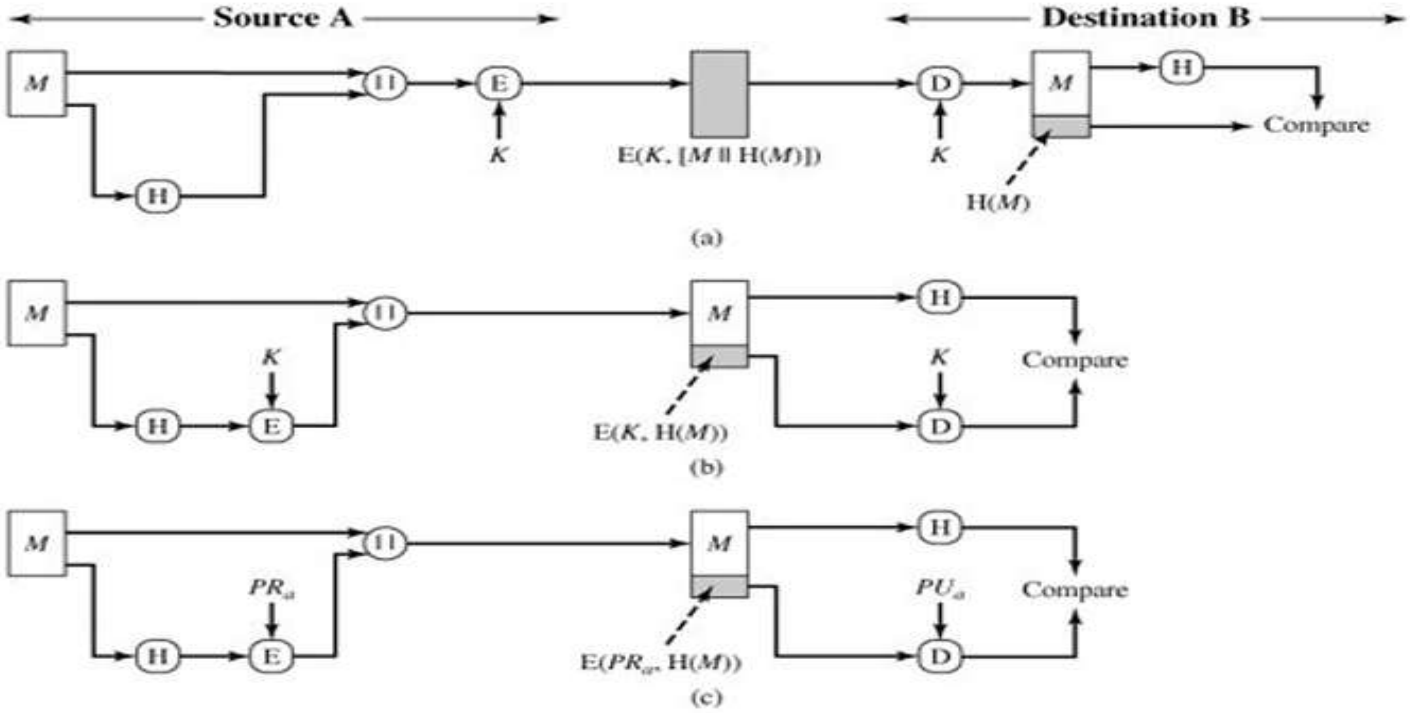
- Opponent generates $2^{m/2}$ variations on the message
  - Prepares an equal number of messages, all of which are variations on the fraudulent message to be substituted for the real one

A STRATEGY CONTD..

- The two sets of messages are compared to find a pair of messages that produces the same hash code. The probability of success, by the birthday paradox, is greater than 0.5. If no match is found, additional valid and fraudulent messages are generated until a match is made
- The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known

BASIC USES OF HASH FUNCTION

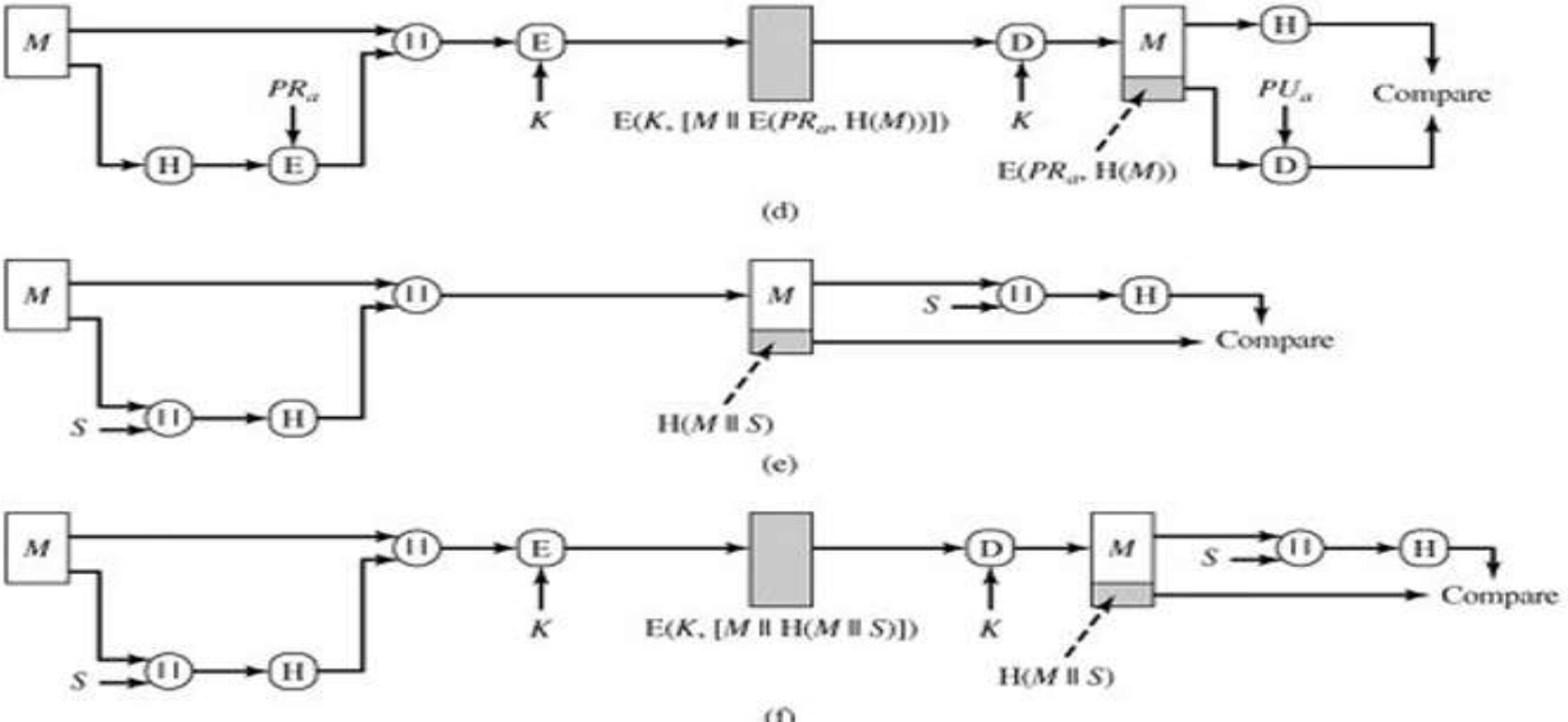
- M+MAC encrypted using symmetric encryption
- Only MAC encrypted, using symmetric – reduces burden
- Only MAC -public-key encryption with sender's private key



BASIC USES OF HASH FUNCTION CONTD..

