Graphic Era Deemed to be University
DEHRADUN

GRAPHIC ERA deemed to be UNIVERSITY formerly GEIT

Cryptography
(TBC 504)
UNIT-5

BCA

3rd Year (5th semester)

Dr. Upma Jain

Assistant Professor

9548283808

Message authentication

- Message authentication is a mechanism or service used to verify the integrity of a message.

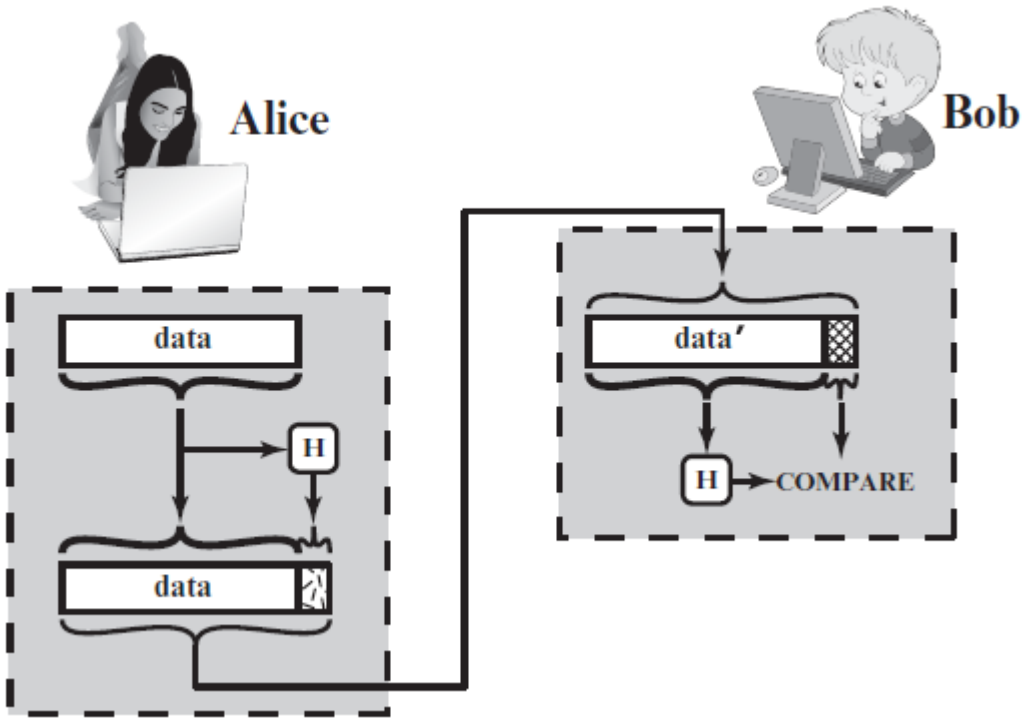
- Message authentication assures that data received are exactly as sent (i.e., there is no modification, insertion, deletion, or replay).

- When a hash function is used to provide message authentication, the hash function value is often referred to as a **message digest**.

- 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.

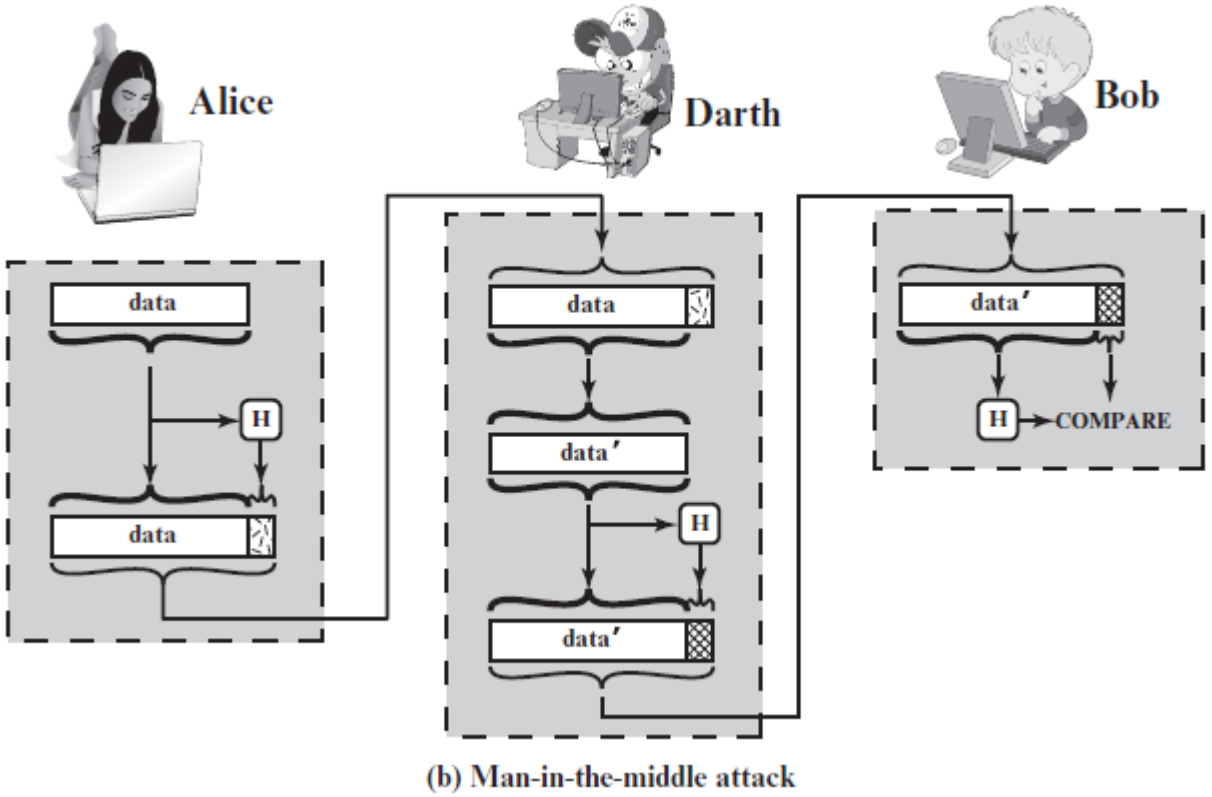


- The hash value must be transmitted in a secure fashion. That is, the hash value must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver.

