Chapter 3

Message Authentication and Hash Functions

Table of Contents

Message Authentication

Message authentication is a security mechanism used to verify the integrity and authenticity of a message.

It ensures that the data received is exactly as sent, without any modifications, and that the sender's identity is valid.

This process protects against unauthorized modifications, insertions, deletions, and replay attacks.

Key Aspects of Message Authentication

1. **Integrity Verification**:
    - Ensures that the message has not been altered during transmission.
    - Detects any unauthorized modifications, insertions, deletions, or replay of messages.
2. **Authenticity**:
    - Confirms the identity of the sender, ensuring that the message is from a legitimate source.

Methods of Message Authentication

Message Authentication Code (MAC)

A Message Authentication Code (MAC) is an authentication technique that uses a secret key to generate a small, fixed-size block of data that is appended to the message.

This technique ensures the integrity and authenticity of a message during transmission between two parties who share a common secret key.

Process of MAC Generation and Verification

1. **Shared Secret Key**:
    - The two communicating parties, say A (the sender) and B (the receiver), share a common secret key K.
2. **MAC Calculation by Sender (A)**:
    - When A wants to send a message M to B, it calculates the MAC using a function C that takes the message M and the secret key K as inputs: MAC=C(K,M)
    - Here:
        - M = input message

- C = MAC function (e.g., HMAC, CMAC)
- K = shared secret key
- MAC = message authentication code

3. **Transmission**:
   - The sender (A) transmits the message M along with the MAC to the intended recipient (B).
4. **MAC Verification by Receiver (B)**:
   - Upon receiving the message M and the MAC, the recipient (B) performs the same calculation using the received message M and the shared secret key K to generate a new MAC:

     $$MAC_{new}=C(K,M)$$

   - The recipient then compares the received MAC with the newly calculated MAC:

     if $MAC_{received}=MAC_{new}$, then the message is authenticated and unaltered.

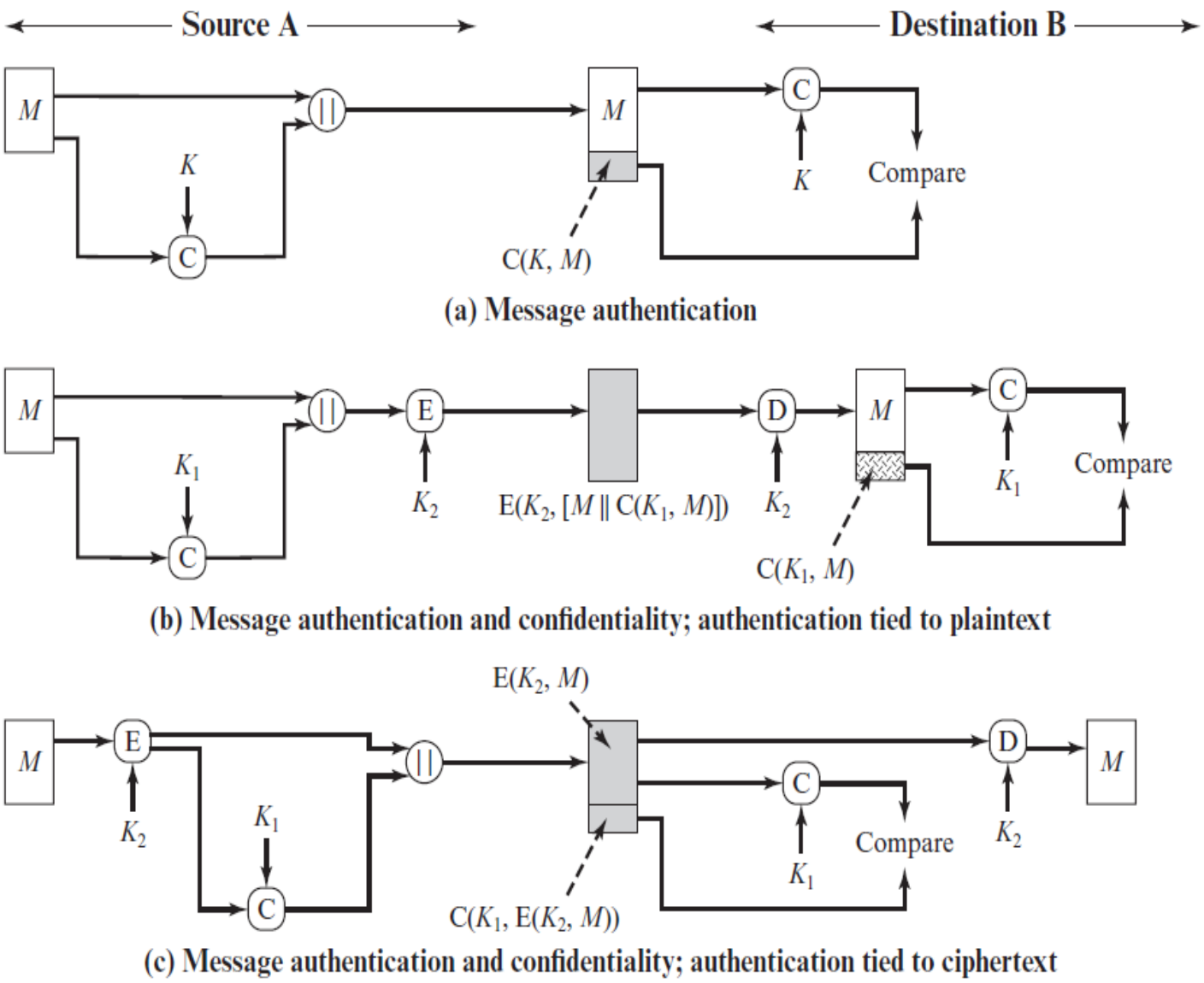


Figure: Basic Uses of Message Authentication code (MAC)

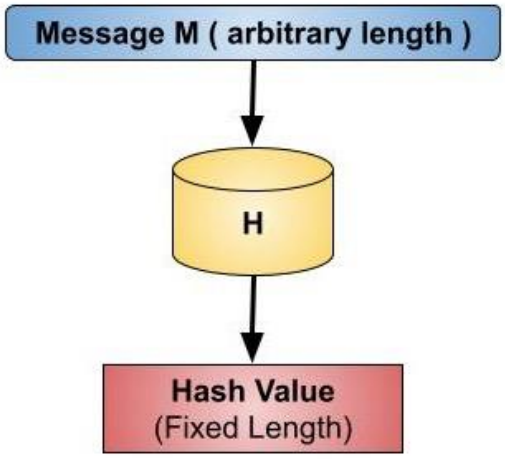
Hash Function

A hash function in cryptography is like a mathematical function that takes various inputs, like messages or data, and transforms them into fixed-length strings of characters.