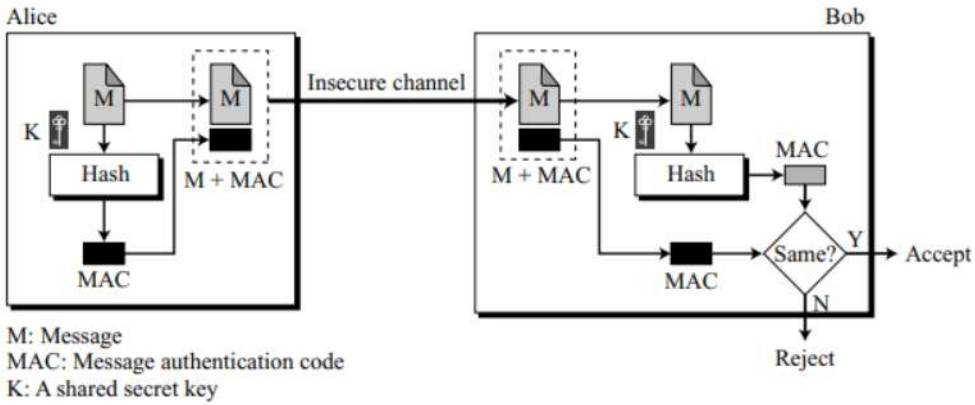
private-key encrypted M+MAC –further symmetric encrypted

A computes the hash value over the concatenation of M and S(a shared secret) and appends the resulting hash value to M



MESSAGE AUTHENTICATION CODE-MAC

- To ensure the integrity of the message and the data origin
  - authentication that Alice is originator of message, not somebody else
- Need to change a modification detection code (MDC)
- MAC includes a secret between Alice and Bob
  - a secret key that Eve does not possess

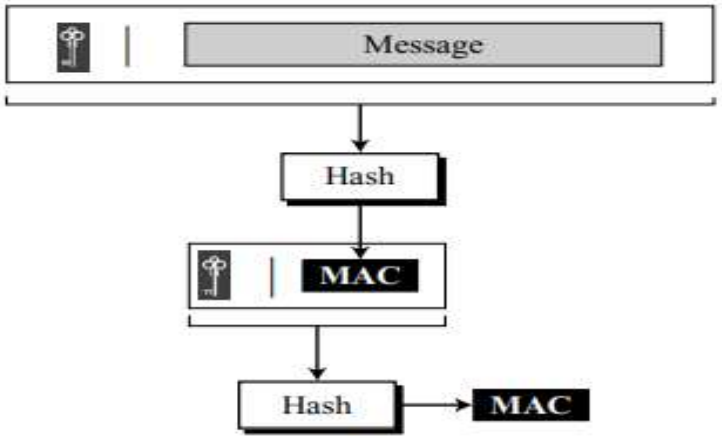


SECURITY OF A MAC

- 3 possible methods to forge message without secret key if Eve intercepted the message M and the digest h(K|M)
- Exhaustive Search-If key size is small, prepend all possible keys at the beginning of the message and make a digest of the (K|M) to find the digest equal to the one intercepted
  - Once key is known and message can be replaced with forged
- Pre-image Attack- If key is large, She uses the algorithm until she finds X such that h(X) is equal toMAC intercepted
- Given some pairs of messages and their MACs, Eve can manipulate them to come up with a new message and its MAC

NESTED MAC

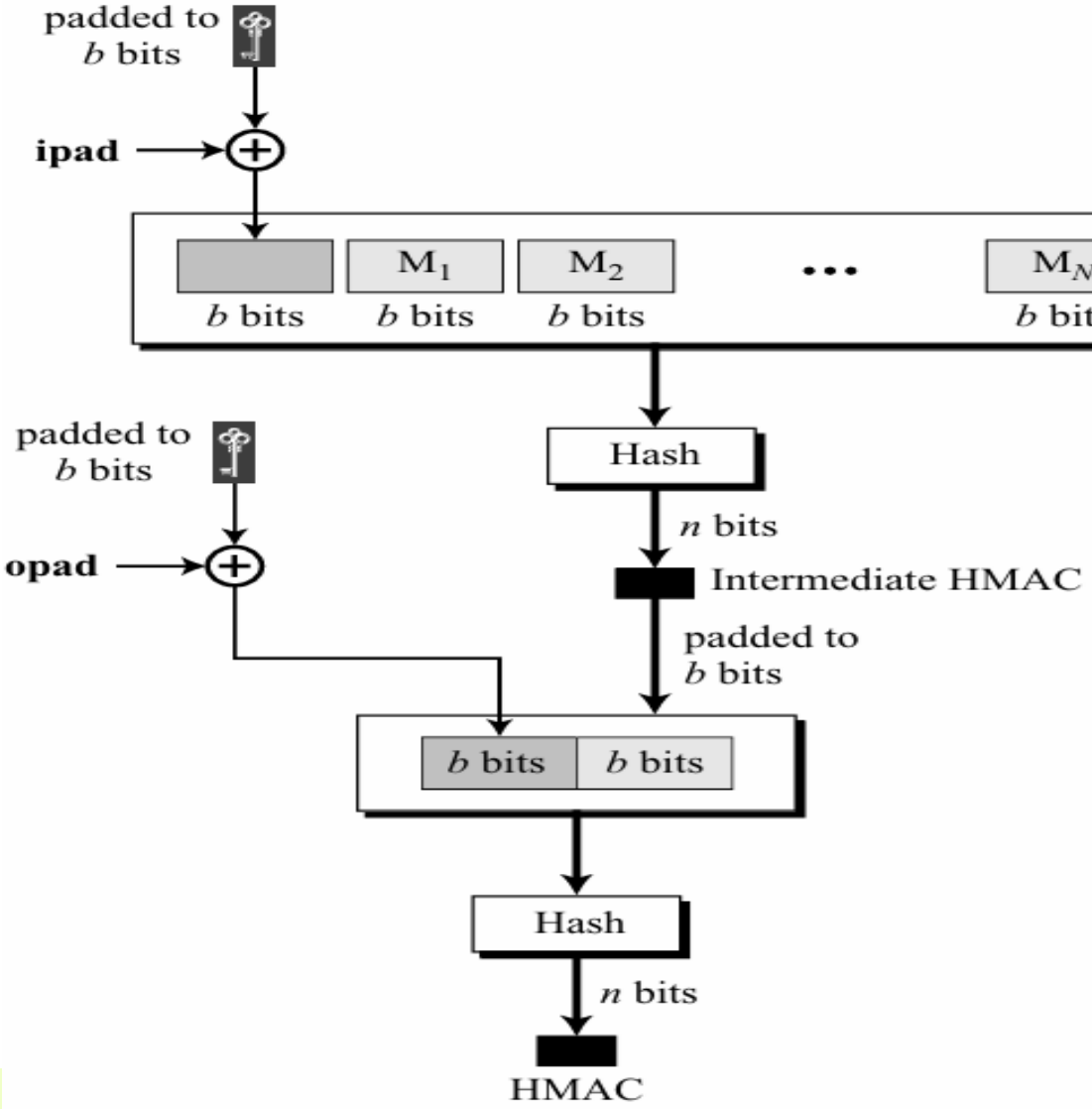
- To improve the security of a MAC, nested MACs were designed in which hashing is done in two steps. In the first step, the key is concatenated with the message and is hashed to create an intermediate digest. In the second step, the key is concatenated with the intermediate digest to create the final digest. Figure shows the general idea



HASHED MAC - HMAC

- More complex implementation than simplified nested MAC
- More widely adopted
- Includes additional features, such as padding
  - Two types of padding
  - Inner and Outer Padding
- By XORing the key with ipad and opad, HMAC ensures that the inner and outer hashes are distinct, even if the same key is used
- Ensures that key is processed in a way that provide high level of security by preventing certain types of attacks

HMAC WORKING



Process Summary:

Compute:

Inner hash-

H(K ⊕ ipad || message)

Outer hash- H(K ⊕ opad || inner_hash)

Where H is the hash function, K is the key, and || denotes concatenation

INNER/INTERMEDIATE MAC

- 1. The message is divided into N blocks, each of b bits.
- 2. The secret key is left-padded with 0's to create a b-bit key secret key size> n bits, n - size of HMAC
- 3. The result of step 2 is exclusive-ored with a constant calle ipad (input pad) tocreate a b-bit block. The value of ipad is the b/8 repetition of the sequence 00110110 (36 in hexadec)
- 4. The resulting block is prepended to the N-block message. The result is N + 1 blocks.
- 5. The result of step 4 is hashed to create an n-bit digest. We call the digest the intermediate HMAC