- In this aches a hash va

- Darth i data block, a .

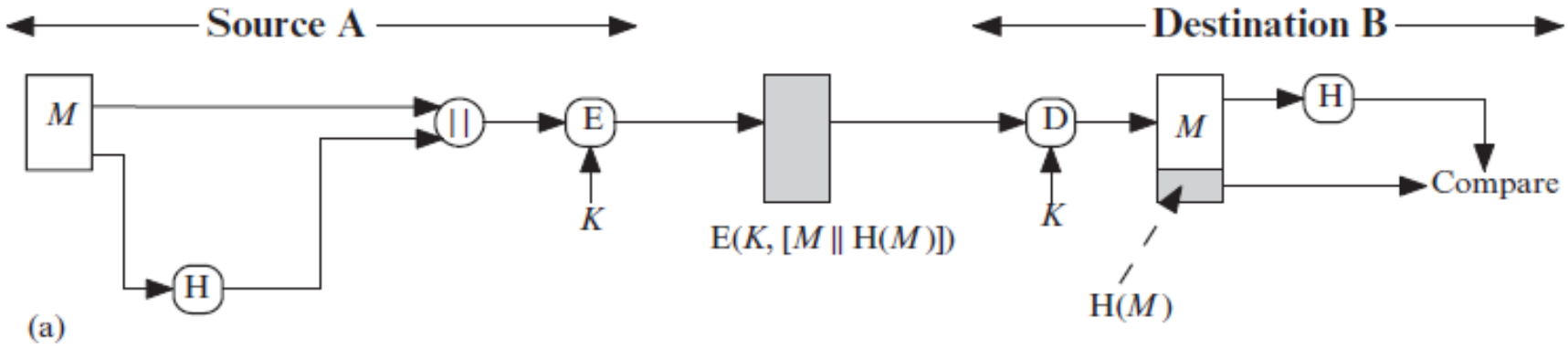
- Bob re ue and does n



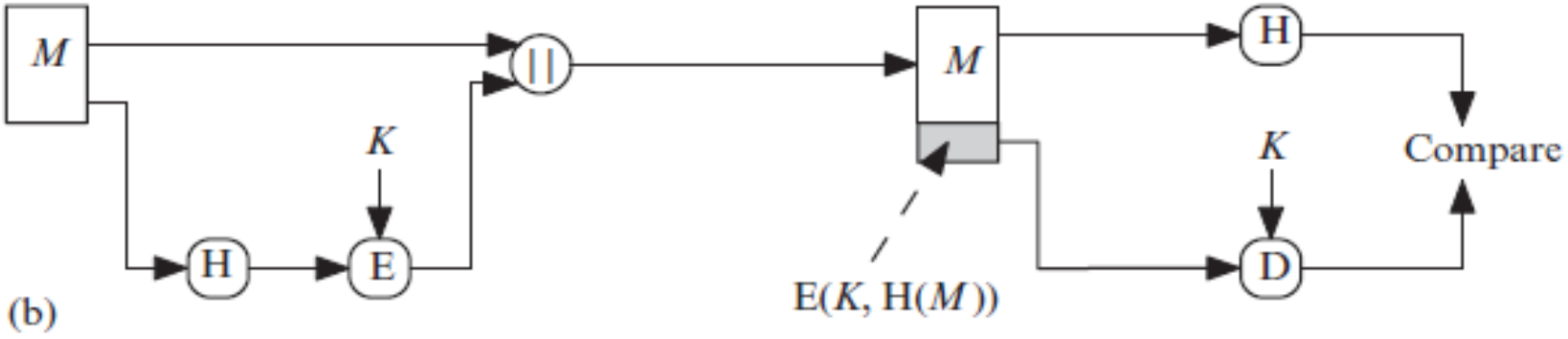
(b) Man-in-the-middle attack

- To prevent this attack, the hash value generated by Alice must be protected.

- a variety of ways in which a hash code can be used to provide message authentication;
  - 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.

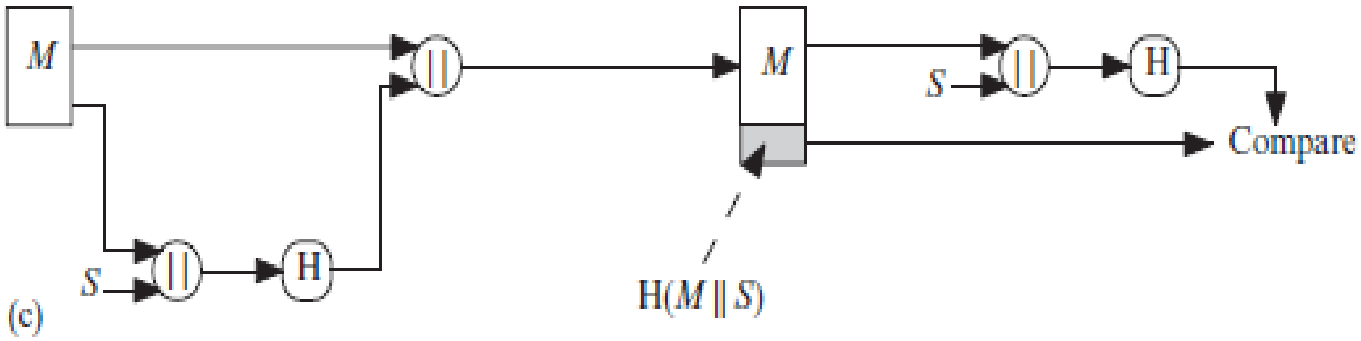


– Only the hash code is encrypted, using symmetric encryption.
– This reduces the processing burden for those applications that do not require confidentiality.