Means the input to the hash function is of any length but output is always of fixed length. This is like compressing a large balloon into a compact ball.

Hash functions play a crucial role in various security applications, including password storage (hash values instead of passwords), digital signatures, and data integrity checks.

Hash values, or message digests, are values that a hash function returns. The hash function is shown in the image below −



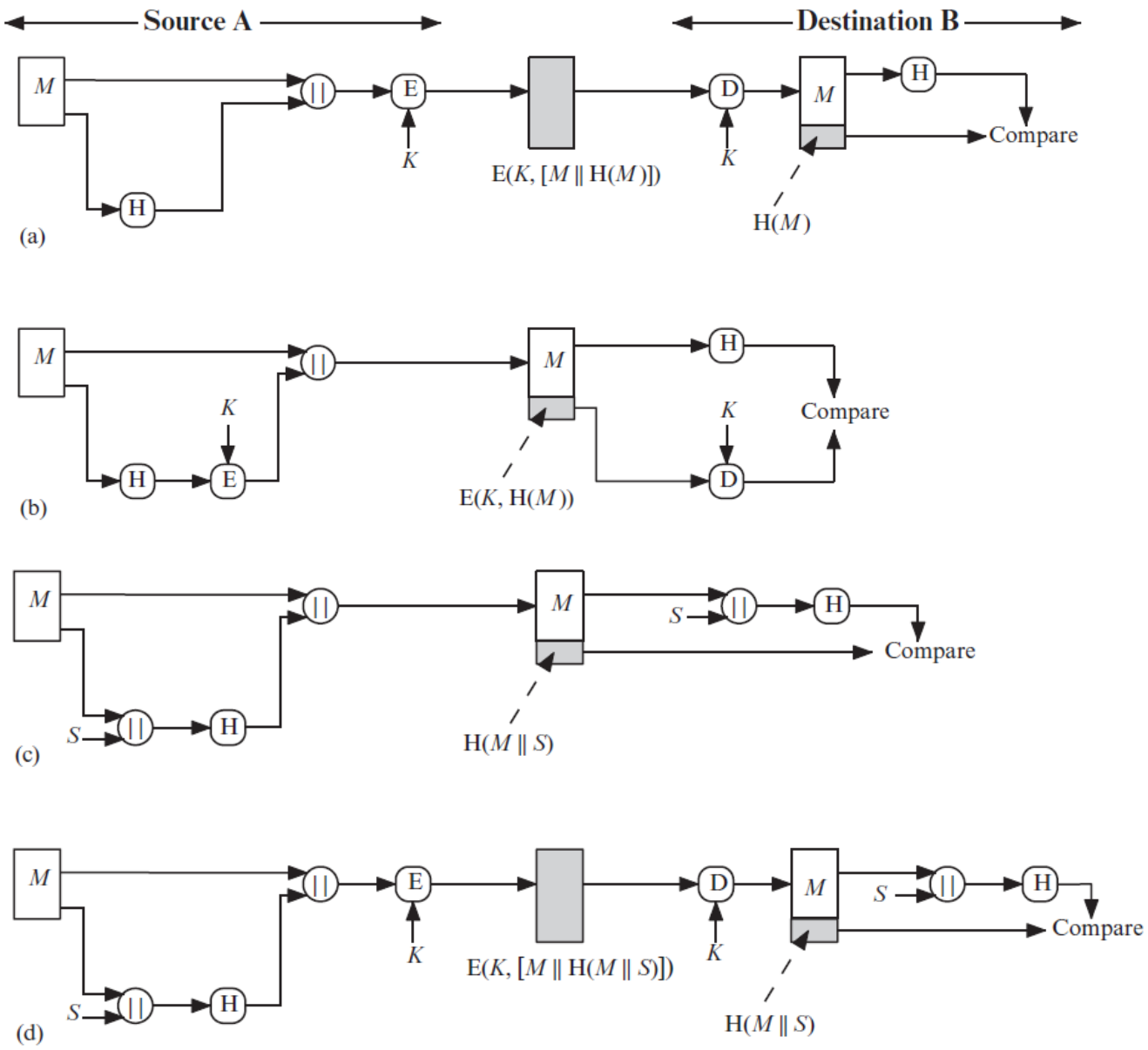


Figure: Simplified Examples of the Use of a Hash Function for Message Authentication