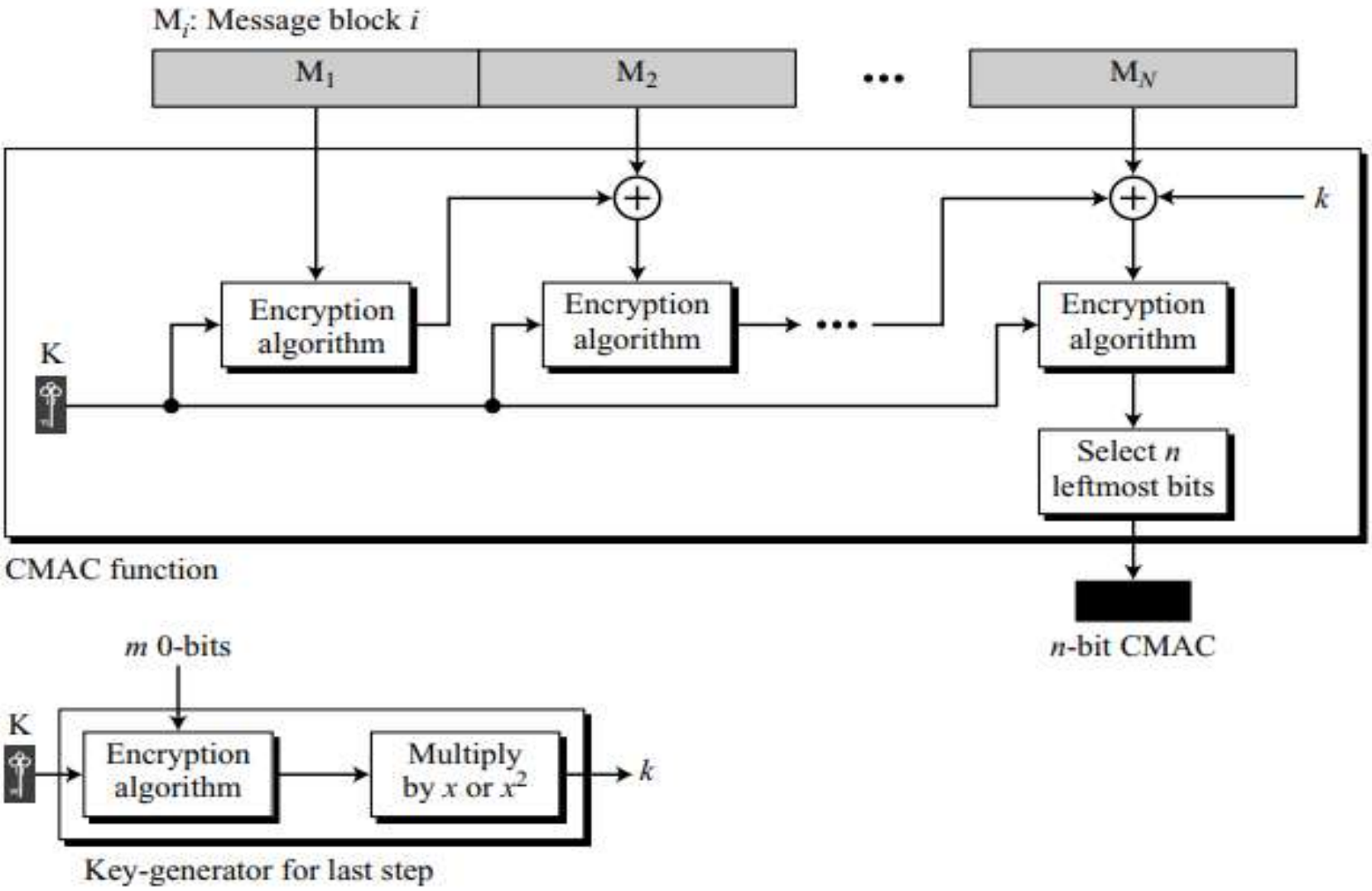
FINAL HMAC

- 6. The intermediate n-bit HMAC is left padded with 0s to make a b-bit block
- 7. Steps 2 and 3 are repeated by a different constant opad (output pad). The value of opad is the b/8 repetition of the sequence 01011100 (5C in hexadecimal).
- 8. The result of step 7 is prepended to the block of step 6.
  9. The result of step 8 is hashed with the same hashing algorithm to create the final n-bit HMAC.

CMAC Concept

- CMAC, or CBCMAC. The method is similar to the cipher block chaining (CBC) for symmetric-key encipherment
- Cipher based mac- calculates mac using block cipher coupled with a secret key
- However, the idea here is not to create N blocks of ciphertext from N blocks of plaintext
- The idea is to create one block of MAC from N blocks of plaintext using a symmetric-key cipher N times.

CMAC WORKING



CMAC Initial Steps

- Message is divided into N blocks, each m bits long
- The size of the CMAC is n bits. If the last block is not m bits, it is padded with a 1-bit followed by enough 0-bits to make it m bits
- The first block of the message is encrypted with the symmetric key to create an m-bit block of encrypted data
- This block is XORed with the next block and the result is encrypted again to create a new m-bit block. The process continues until the last block of the message is encrypted. The n leftmost bit from the last block is the CMAC.

CMAC Final Steps Involved

- In addition to the symmetric key, K, CMAC also uses another key, k, which is applied only at the last step
- This key is derived from the encryption algorithm with plaintext of m 0-bits using the cipher key, K. The result is then multiplied by x if no padding is applied and multiplied by x^2 if padding is applied.
- Different from the CBC used for confidentiality

SECURE HASH ALGORITHM- 512

- SHA-512 (512-bit) is a cryptographic hash function from the SHA-2 family
- Takes input data of any length and produces a fixed-size output—a 512-bit (64-byte) hexadecimal hash value
- It's like a digital fingerprint for data
- Generates a unique hash- Even a tiny change in the input results in a completely different hash. This property makes it useful for ensuring data integrity, verifying message authenticity, and protecting against tampering
- It is document's personal digital detective!

APPLICATIONS

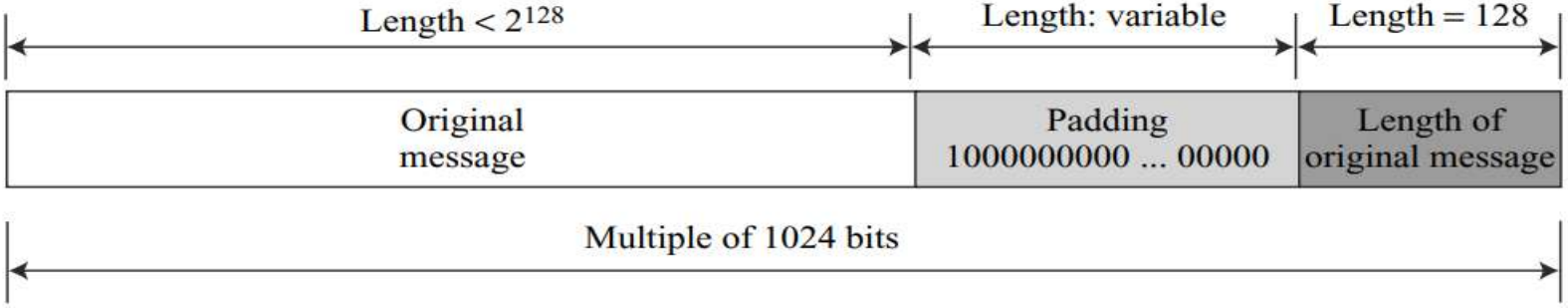
- Secure Password Storage and Verification: SHA-512 helps securely store and verify passwords
  - When login to a website, password is hashed using SHA-512, making it resistant to reverse engineering
- Data Integrity and authenticity
  - Widely used in a variety of security applications and protocols, including TLS and SSL, PGP, SSH, Ipsec
  - Part of the cryptographic magic behind blockchain technology

SHA-512 CORE STRUCTURE

- SHA-512's core structure consists of:
- Message Padding and Preprocessing
  - Input messages are padded to a multiple of 1024 bits
- Message Block Formation and Processing:
  - The padded message is divided into 1024-bit blocks
  - Each block is processed through a series of rounds (80 rounds for SHA-512)
  - Initial Hash Values: Introduce the eight initial hash values (H0, H1, ..., H7) used in SHA-512.

MESSAGE PADDING

- Processes messages in 1024-bit (128-byte) blocks. If the input message isn't a multiple of 1024 bits, it needs to be padded to reach the required length. The padding process involves the following steps:
- Append a single '1' bit to the end of the message.
- Append '0' bits until the length of the message is 1024 bits fewer than a multiple of 1024.
- Append a 128-bit representation of the original message length (in bits) to the end of the padded message.

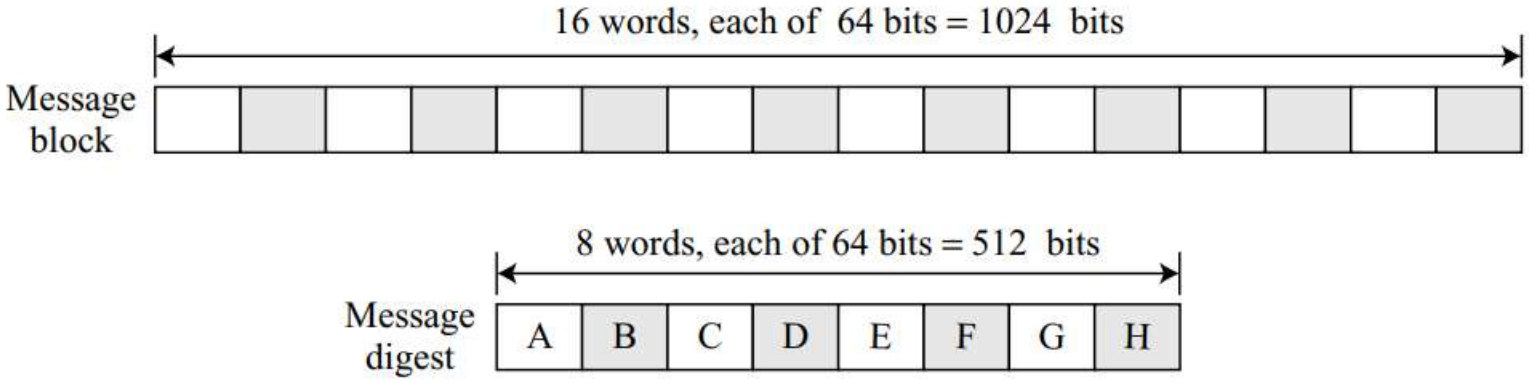


PROCESSING

- Parse the message into 1024-bit blocks: The padded message is divided into 1024-bit blocks.
- Initialize the hash values: Uses 8 initial hash values, derived from the square roots of the first 8 prime numbers
- Compression function: Each block is processed through a series of 80 rounds of operations, including bitwise operations (AND, XOR, NOT), modular additions, and logical functions
- Ensures that the message is properly formatted and ready for the hashing algorithm to produce a fixed-size output, which serves as a unique digital fingerprint of the original message