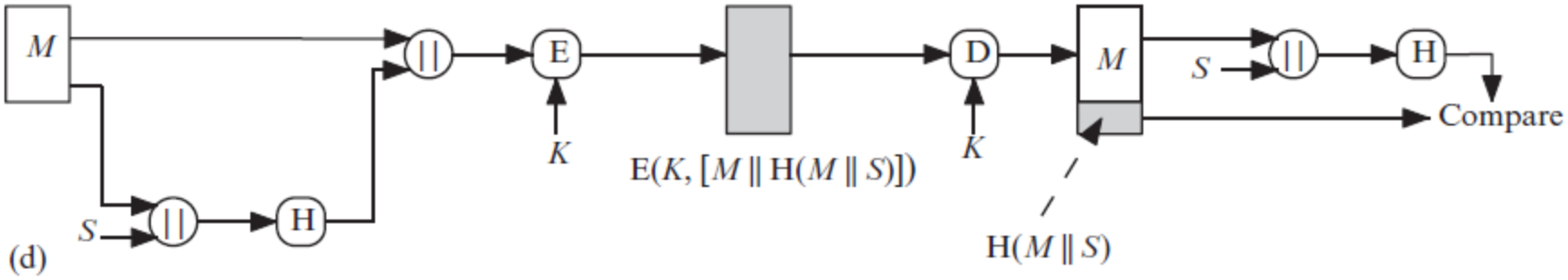
- It is possible to use a hash function but no encryption for message authentication.

- The technique assumes that the two communicating parties share a common secret value *S*.

- A co  of *M* a *1*.

- Because B possesses S, it can recomputed 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.

- Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.



- Conclusion:
  - When confidentiality is not required, method (2) has an advantage over methods (1) and (4), which encrypts the entire message, in that less computation is required.

- 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.

- More commonly, message authentication is achieved using a **message authentication code (MAC)**, also known as a **keyed hash function**.

- Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.

- A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC, which is associated with the protected message.

- If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value. An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key.

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) with $x \neq y$, such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

MESSAGE AUTHENTICATION REQUIREMENTS

- In the context of communications across a network, the following attacks can be identified.
  - Disclosure
  - **Traffic analysis**
  - **Content modification**
  - **Sequence modification**
  - **Timing modification**
  - **Source and destination repudiation**

- message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. Message authentication may also verify sequencing and timeliness.

- A digital signature is an authentication technique that also includes measures to counter repudiation by the source.

- The types of functions that may be used to produce an authenticator. These may be grouped into three classes:
  - **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

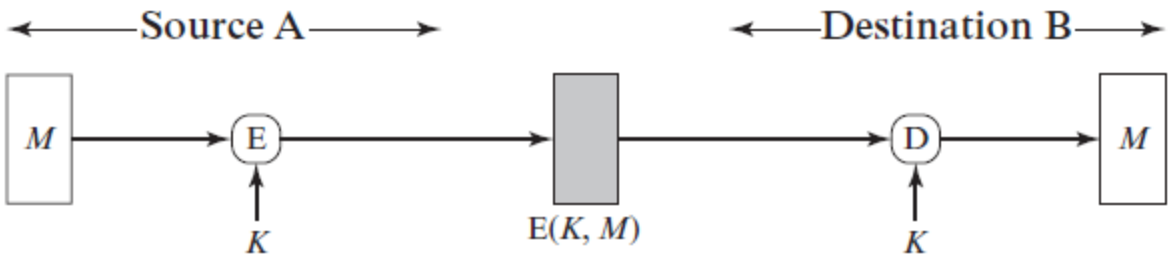
message authentication function

- Any message authentication or digital signature mechanism has two levels of functionality.

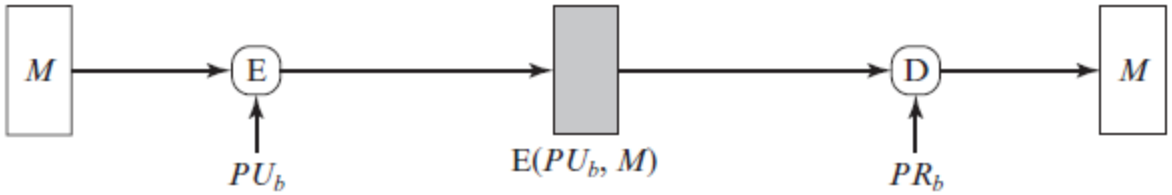
- At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message.

- 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.

- Ha
- M

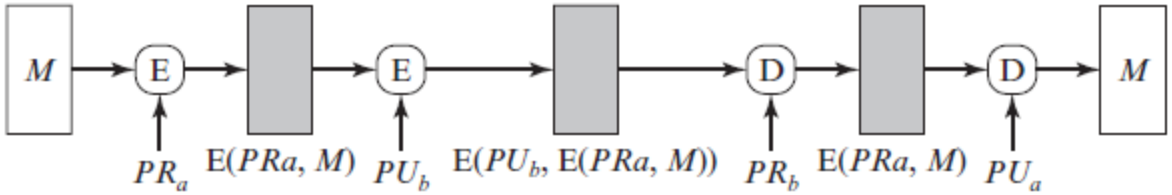


(a) Symmetric encryption: confidentiality and authentication



(b) Public-key encryption: confidentiality

(c) Public-key encryption: authentication and signature

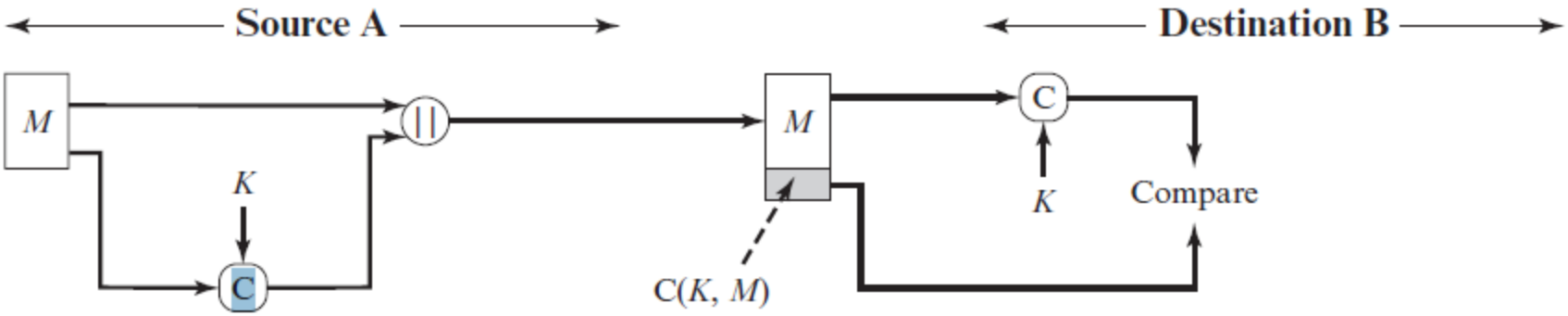


(d) Public-key encryption: confidentiality, authentication, and signature

- **Message Authentication Code**
  - An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a **cryptographic checksum** or MAC, that is appended to the message.
  - 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:

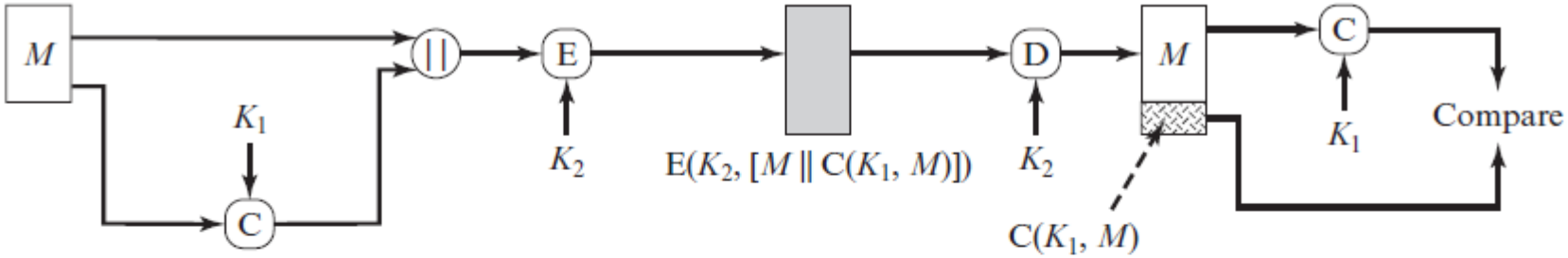
    MAC = C(*K*, *M*)
  - where
    - *M* = input message
    - C = MAC function
    - *K* = shared secret key
    - MAC = message authentication code

- 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

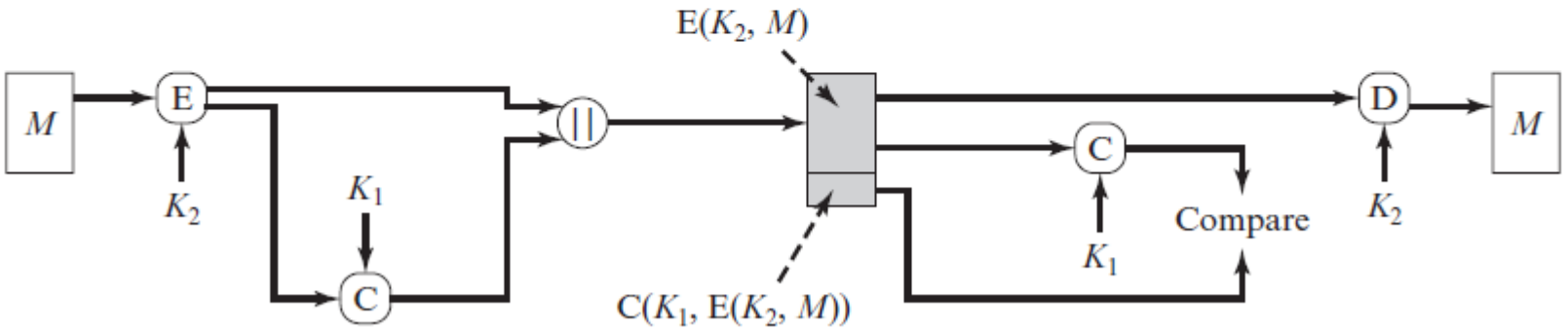


(a) Message authentication

- A MAC function is similar to encryption. One difference is that the MAC algorithm need not



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



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

Security of MACs and hash Function

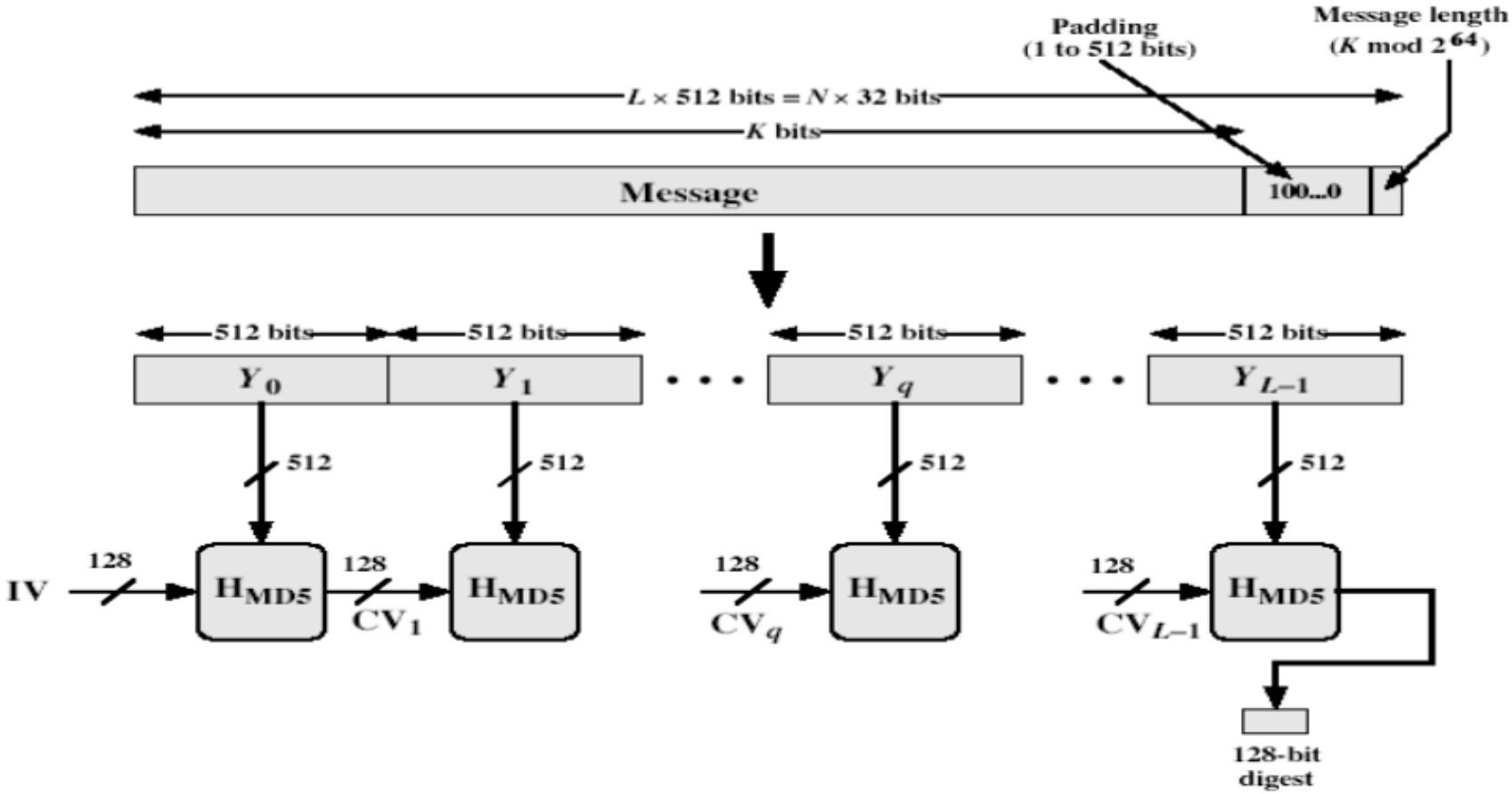
- **Brute-Force Attacks**
  - A brute-force attack on a MAC is a more difficult undertaking than a brute-force attack on a hash function because it requires known message-tag pairs.
  - To attack a hash code, we can proceed in the following way. Given a fixed message x with n-bit hash code $h = H(x)$, a brute-force method of finding a collision is to pick a random bit string y and check if $H(y) = H(x)$.