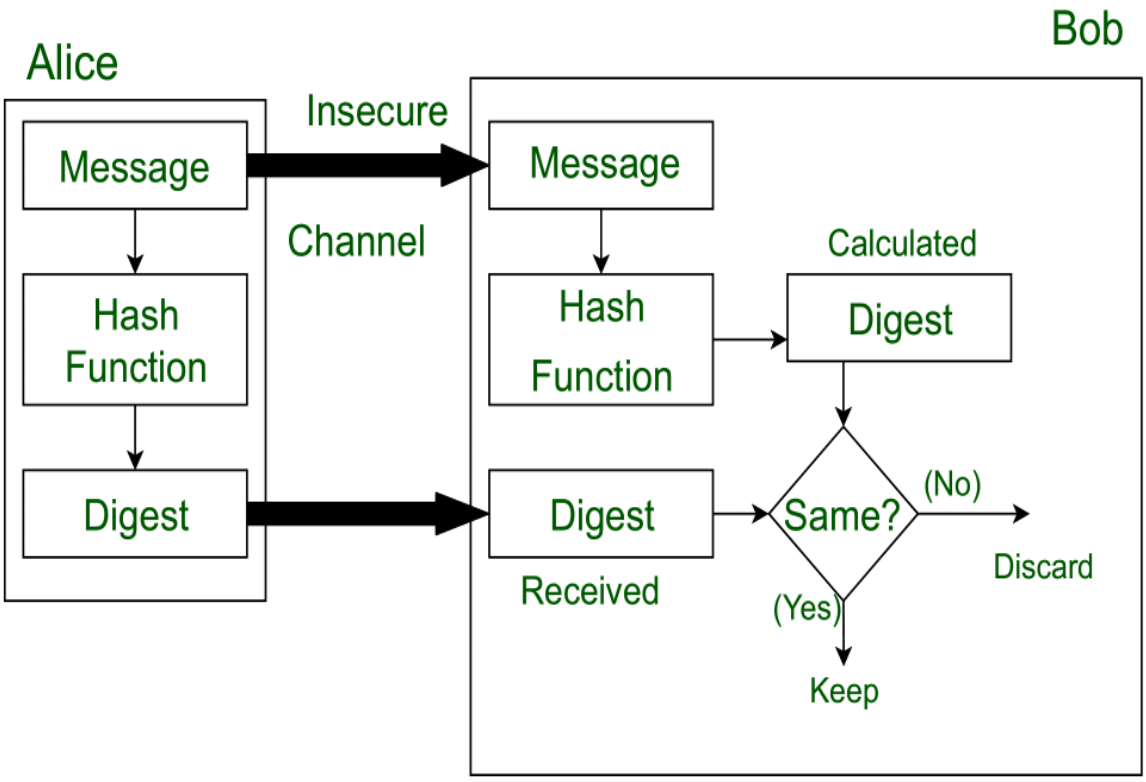
Message Digest

A message digest, also known as a hash value or hash code, is the output of a cryptographic hash function.

It is a fixed-size string of characters that uniquely represents the input data.

Message digests are crucial in ensuring data integrity and authenticity in various applications such as digital signatures, password storage, and data integrity verification.

Lets assume, Alice sent a message and digest pair to Bob. To check the integrity of the message Bob runs the cryptographic hash function on the received message and gets a new digest. Now, Bob will compare the new digest and the digest sent by Alice. If, both are same then Bob is sure that the original message is not changed.



MD4

MD4 (Message Digest Algorithm 4) is a cryptographic hash function developed by Ronald Rivest in 1990.

MD4 is designed to take an input message of arbitrary length and produce a 128-bit (16-byte) hash value, often represented as a 32-character hexadecimal number.

Working Principle of MD4

1. **Input Message Preparation**:
    - **Message Length**: MD4 can handle messages of arbitrary length. However, the total length of the message (in bits) must be congruent to 448 modulo 512 after padding.
2. **Padding the Message**:
    - **Step 1**: Append a single '1' bit to the message.
    - **Step 2**: Append '0' bits until the total length of the message is 448 bits modulo 512.
        - This means the padded message length will be 448 bits modulo 512, making the length equal to 448 bits when combined with the original message length.

Example:

- For a message "abc" (in binary: 01100001 01100010 01100011 which is 24 bits):
    - Append a '1' bit: 01100001 01100010 01100011 10000000 (now 32 bits).

- Append '0' bits until the total length is 448 bits modulo 512.
- The message becomes: 01100001 01100010 01100011 10000000 followed by 416 '0' bits to make a total length of 448 bits.

3. **Appending the Length**:
   - **Step 3**: Append a 64-bit representation of the original message length (before padding).
     - The length is represented in little-endian format (least significant byte first).

Example:

- Original length of "abc" in bits is 24 (which is 0x18 in hexadecimal).
- Represent 24 as a 64-bit number: 00000000 00000000 00000000 00000000 00000000 00000000 00011000 00000000 (in little-endian format).
- Append this to the padded message to make the total length a multiple of 512 bits.

4. **Initialize MD Buffer**:
    - MD4 uses four 32-bit buffers initialized to specific constants:
        - A = 0x67452301
        - B = 0xefcdab89
        - C = 0x98badcfe
        - D = 0x10325476
5. **Process Each 512-bit Block**:
    - The padded message is divided into 512-bit blocks.
    - Each block is processed in three rounds, with 16 operations per round.
        - **Round 1**: Uses function F (X, Y, Z) = (X & Y) | (~X & Z)
        - **Round 2**: Uses function G (X, Y, Z) = (X & Y) | (X & Z) | (Y & Z)
        - **Round 3**: Uses function H (X, Y, Z) = X ^ Y ^ Z

- In each round, specific bitwise operations and shifts are performed using the message block and the current state of buffers A, B, C, and D.

6. **Updating the Buffers**:
   - After processing each block, the results are added to the initial values of A, B, C, and D.
7. **Final Hash**:
   - After all blocks are processed, the final values of A, B, C, and D are concatenated to form the 128-bit hash (16 bytes).

Aspect	MD4	MD5
Rounds	3 rounds, 16 operations each	4 rounds, 16 operations each
Functions Used	A, B, C	A, B, C, D
Logical Functions	Simpler logical functions	More complex logical functions
A Function	A(X, Y, Z) = (X & Y) | (~X & Z)	A(X, Y, Z) = (X & Y) | (~X & Z)
B Function	B(X, Y, Z) = (X & Y) | (X & Z) | (Y & Z)	B(X, Y, Z) = (X & Z) | (Y & ~Z)
C Function	C(X, Y, Z) = X ^ Y ^ Z	C(X, Y, Z) = X ^ Y ^ Z
D Function	N/A	D(X, Y, Z) = Y ^ (X | ~Z)
Constants	Few constants, simpler usage	Unique set of constants derived from the sine function
Bit Shifts	Simpler bit-shifting operations	More varied bit-shifting operations
Padding	Message length congruent to 448 modulo 512	Message length congruent to 448 modulo 512
Length Appending	64-bit little-endian representation of message length	64-bit little-endian representation of message length
Buffer Initialization	A = 0x67452301, B = 0xefcdab89, C = 0x98badcfe, D = 0x10325476	A = 0x67452301, B = 0xefcdab89, C = 0x98badcfe, D = 0x10325476

MD5

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 as an improvement of MD4, with advanced security purposes.