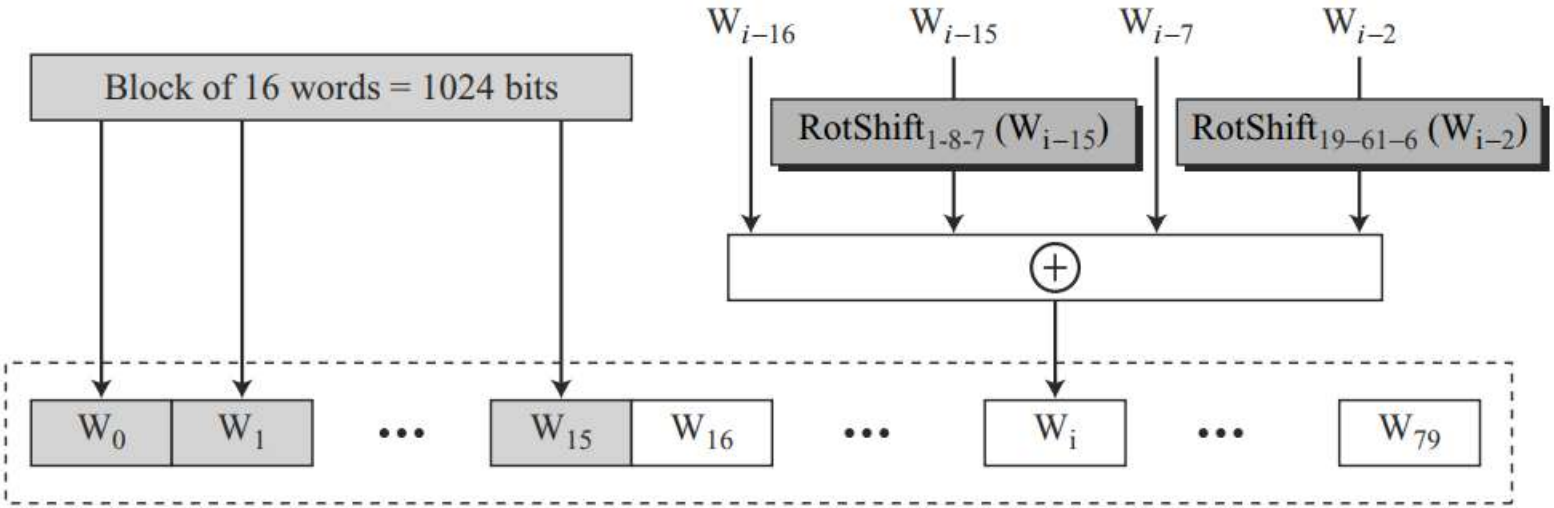
WORD ORIENTED

- SHA-512 considers blocks and digest as words; of 64 bits
- After the padding, each block is of 16, 64-bit words
- Message digest is made of 64-bit words, 8 words
  - named as A, B,C, D, E, F, G, and H



WORD EXPANSION

- Before processing, each message block must be expanded
- Need 80 words in processing phase, so 16 to 80 W0 to W79
- First 16 direct words; rest of words after specific operation



RotShift$_{1\text{-}m\text{-}n}$ (x): $\boldsymbol{RotR_l(x)} \oplus \boldsymbol{RotR_m(x)} \oplus \boldsymbol{ShL_n(x)}$

$\boldsymbol{RotR_i(x)}$: Right-rotation of the argument x by i bits
$\boldsymbol{ShL_i(x)}$: Shift-left of the argument x by i bits and padding the left by 0's.

EXAMPLE SHOWING WORD 60

- Each word in the range W16 to W79 is made from four previously-made words. W60 is made as

$$W_{60} = W_{44} \oplus \text{RotShift}_{1\text{-}8\text{-}7}(W_{45}) \oplus W_{53} \oplus \text{RotShift}_{19\text{-}61\text{-}6}(W_{58})$$

MESSAGE DIGEST INITIALIZATION

- Uses 8 constants for message digest initialization, A0 to H0

Values of constants in message digest initialization of SHA-512

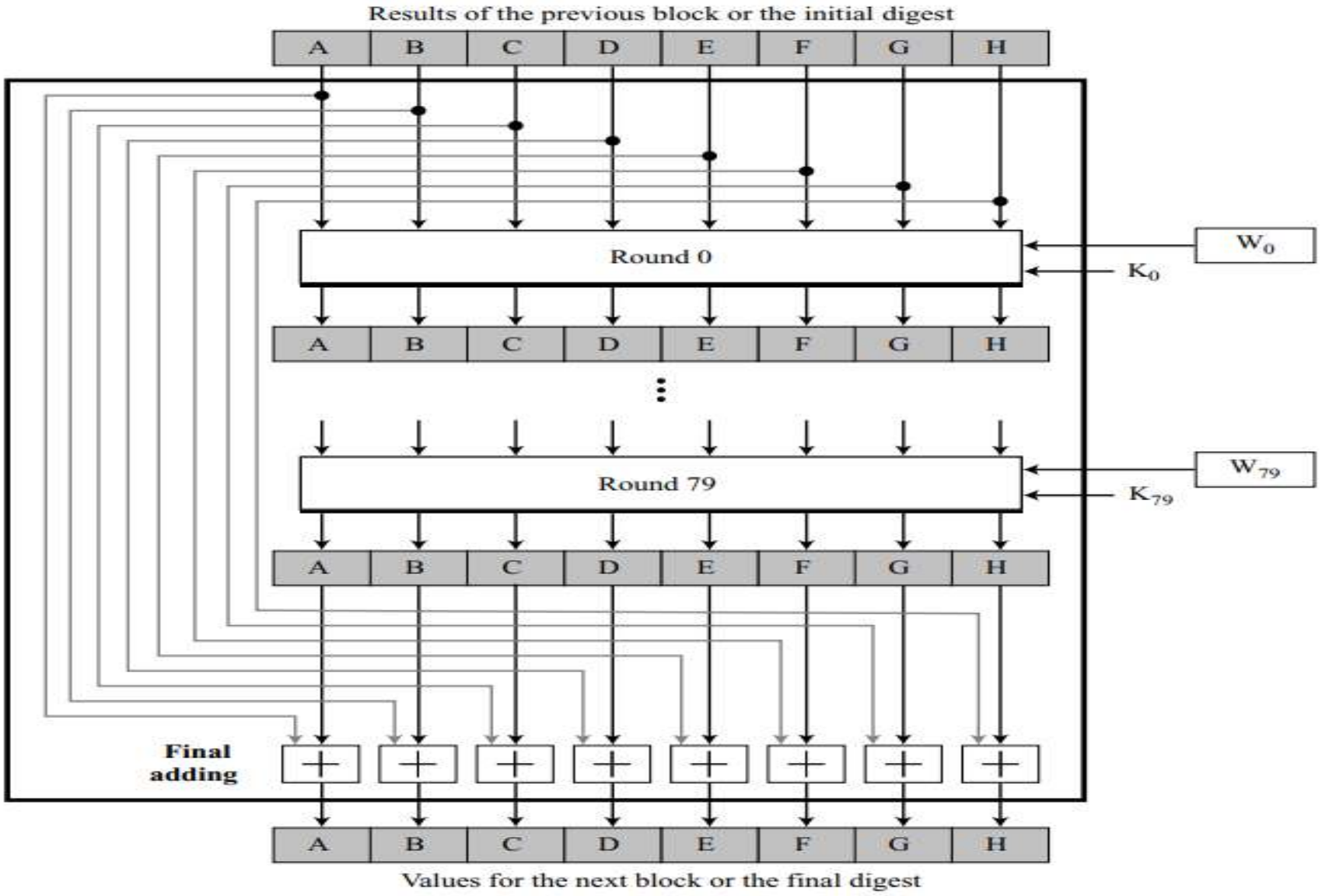
Buffer	*Value (in hexadecimal)*	*Buffer*	*Value (in hexadecimal)*
A_0	6A09E667F3BCC908	E_0	510E527FADE682D1
B_0	BB67AE8584CAA73B	F_0	9B05688C2B3E6C1F
C_0	3C6EF372EF94F82B	G_0	1F83D9ABFB41BD6B
D_0	A54FE53A5F1D36F1	H_0	5BE0CD19137E2179