- If an attacker can determine the MAC key, then it is possible to generate a valid MAC value for any input *x*.

- **Cryptanalysis**
  - an ideal MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

MD5

- designed by Ronald Rivest (the R in RSA)
- latest in a series of MD2, MD4
- produces a 128-bit hash value

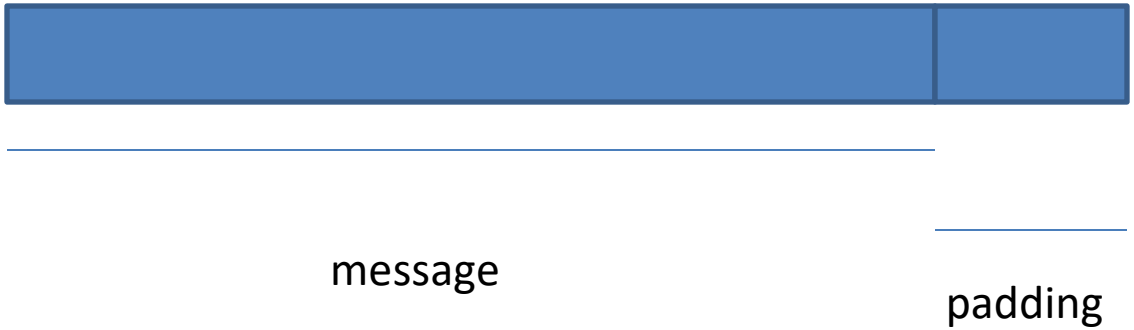


- Padding

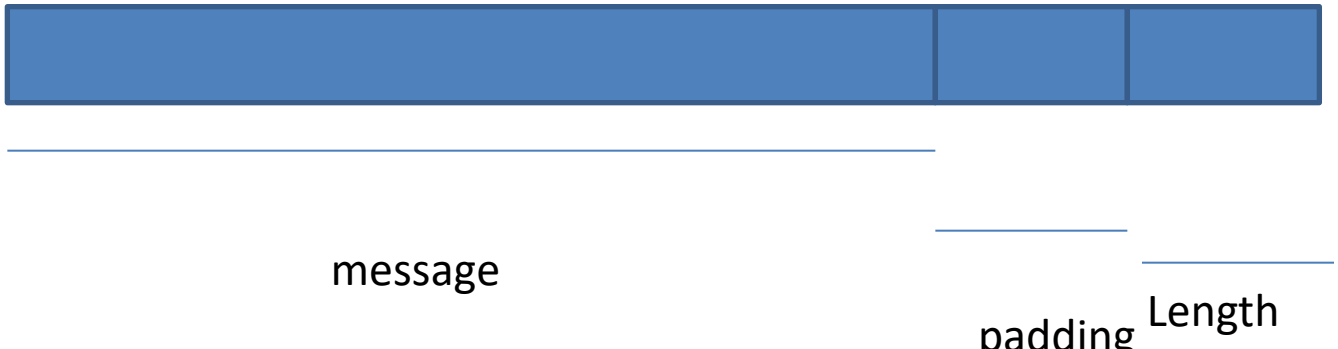
Each component of 512 bit will have 4 rounds of each 16 operations. Hence total 64 operation in each single 512 bit block

Steps

- **Padding**(to make multiple of 512)
- But the size should be multiple of 512 – 64.