The output of MD5 (Digest size) is always **128 bits. MD5** was developed in 1991 by **Ronald Rivest.**

Use Of MD5 Algorithm:

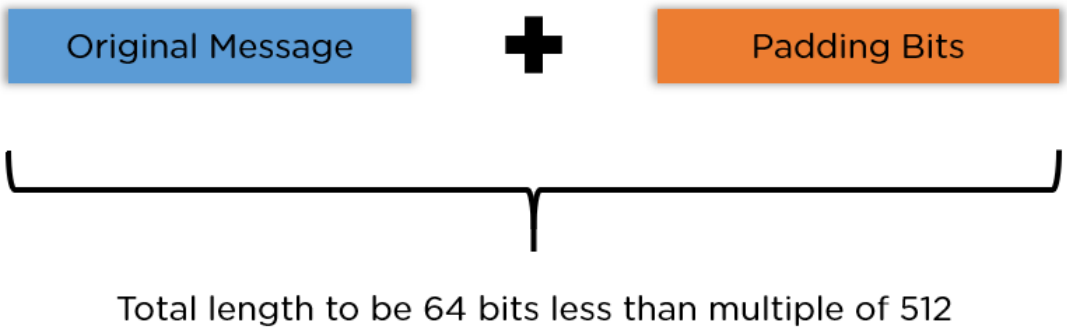
- It is used for file authentication.
- In a web application, it is used for security purposes. e.g. Secure password of users etc.
- Using this algorithm, We can store our password in 128 bits format.

Steps in MD5 Algorithm

There are four major sections of the algorithm:

Padding Bits

When you receive the input string, you have to make sure the size is 64 bits short of a multiple of 512. When it comes to padding the bits, you must add one(1) first, followed by zeroes to round out the extra characters.



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 . . .

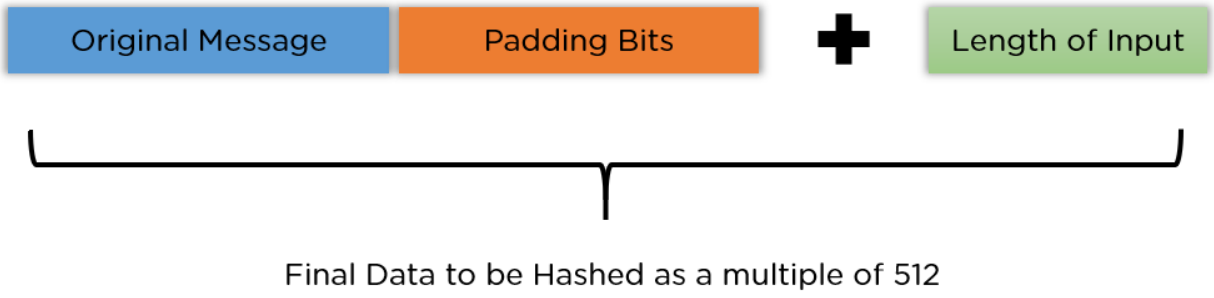
Padding Length

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.



Initialize MD Buffer

The entire string is converted into multiple blocks of 512 bits each. You also need to initialize four different buffers, namely A, B, C, and D. These buffers are 32 bits each and are initialized as follows:

A = 01 23 45 67

B = 89 ab cd ef

C = fe dc ba 98

D = 76 54 32 10

Process Each Block

Each 512-bit block gets broken down further into 16 sub-blocks of 32 bits each. There are four rounds of operations, with each round utilizing all the sub-blocks, the buffers, and a constant array value.

This constant array can be denoted as T[1] -> T[64].

Each of the sub-blocks are denoted as M[0] -> M[15].

A total of 64 operations are performed in 4 rounds, each round with 16 operations.

We apply a different function on each round i.e.

Round 1: (b AND c) OR ((NOT b) AND (d))

Round 2: (b AND d) OR (c AND (NOT d))

Round 3: b XOR c XOR d

Round 4: c XOR (b OR (NOT d))

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

- add modulo 2^{32}
- M[i] – 32 bit message.
- K[i] – 32-bit constant.
- <<<n – Left shift by n bits.

In the MD5 algorithm, the number of bits to rotate varies for each operation within a round. MD5 has four rounds, and each round uses a different set of rotation amounts.

Here are the specific rotation amounts used in each round:

Round 1:

- s[1]=7
- s[2]=12
- s[3]=17
- s[4]=22

Round 2:

- s[1]=5
- s[2]=9
- s[3]=14
- s[4]=20

Round 3:

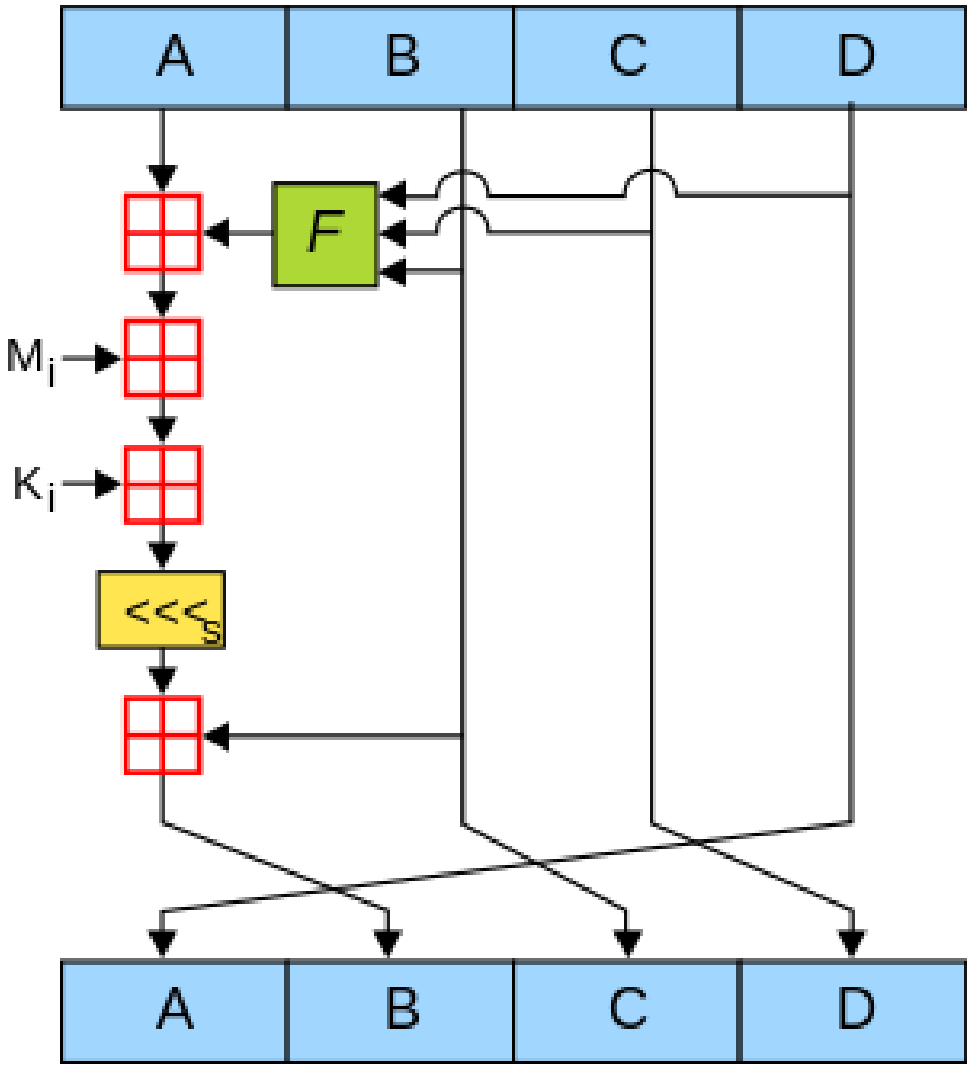
- s[1]=4
- s[2]=11
- s[3]=16
- s[4]=23

Round 4:

- s[1]=6
- s[2]=10
- s[3]=15
- s[4]=21

During each round, there are 16 operations, and the rotation amounts are used cyclically.

Specifically, in the ith operation of a round, the rotation amount used is determined by $s[(i \bmod 4)+1]$



Output:

After all, rounds have been performed, the buffer A, B, C, and D contains the MD5 output starting with the lower bit A and ending with Higher bits D.

Secure Hash Algorithm 1 (SHA-1)

SHA-1 is a cryptographic hash function designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard. Although SHA-1 was once widely used, it is now considered insecure against well-funded attackers and has been largely replaced by newer hash functions in the SHA-2 and SHA-3 families.

SHA-1 is now considered insecure since 2005. Major tech giants browsers like Microsoft, Google, Apple and Mozilla have stopped accepting SHA-1 SSL certificates by 2017.

Overview

- **Hash Size**: 160 bits (20 bytes)
- **Block Size**: 512 bits (64 bytes)
- **Digest Length**: 40 hexadecimal characters
- **Rounds**: 80 rounds of processing

Steps in SHA-1

1. **Padding the Message**:
    - The original message is padded so that its length (in bits) is congruent to 448 modulo 512.
    - Padding is done by appending a single '1' bit, followed by '0' bits, until the message length is 448 modulo 512.
    - A 64-bit representation of the original message length (before padding) is then appended to the message.
2. **Initialize Buffers**:
    - SHA-1 uses five 32-bit buffers initialized to specific constants:
        - H0=0x67452301
        - H1=0xEFCDAB89
        - H2=0x98BADCFE
        - H3=0x10325476
        - H4=0xC3D2E1F0

3. **Process Each 512-bit Block**:
    - The padded message is divided into 512-bit blocks.
    - Each block is processed in four rounds of 20 operations each, making a total of 80 operations per block.
4. **Round Functions**:
    - SHA-1 uses four different functions in its rounds, which are applied to the message schedule and the current hash value.
    - **Round 1 ($0 \leq t \leq 19$)**:

$$f(t,B,C,D)=(B\&C)\|((\sim B)\&D)$$

        - Constant: K=0x5A827999
    - **Round 2 ($20 \leq t \leq 39$)**:

$$f(t,B,C,D)=B\oplus C\oplus D$$

        - Constant: K=0x6ED9EBA1
    - **Round 3 ($40 \leq t \leq 59$)**:

$$f(t,B,C,D)=(B\&C)\|(B\&D)\|(C\&D)$$

- Constant: K=0x8F1BBCDC

- **Round 4 ($60 \leq t \leq 79$)**:

$$f(t,B,C,D)=B \oplus C \oplus D$$

- Constant: K=0xCA62C1D6

5. **Message Schedule**:

- Each 512-bit block is divided into sixteen 32-bit words: W0,W1,…,W15
- The message schedule array WWW is expanded to 80 words using: $W_t=\text{ROTATE\_LEFT}(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16},1)$

6. **Main Loop**:

- For each block, the initial hash values H0H_0H0 through H4H_4H4 are updated:

$$A=H0$$

$$B=H1$$

$$C=H2$$

$$D=H3$$

$$E=H4$$

- For each round t ($0 \leq t \leq 79$):

  $TEMP=ROTATE\_LEFT(A,5)+f(t,B,C,D)+E+W_t+K_t$

  $E=D$

  $D=C$

  $C=ROTATE\_LEFT(B,30)$

  $B=A$

  $A=TEMP$

- Add the result to the current hash value:

  $H0=H0+A$

  $H1=H1+B$

  $H2=H2+C$

  $H3=H3+D$

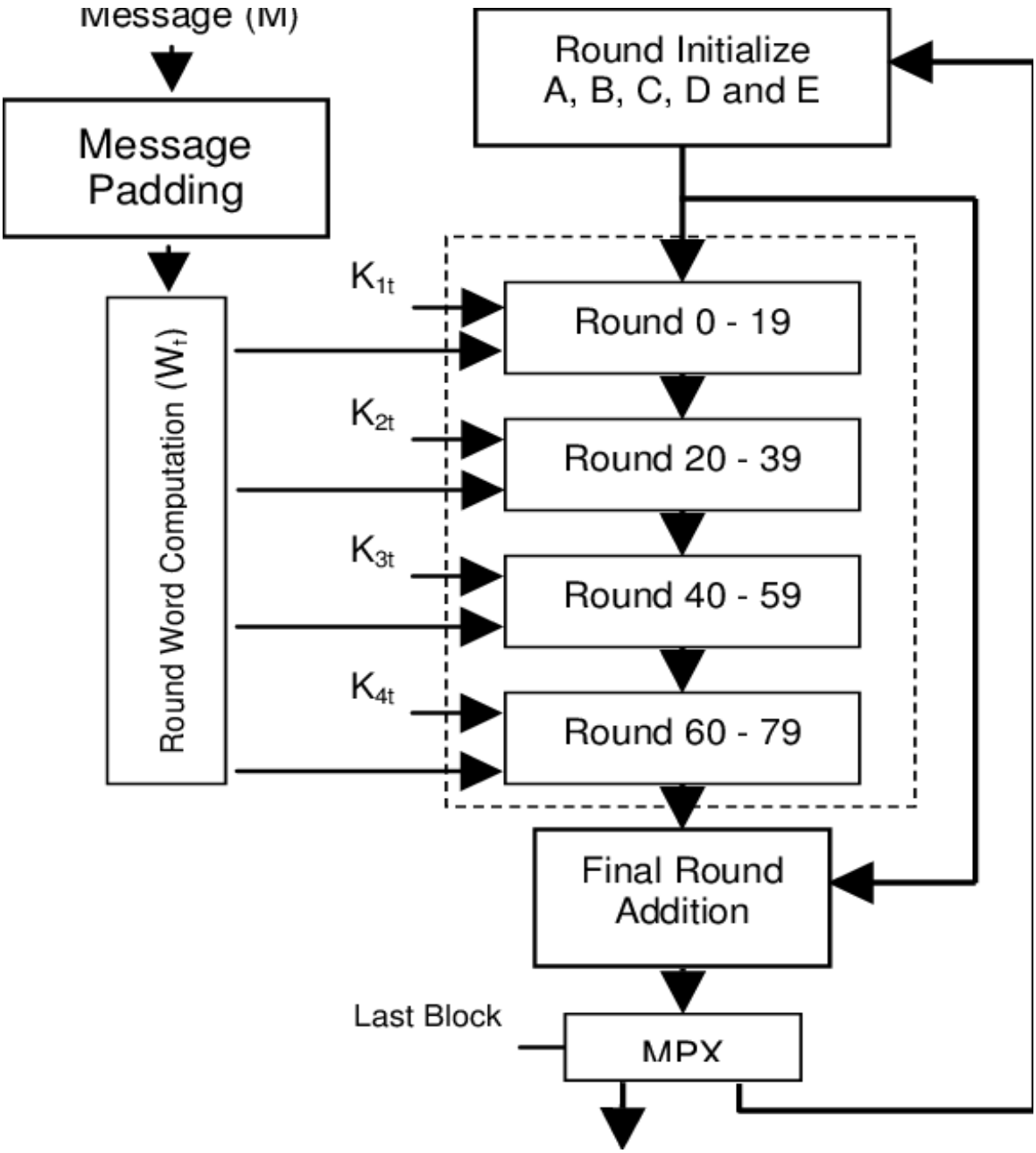
  $H4=H4+E$

7. **Output**:

   - The final hash value is the concatenation of H0, H1, H2, H3, and H4:

Hash=H0‖H1‖H2‖H3‖H4

- This results in a 160-bit (20-byte) hash value.