- Calculated from first 8 primes 2, 3, 5, 7, 11, 13, 17, 19
- Each value is fractional part of the square root of the corresponding to first 64 bits of prime number in binary
- 8^{th} prime is 19, with square root $(19)^{1/2} = 4.35889894354$
- Converting to binary $(100.0101\ 1011\ 1110 \ldots 1001)_2 \rightarrow (4.5BE0CD19137E2179)_{16}$
- Takes first 64 bits, Keeps fraction part, $(5BE0CD19137E2179)_{16}$, as unsigned integer

COMPRESSION FUNCTION

- Uses a 512-bit (eight 64-bit words) message digest from a multiple-block message where each block is 1024 bits
- Processing of each block in SHA 512 involves 80 rounds
- In each round, contents of 8 previous buffers, one word from expanded block (Wi), and one 64-bit constant (Ki) are mixed together and then operated on to create a new set of 8 buffers
- At the beginning, values of the eight buffers are saved into eight temporary variables
- At the end of the processing (after step 79), these values are added to the values created from step 79. We call this last operation the final adding

COMPRESSION FUNCTION- DIAGRAM

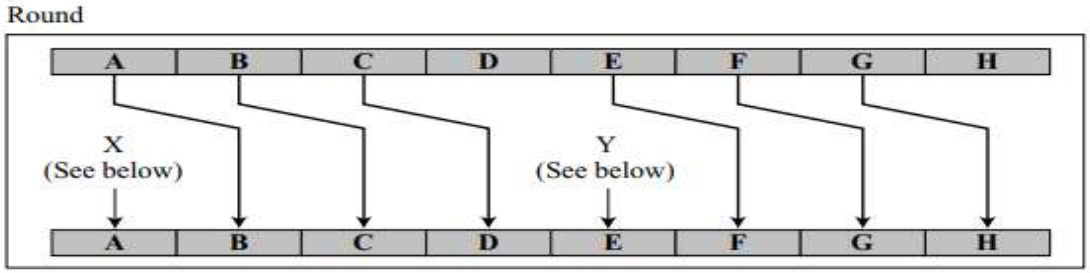


STRUCTURE OF EACH ROUND

- In each round, values for 8 64-bit buffers are created from the values of buffers in the previous round
- Six buffers are copies of one of buffers in previous round

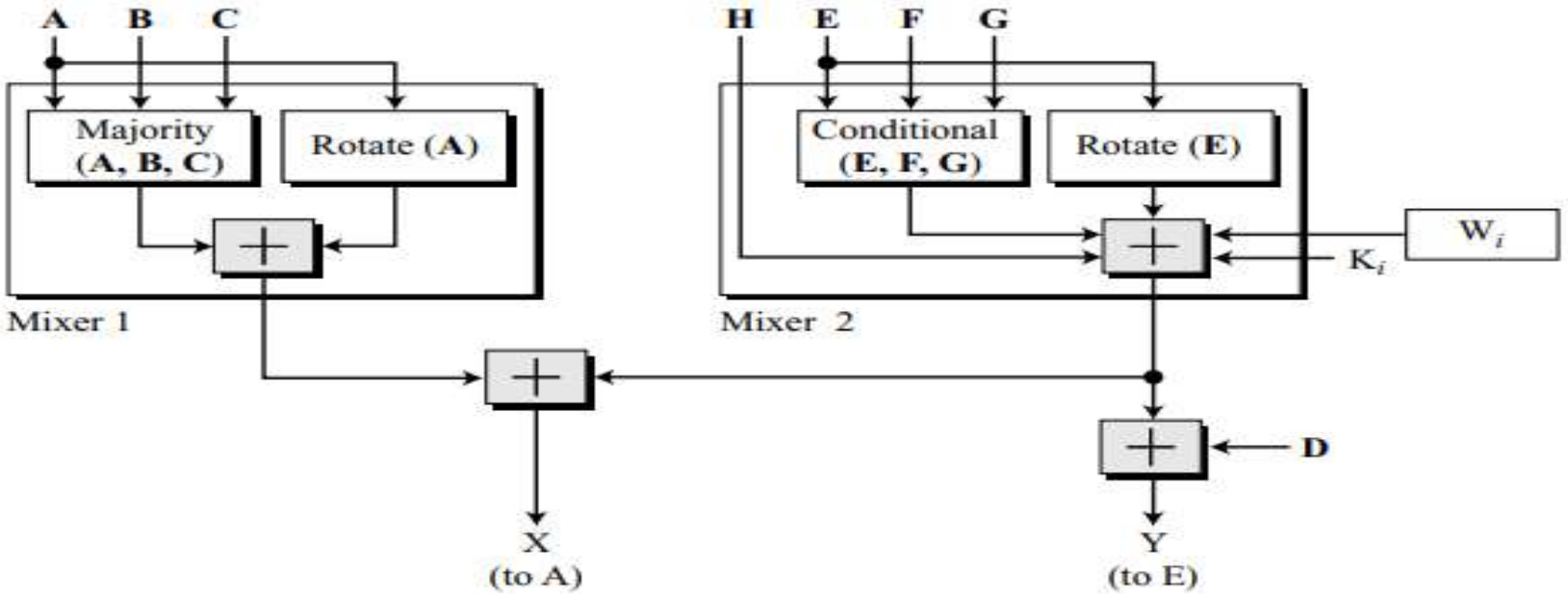
$$A \rightarrow B \quad B \rightarrow C \quad C \rightarrow D \quad E \rightarrow F \quad F \rightarrow G \quad G \rightarrow H$$

- Two of the new buffers, A and E, receive their inputs from some complex functions that involve some of the previous buffers, the corresponding word for this round (Wi), and the corresponding constant for this round (Ki)



DERIVATION FOR NEW BUFFERS

- 2 mixers, majority, rotate and conditional operations



Majority (x, y, z)

$(x \text{ AND } y) \oplus (y \text{ AND } z) \oplus (z \text{ AND } x)$

Conditional (x, y, z)

$(x \text{ AND } y) \oplus (\text{NOT } x \text{ AND } z)$

Rotate (x)

$RotR_{28}(x) \oplus RotR_{34}(x) \oplus RotR_{39}(x)$

$\boxplus$ addition modulo 2^{64}

$RotR_i(x)$: Right-rotation of the argument x by i bits

MAJORITY AND CONDITIONAL FUNCTIONS

- Majority function
  - Bitwise function - takes three corresponding bits in three buffers (A, B, and C) and calculates

    $(A_j \text{ AND } B_j) \oplus (B_j \text{ AND } C_j) \oplus (C_j \text{ AND } A_j)$

    (Aj AND Bj) ⊕ (BjAND Cj) ⊕ (Cj AND Aj)
  - Resulting bit is the majority of 3 bits. If 2 or 3 bits are 1's, the resulting bit is 1; otherwise it is 0
- Conditional function
  - Bitwise function - takes three corresponding bits in three buffers (E, F, and G) and calculates
  - (EjAND Fj) ⊕ (NOT EjAND Gj)
  - Resulting bit is the logic "If Ej then Fj; else Gj".

ROTATION AND ADDITION FUNCTIONS

- Rotate function - right-rotates the three instances of the same buffer (A or E) and applies exclusive-or operation on results

  Rotate (A): $\text{RotR}_{28}(A) \oplus \text{RotR}_{34}(A) \oplus \text{RotR}_{29}(A)$

  Rotate (E): $\text{RotR}_{28}(E) \oplus \text{RotR}_{34}(E) \oplus \text{RotR}_{29}(E)$

- Right-rotation function RotRi(x)- same as in word expansion
  - Right-rotates its argument i bits; a circular shift right
- Addition operator used in the process is addition modulo 2^{64}
  - So result of adding 2 or more buffers is always a 64-bit word

CONSTANT VALUES

- 80 constants, K0 to K79, calculated from first 80 prime numbers (2, 3,…, 409)
- Each value is the fraction part of the cubic root of the corresponding prime number after converting it to binary and keeping only first 64 bits
- For example, 80th prime is 409, with cubic root $(409)^{1/3} = 7.42291412044$ and converting to binary with only 64 bits of fraction part $(111.0110\ 1100\ 0100\ 0100\ \ldots\ 0111)_2 \rightarrow (7.6C44198C4A475817)_{16}$
- Takes first 64 bits, Keeps the fraction part, $(6C44198C4A475817)_{16}$, as an unsigned integer