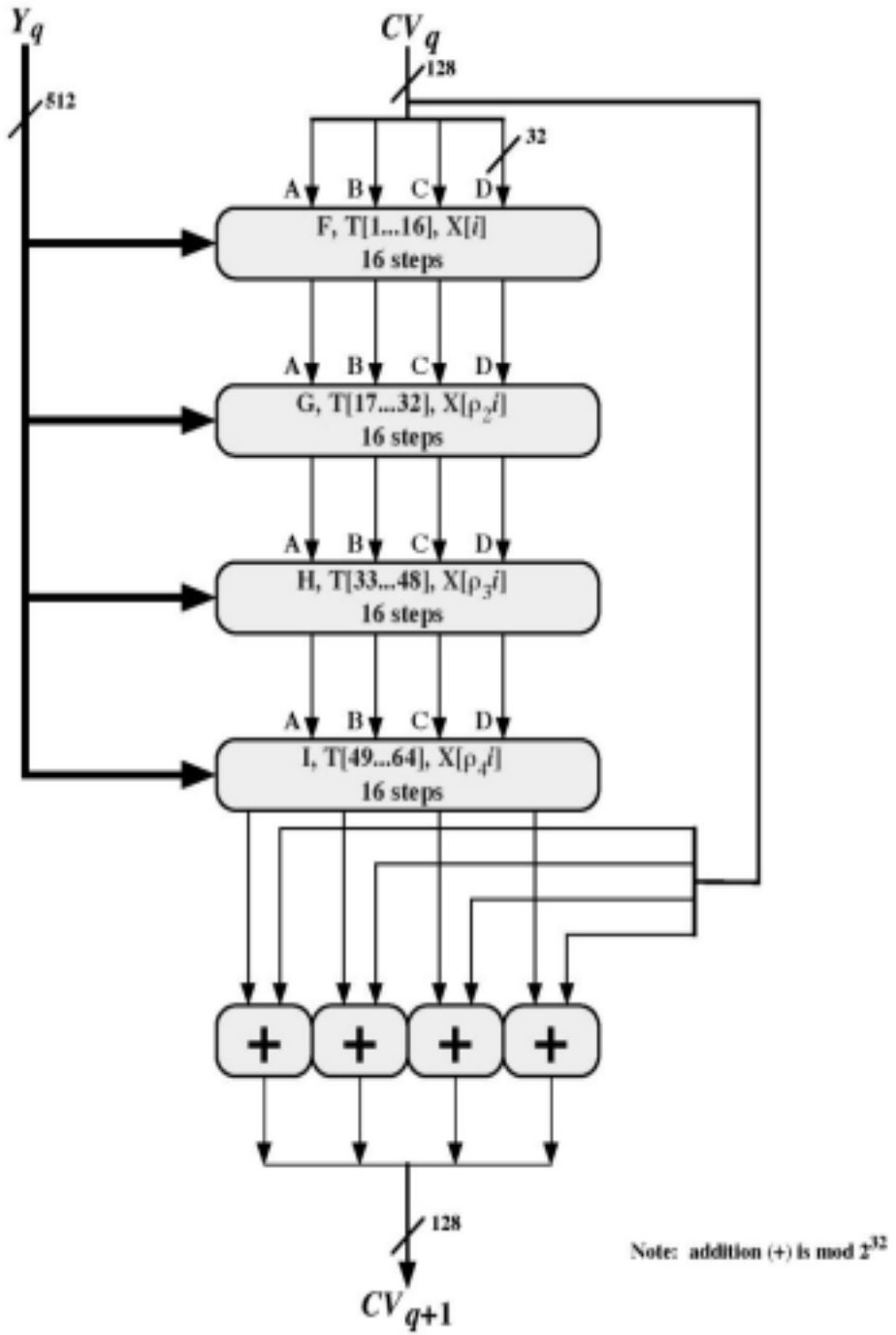
- II step Append length bits (64 bits)



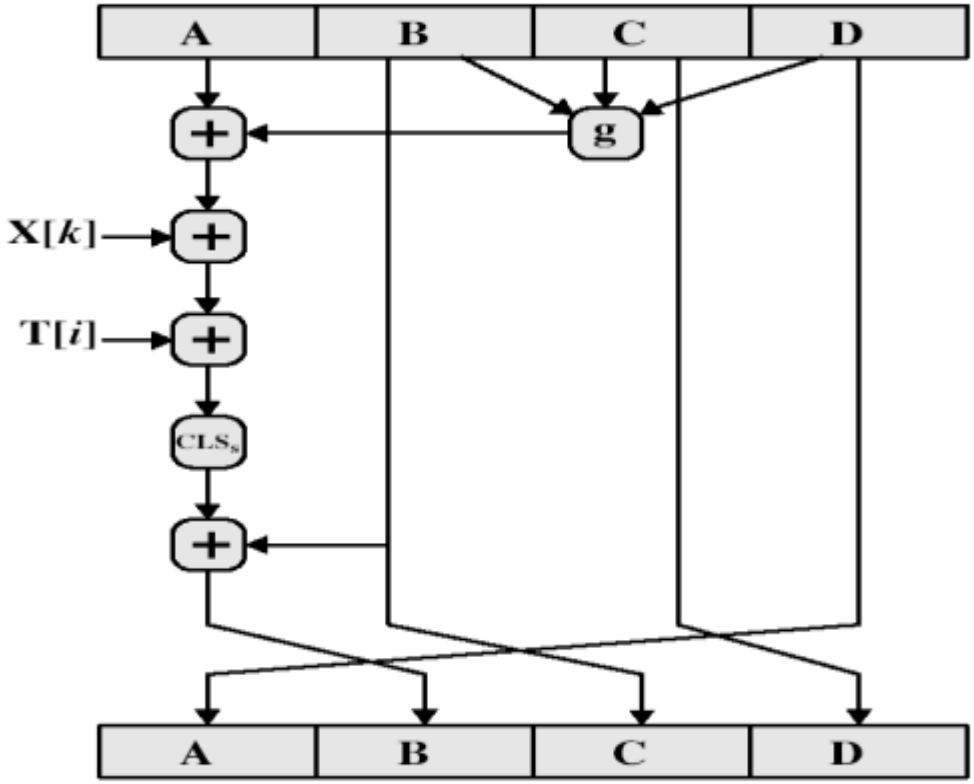
- Example:
- 1000bits
- Multiple of 512 minus 64.

- 1536 -64 = 1472
- 1472-1000 = **472** (padding bits)

- III step ir 2 bits)
  - A : 0 1
  - B: 8 9
  - C: f e d
  - D: 7 6 5
  - Now 64 eration total 64



- MD5 Compression function



- each round has 16 steps of the form:
  - a = b+((a+g(b,c,d)+X[k]+T[i])<<<s)
  - a,b,c,d refer to the 4 words of the buffer, but used in varying permutations
  - note this updates 1 word only of the buffer
  - after 16 steps each word is updated 4 times
  - where g(b,c,d) is a different nonlinear function in each round (F,G,H,I)
  - T[i] is a constant value
  - The point of all this complexity:
    - To make it difficult to generate collisions