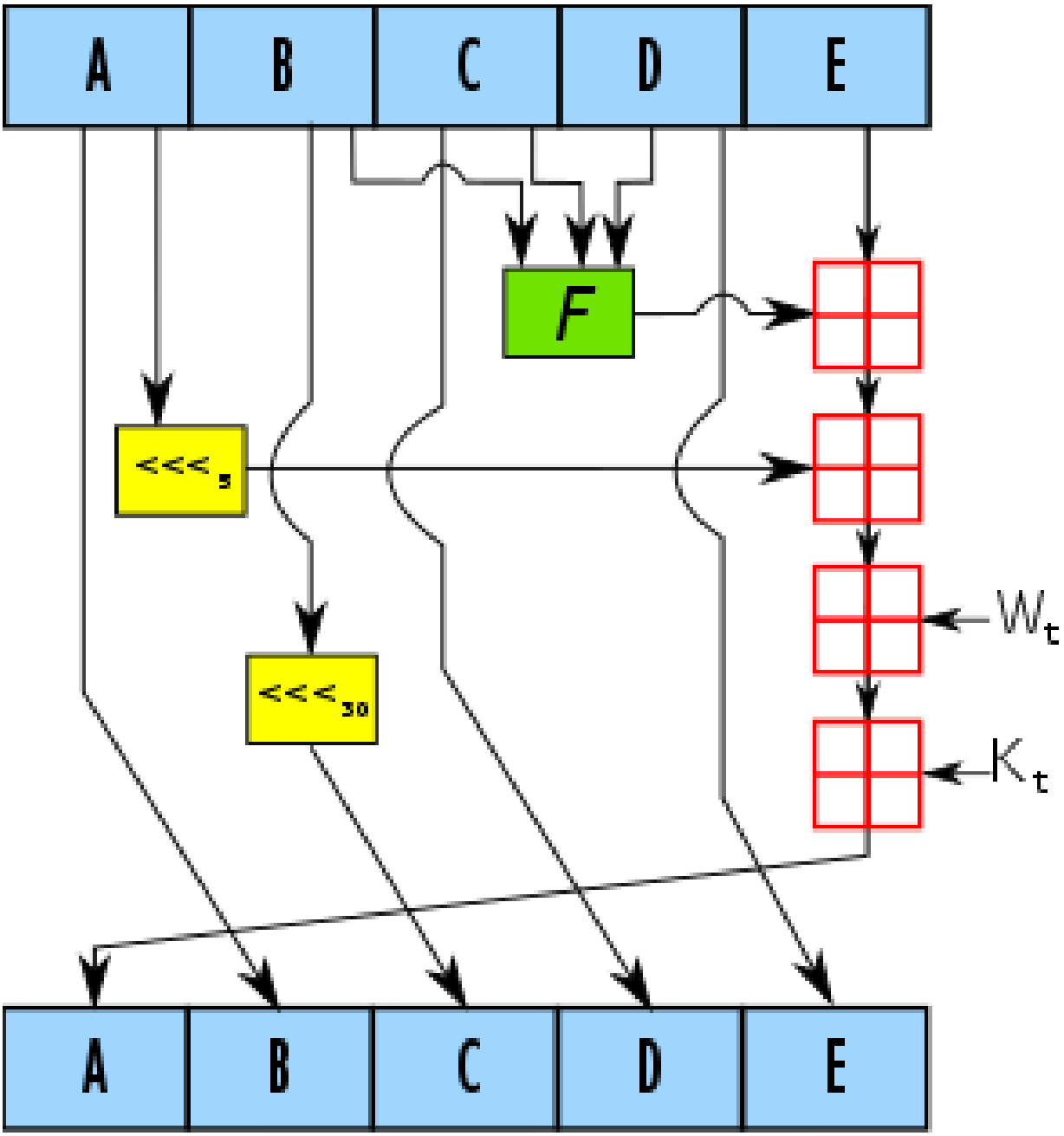


Figure: working of each round

Aspect	MD5	SHA-1
Full Form	Message Digest	Secure Hash Algorithm
Message Digest Length	128 bits	160 bits
Speed	Faster	Slower
Security (Operations needed to find the initial message)	2^{128} operations	2^{160} operations
Complexity	Simpler	More complex
Security Level	Poor	Balanced/Moderate
Collision Resistance (Operations needed to find two messages with the same digest)	2^{64} operations	2^{80} operations
Year of Introduction	1992	1995

Figure: MD5 vs SHA-1

HMAC (Hash-based Message Authentication Code)

HMAC is a type of message authentication code (MAC) that involves a cryptographic hash function and a secret cryptographic key. It can be used with any iterative cryptographic hash function, like MD5 or SHA-1, in combination with a secret shared key. The resulting HMAC provides both data integrity and authenticity.

Key Features of HMAC

1. **Message Integrity and Authentication**: Ensures the message has not been altered and verifies the sender.
2. **Uses a Secret Key**: Enhances the security provided by the hash function alone.
3. **Supports Various Hash Functions**: Can be implemented with different hash functions (e.g., MD5, SHA-1).

HMAC Algorithm Steps

Here's how HMAC works in simple terms:

1. **Inputs**: HMAC takes two inputs:
    - A **message** that needs to be authenticated.
    - A **secret key** known only to the sender and the receiver.

2. **Hash Function**: HMAC uses a cryptographic hash function (such as MD5, SHA-1, SHA-256, etc.). Let's denote the hash function by H.
3. **Processing**:
    - The message is hashed twice:
        - First, the secret key is XORed with an inner pad (IPAD) to create a modified key.
        - Second, the modified key is concatenated with the message and hashed using the hash function H to produce an intermediate hash value.
        - This intermediate hash value is then XORed with an outer pad (OPAD) derived from the secret key.
        - Finally, the result of the XOR operation is hashed again using H to produce the HMAC.
4. **Output**: The HMAC is a fixed-size string that serves as a message authentication code. It is sent along with the message.

5. **Verification**: On the receiving end, the HMAC is recalculated using the received message and the same secret key. If the recalculated HMAC matches the received HMAC, the message integrity and authenticity are verified.