EIGHTY CONSTANTS IN SHA 512

428A2F98D728AE22	7137449123EF65CD	B5C0FBCFEC4D3B2F	E9B5DBA58189DBBC
3956C25BF348B538	59F111F1B605D019	923F82A4AF194F9B	AB1C5ED5DA6D8118
D807AA98A3030242	12835B0145706FBE	243185BE4EE4B28C	550C7DC3D5FFB4E2
72BE5D74F27B896F	80DEB1FE3B1696B1	9BDC06A725C71235	C19BF174CF692694
E49B69C19EF14AD2	EFBE4786384F25E3	0FC19DC68B8CD5B5	240CA1CC77AC9C65
2DE92C6F592B0275	4A7484AA6EA6E483	5CB0A9DCBD41FBD4	76F988DA831153B5
983E5152EE66DFAB	A831C66D2DB43210	B00327C898FB213F	BF597FC7BEEF0EE4
C6E00BF33DA88FC2	D5A79147930AA725	06CA6351E003826F	142929670A0E6E70
27B70A8546D22FFC	2E1B21385C26C926	4D2C6DFC5AC42AED	53380D139D95B3DF
650A73548BAF63DE	766A0ABB3C77B2A8	81C2C92E47EDAEE6	92722C851482353B
A2BFE8A14CF10364	A81A664BBC423001	C24B8B70D0F89791	C76C51A30654BE30
D192E819D6EF5218	D69906245565A910	F40E35855771202A	106AA07032BBD1B8
19A4C116B8D2D0C8	1E376C085141AB53	2748774CDF8EEB99	34B0BCB5E19B48A8
391C0CB3C5C95A63	4ED8AA4AE3418ACB	5B9CCA4F7763E373	682E6FF3D6B2B8A3
748F82EE5DEFB2FC	78A5636F43172F60	84C87814A1F0AB72	8CC702081A6439EC
90BEFFFA23631E28	A4506CEBDE82BDE9	BEF9A3F7B2C67915	C67178F2E372532B
CA273ECEEA26619C	D186B8C721C0C207	EADA7DD6CDE0EB1E	F57D4F7FEE6ED178
06F067AA72176FBA	0A637DC5A2C898A6	113F9804BEF90DAE	1B710B35131C471B
28DB77F523047D84	32CAAB7B40C72493	3C9EBE0A15C9BEBC	431D67C49C100D4C
4CC5D4BECB3E42B6	4597F299CFC657E2	5FCB6FAB3AD6FAEC	6C44198C4A475817

KEY POINTS

- SHA-512's strength lies in its iterative structure, which ensures that small changes to the input result in significant changes to the output.
- The compression function uses a combination of mathematical operations to create a complex transformation of the input data.
- The round constants and initial hash values are carefully chosen to provide security and prevent collisions.

SIGNATURE

- A person signs a document to show that it originated from her or was approved by her. The signature is proof to the recipient that the document comes from the correct entity. When a customer signs a check, the bank needs to be sure that the check is issued by that customer and nobody else. In other words, a signature on a document, when verified, is a sign of authentication the document is authentic
- Consider a painting signed by an artist. The signature on the art, if authentic, means that the painting is probably authentic.

DIGITAL SIGNATURE

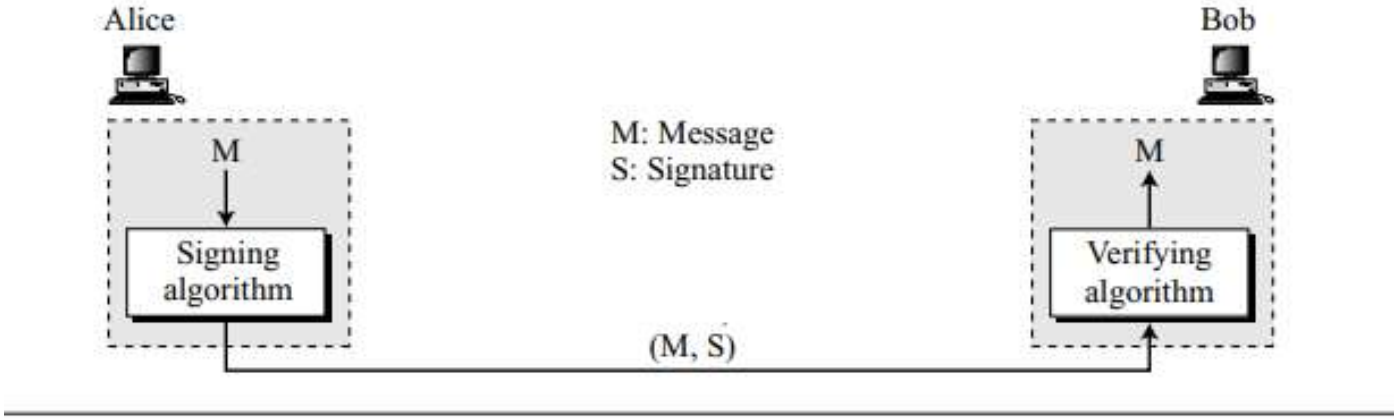
- When Alice sends a message to Bob, Bob needs to check the authenticity of the sender; he needs to be sure that the message comes from Alice and not Eve. Bob can ask Alice to sign the message electronically. In other words, an electronic signature can prove the authenticity of Alice as the sender of the message. We refer to this type of signature as a digital signature

COMPARISON

Conventional	Digital Signature
Included in the document	Signature separate document; Both are received and verified
Copy of signature stored on file; Compares the signature on the document with the signature on file	Copy of signature not stored; Recipient needs to apply a verification technique to the combination of the message and the signature to verify the authenticity
one-to-many relationship between a signature and document	a one-to-one relationship between a signature and a message; Each message has its own signature
copy of the signed document can be distinguished from the original one on file	no such distinction unless there is a factor of time (such as a timestamp) on the document; if Alice sends a document instructing Bob to pay Eve. If Eve intercepts the document and the signature, she can replay it later to get money again from Bob

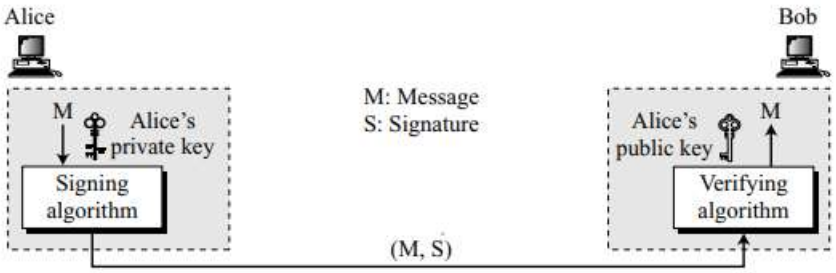
PROCESS

- Sender uses a signing algorithm to sign the message
- Message and the signature are sent to the receiver
- Receiver receives the message and the signature and applies the verifying algorithm



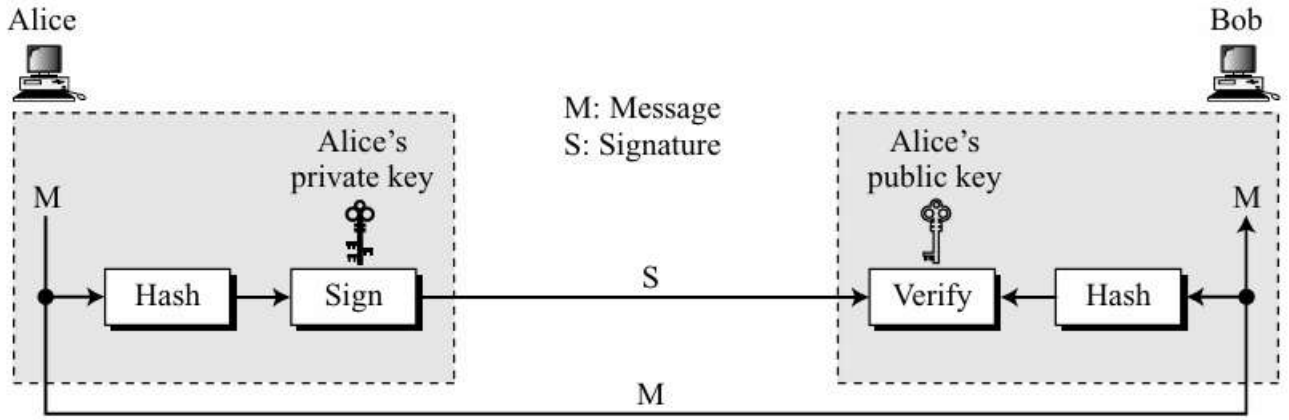
SIGNING DOCUMENT

- A conventional signature is like a private "key" belonging to the signer of the document
- In a digital signature, the signer uses her private key, applied to a signing algorithm, to sign the document. The verifier, on the other hand, uses the public key of the signer, applied to the verifying algorithm, to verify the document.



SIGNING THE DIGEST

- Asymmetric-key systems inefficient with long messages
- Solution is to sign digest of message, shorter than message
- Carefully selected message digest has a one-to-one relationship with the message
- The sender can sign the message digest and the receiver can verify the message digest. The effect is the same.