SHA-512

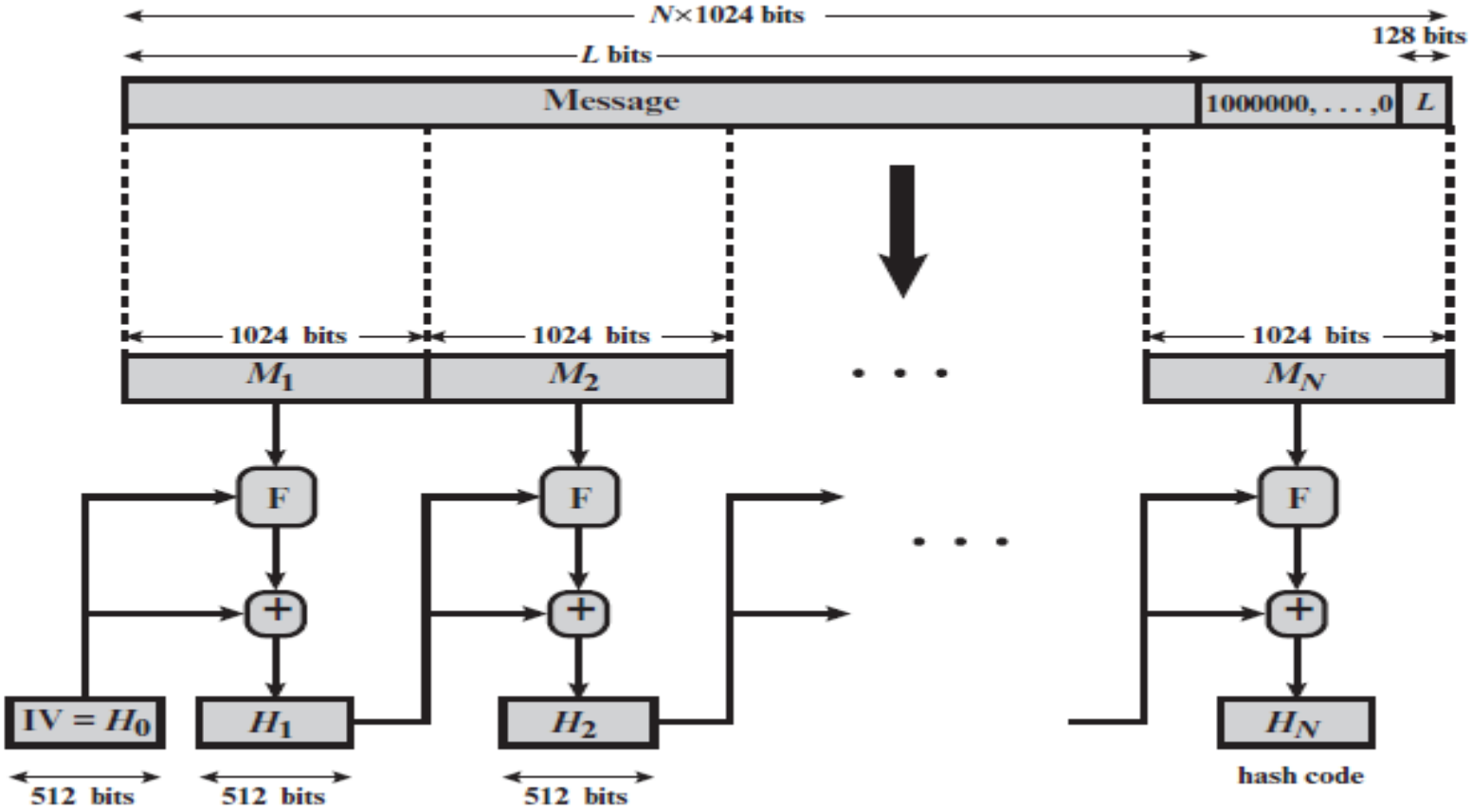
- the most widely used hash function has been the Secure Hash Algorithm (SHA).

- SHA was developed by the National Institute of Standards and Technology (NIST)

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

- The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest.

- The input is processed in 1024-bit blocks.



- **Step 1 Append padding bits.**
  - The message is padded so that its length is congruent to 896 modulo 1024 [length ≡ 896(mod 1024)].
  - Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 1024.
- **Step 2 Append length.**
  - A block of 128 bits is appended to the message.
  - This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the length of the original message in bits (before the padding).

- The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length.

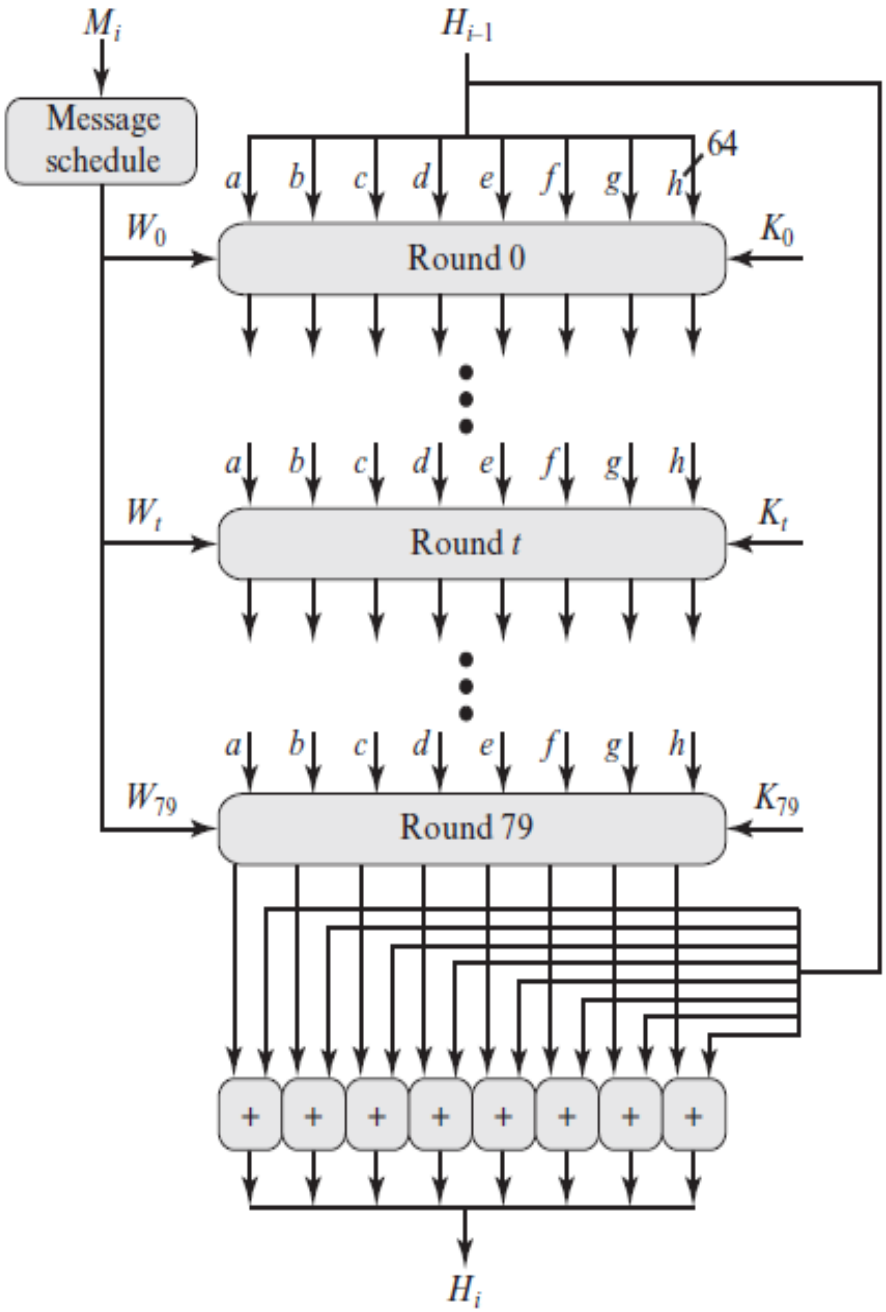
- **Step 3 Initialize hash buffer.**
  - A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h). These registers are initialized to the following 64-bit integers (hexadecimal values):

```
a = 6A09E667F3BCC908    e = 510E527FADE682D1
b = BB67AE8584CAA73B    f = 9B05688C2B3E6C1F
c = 3C6EF372FE94F82B    g = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1    h = 5BE0CD19137E2179
```