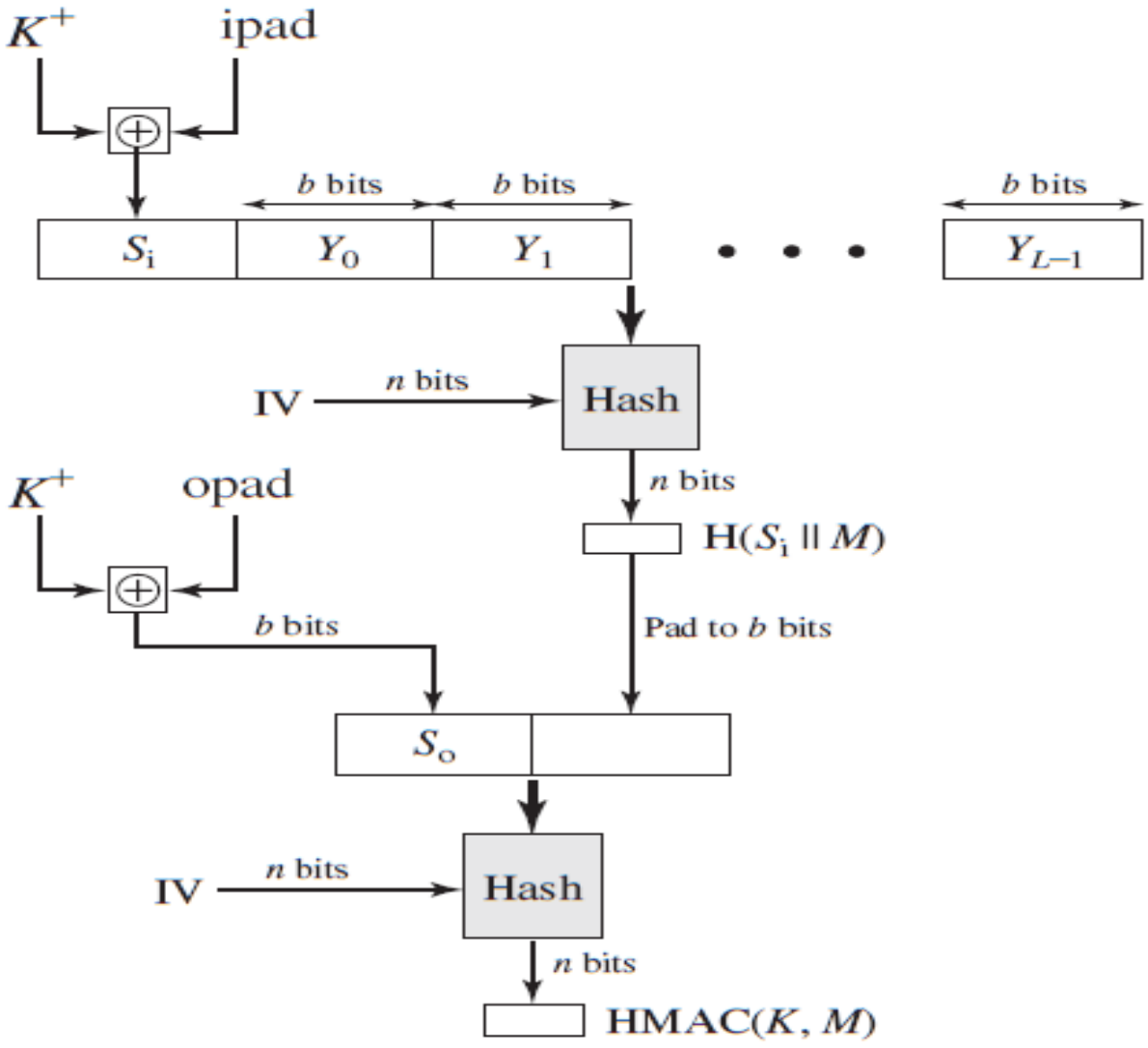


Figure: HMAC Structure

K+ = *K* padded with zeros on the left so that the result is *b* bits in length

ipad = 00110110 (36 in hexadecimal) repeated *b*/8 times

opad = 01011100 (5C in hexadecimal) repeated *b*/8 times

Then HMAC can be expressed as follows:

HMAC(*K*, *M*) = H[(*K*+ _opad) }H[*K*+ _ipad] }*M*]]

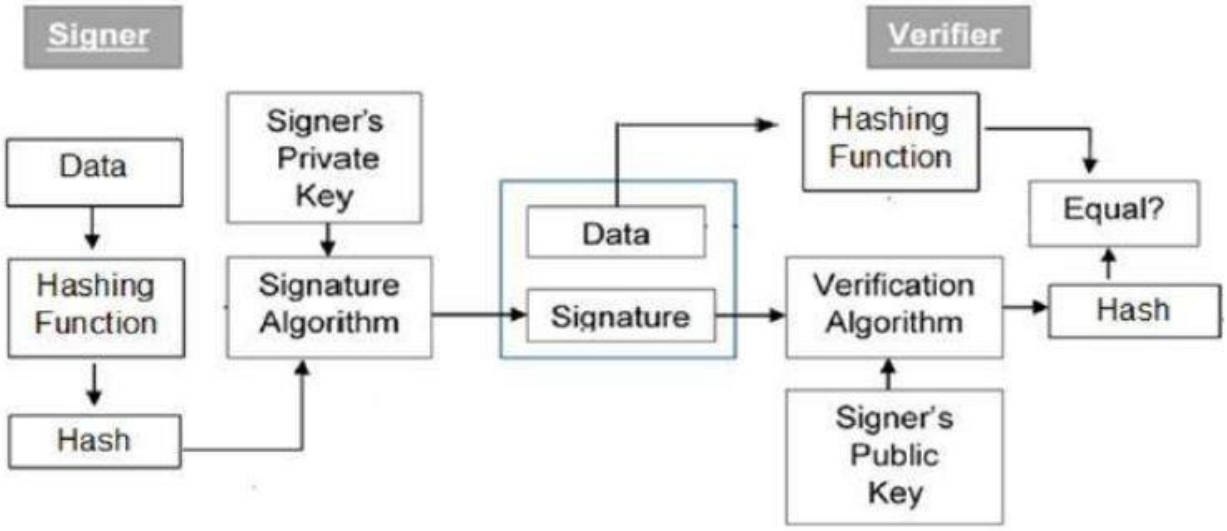
Components and Notations

- **H**: Embedded hash function (e.g., SHA-1, SHA-256).
- **M**: Message input to HMAC.
- **Y_i**: The ith block of the message M, where $0 \leq i \leq (L-1)$.
- **L**: Number of blocks in the message M.
- **b**: Number of bits in a block.
- **n**: Length of the hash code produced by the embedded hash function.
- **K**: Secret key.
- **IV**: Initial vector used in the hash function.

Digital Signatures

- Digital signature is a technique that binds a person/entity to digital data. This binding can be independent verified by the receiver as well as third party
- Digital signature is a cryptographic value that is calculated from the data and a secret key known only to signer
- In real world the receiver of message needs assurance that the message belongs to the sender. This requirement is very crucial in business application

Model of digital signatures



- Everyone using this scheme has public private key pair
- Generally, the key pairs used for encryption/decryption and signing/verifying are different. The private key used for signing is referred to as signature key and public key as verification key
- Signer feeds data to the hash function and generated the hash of data
- Hash value and signature key are then fed to the signature algorithm which produces the digital signature on given hash. Signature is appended to the data and then both are sent to the verifier
- Verifier feeds the digital signature and verification key into verification algorithm. The verification algorithm gives some value as output.
- Verifier also run same hash function on received data to generate hash value
- For verification, this hash value and output of verification algorithm are compared. Based on comparison results, verifier decides whether the signature is valid

- Since digital signature is created by private key of signer and no one else can have this key; the signer cannot repudiate signing the data in future