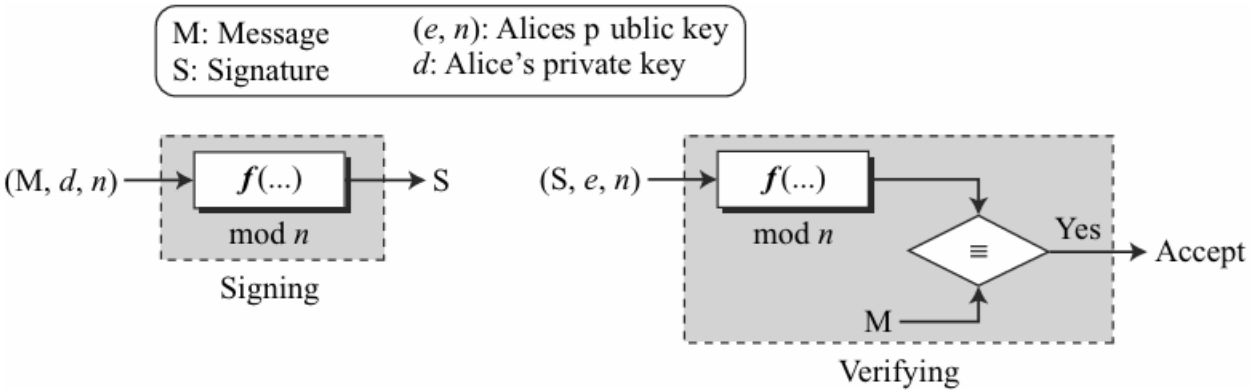


Direct vs Arbitrated Digital Signatures

- Direct Digital Signatures:
- The sender signs the message using their private key. The recipient verifies the signature using the sender's public key.
- If the sender's private key is compromised, the security is breached.
- Arbitrated Digital Signatures:
- Involves a trusted third party (arbiter). The sender signs the message and sends it to the arbiter, who verifies and re-signs it before forwarding it to the recipient.
- Advantage: Adds extra layer of security,can mediate disputes
- Disadvantages: More complex and requires trust in arbiter

RSA Digital Signature Scheme

- Changes roles of private and public keys
- private and public keys of the sender, not the receiver
- Sender uses her own private key to sign
- Private key plays the role of the sender's own signature, sender's public key plays role of copy of signature in file
-



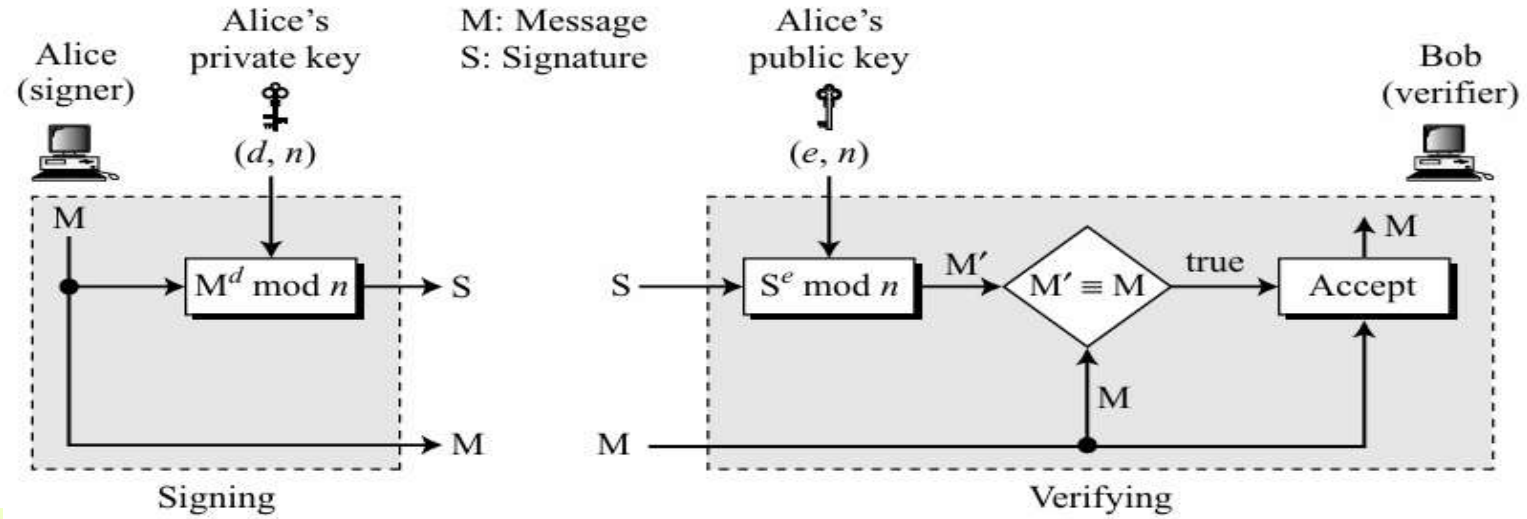
KEY GENERATION

- Same as key generation in the RSA cryptosystem
- Alice chooses two primes p and q and calculates $n = p \times q$
- Alice calculates $\varphi(n) = (p - 1)(q - 1)$
- She then chooses e, the public exponent, and calculates d, the private exponent such that $e \times d = 1 \bmod \varphi(n)$
- Alice keeps d; she publicly announces n and e.

SIGNING AND VERIFYING

- Signing- Alice computes $S = M^d \bmod n$ and sends M,S to Bob
- Verifying - Bob receives M and S and computes M'

$M' = S^e \bmod n$ Bob compares the value of M′ and M. If the two values are congruent, Bob accepts the message



ATTACKS ON RSA SIGNATURE

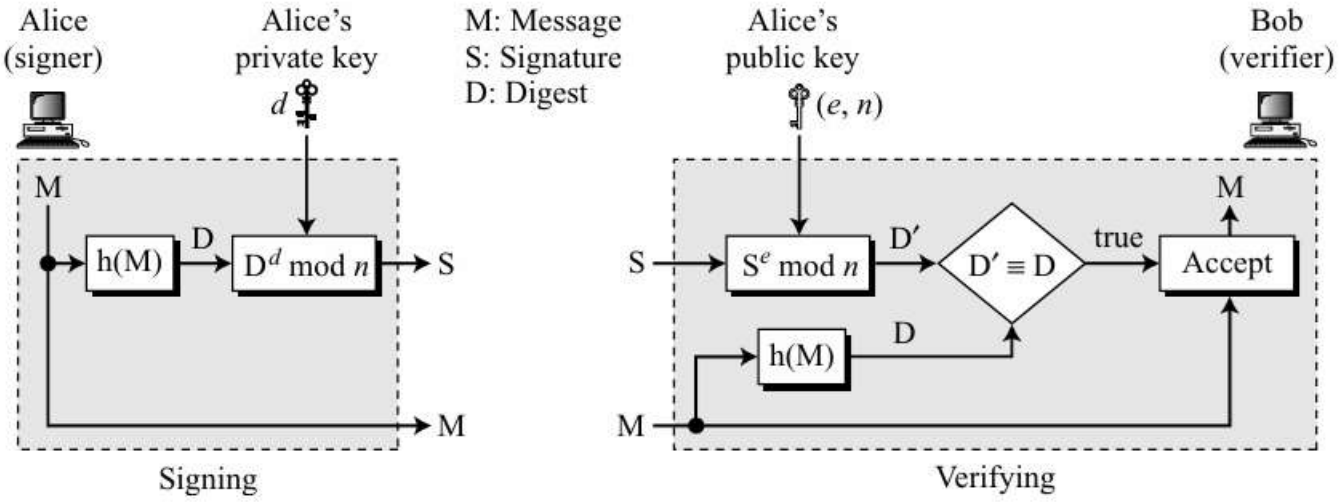
- Key-Only Attack Eve has access only to Alice's public key. Eve intercepts the pair (M, S) and tries to create another message M′ such that $M' \equiv S^e \pmod{n}$. This problem is as difficult to solve as the discrete logarithm problem
- Known-Message Attack: Eve uses the multiplicative property of RSA. Assume that Eve has intercepted two message-signature pairs (M1,S1) and (M2 , S2) created using the same private key.
- If $M = (M1 \times M2) \bmod n$, then $S = (S1 \times S2) \bmod n$.
- Eve can create $M = (M1 \times M2) \bmod n$ & $S = (S1 \times S2) \bmod n$ and fool Bob into believing that S is Alice's signature on the message M

CHOSEN PLAINTEXT ATTACK

- Involves an attacker selecting specific plaintext messages and obtaining their corresponding signatures.
- Selection of Plaintexts: The attacker chooses specific plaintext messages that they want to be signed.
- Obtaining Signatures: The attacker gets the legitimate signer to sign these chosen plaintexts.
- Analysis: By analyzing the signatures of these chosen plaintexts, the attacker attempts to deduce the private key or forge signatures for other messages using the multiplicative property of RSA

RSA Signature on Message Digest

- Signing and verifying processes much faster because RSA digital signature scheme is nothing other than encryption with the private key and decryption with the public key
- Use of a strong crypto graphic hashing function also makes the attack on the signature much more difficult



ELGAMAL DIGITAL SIGNATURE SCHEME

- Digital signature scheme uses the same keys of crypto system, but the algorithm is different
- Key Generation

As with Elgamal encryption, the global elements of **Elgamal digital signature** are a prime number q and α, which is a primitive root of q. User A generates a private/public key pair as follows.

1. Generate a random integer X_A, such that $1 < X_A < q - 1$.
2. Compute $Y_A = \alpha^{X_A} \bmod q$.
3. A's private key is X_A; A's pubic key is $\{q, \alpha, Y_A\}$.

SIGNING A MESSAGE

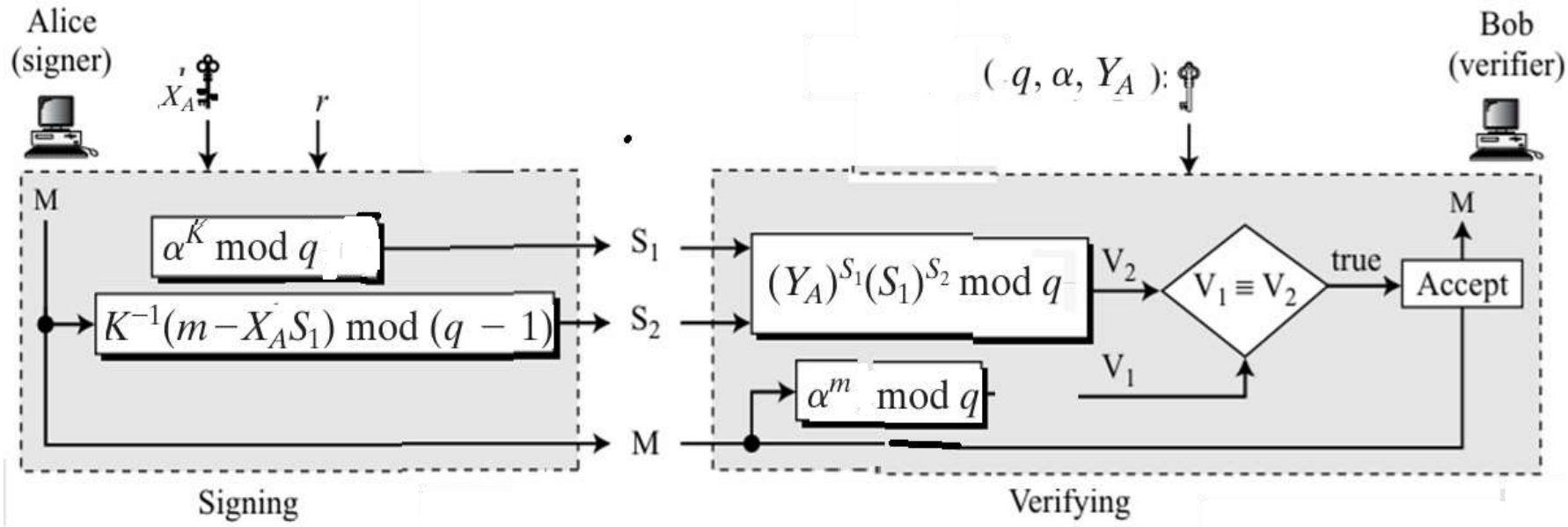
To sign a message M, user A first computes the hash $m = \mathrm{H}(M)$, such that m is an integer in the range $0 \leq m \leq q - 1$. A then forms a digital signature as follows.

1. Choose a random integer K such that $1 \leq K \leq q - 1$ and $\gcd(K, q - 1) = 1$. That is, K is relatively prime to $q - 1$.
2. Compute $S_1 = \alpha^K \bmod q$. Note that this is the same as the computation of C_1 for Elgamal encryption.
3. Compute $K^{-1} \bmod (q - 1)$. That is, compute the inverse of K modulo $q - 1$.
4. Compute $S_2 = K^{-1}(m - X_A S_1) \bmod (q - 1)$.
5. The signature consists of the pair (S_1, S_2).

SIGNING AND VERIFICATION PROCESS

M: Message
S_1, S_2: Signatures
V_1, V_2: Verifications

K : Random secret
X_A : Alice's private key
(q, α, Y_A): Alice's public key



VERIFICATION

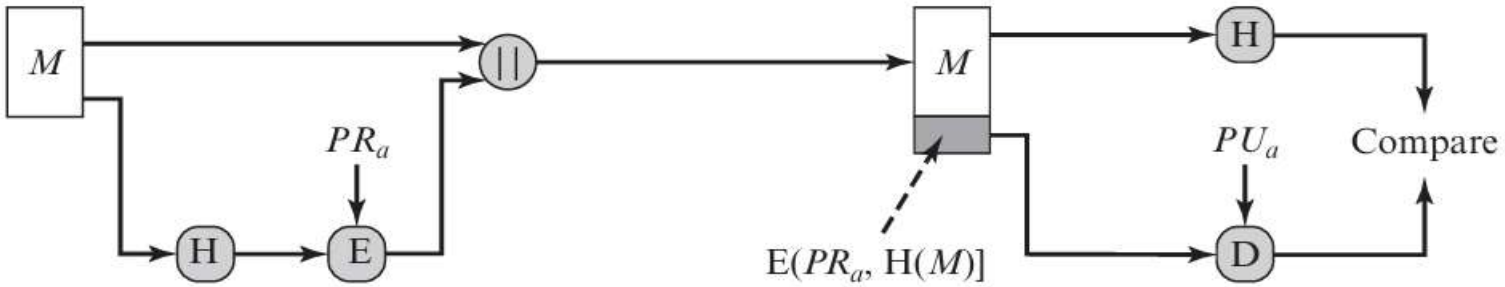
Any user B can verify the signature as follows

1. Compute $V_1 = \alpha^m \bmod q$.
2. Compute $V_2 = (Y_A)^{S_1}(S_1)^{S_2} \bmod q$.

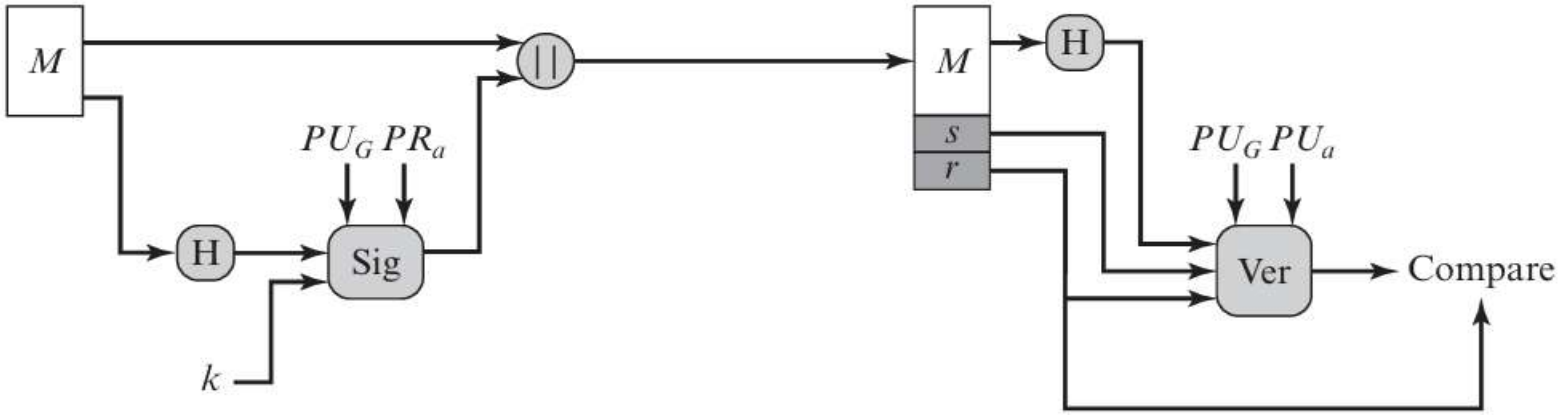
The signature is valid if $V_1 = V_2$.

DSS

- The DSA uses an algorithm that is designed to provide only the digital signature function.



(a) RSA approach



(b) DSA approach

DSA Approach

- 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 (PRa) and a set of parameters known to a group of communicating principals.
- This set constitute a global public key (PUG).The result is a signature consisting of two components, labeled s and r

VERIFICATION

- r is based on the random value k
- and is unique for each signature
- s ensures that signature is tied to
- message and the sender's private key
- Incorporates hash of message
- and private key
- providing a link between
- message and signer

Signing

$r = (g^k \bmod p) \bmod q$

$s = [k^{-1}(H(M) + xr)] \bmod q$

Signature $= (r, s)$

Verifying

$w = (s')^{-1} \bmod q$

$u_1 = [H(M')w] \bmod q$

$u_2 = (r')w \bmod q$

$v = [(g^{u1}y^{u2}) \bmod p] \bmod q$

TEST: $v = r'$