- **Step 4 Process message in 1024-bit (128-byte) blocks.**

  – The heart of the algorithm is a module that consists of 80 rounds;

  – Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer.



SHA-512 Processing of a Single 1024-Bit Block

- The output of the eightieth round is added to the input to the first round (H_{i-1}) to produce H_i.

- **Step 5 Output.**
  - After all *N* 1024-bit blocks have been processed, the output from the *N*th stage is the 512-bit message digest.

We can summarize the behavior of SHA-512 as follows:

$$H_0 = \text{IV}$$
$$H_i = \text{SUM}_{64}(H_{i-1}, \text{abcdefgh}_i)$$
$$MD = H_N$$

where

IV = initial value of the abcdefgh buffer, defined in step 3

abcdefgh_i = the output of the last round of processing of the ith message block

N = the number of blocks in the message (including padding and length fields)

SUM_{64} = addition modulo 2^{64} performed separately on each word of the pair of inputs

MD = final message digest value

$$T_1 = h + \mathrm{Ch}(e, f, g) + (\sum_1^{512} e) + W_t + K_t$$
$$T_2 = (\sum_0^{512} a) + \mathrm{Maj}(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

where

t = step number; $0 \leq t \leq 79$

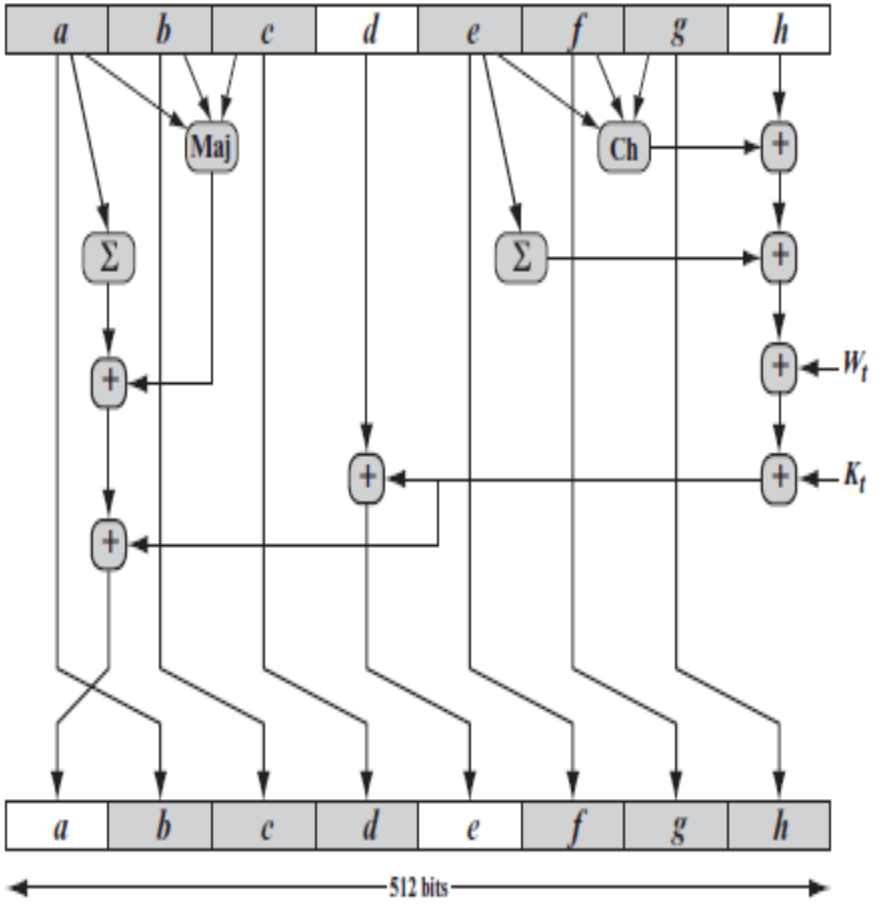
$\mathrm{Ch}(e, f, g)$ = $(e$ AND $f) \oplus ($NOT e AND $g)$
the conditional function: If e then f else g

$\mathrm{Maj}(a, b, c)$ = $(a$ AND $b) \oplus (a$ AND $c) \oplus (b$ AND $c)$
the function is true only of the majority (two or three) of the arguments are true

$(\Sigma_0^{512} a)$ = $\mathrm{ROTR}^{28}(a) \oplus \mathrm{ROTR}^{34}(a) \oplus \mathrm{ROTR}^{39}(a)$

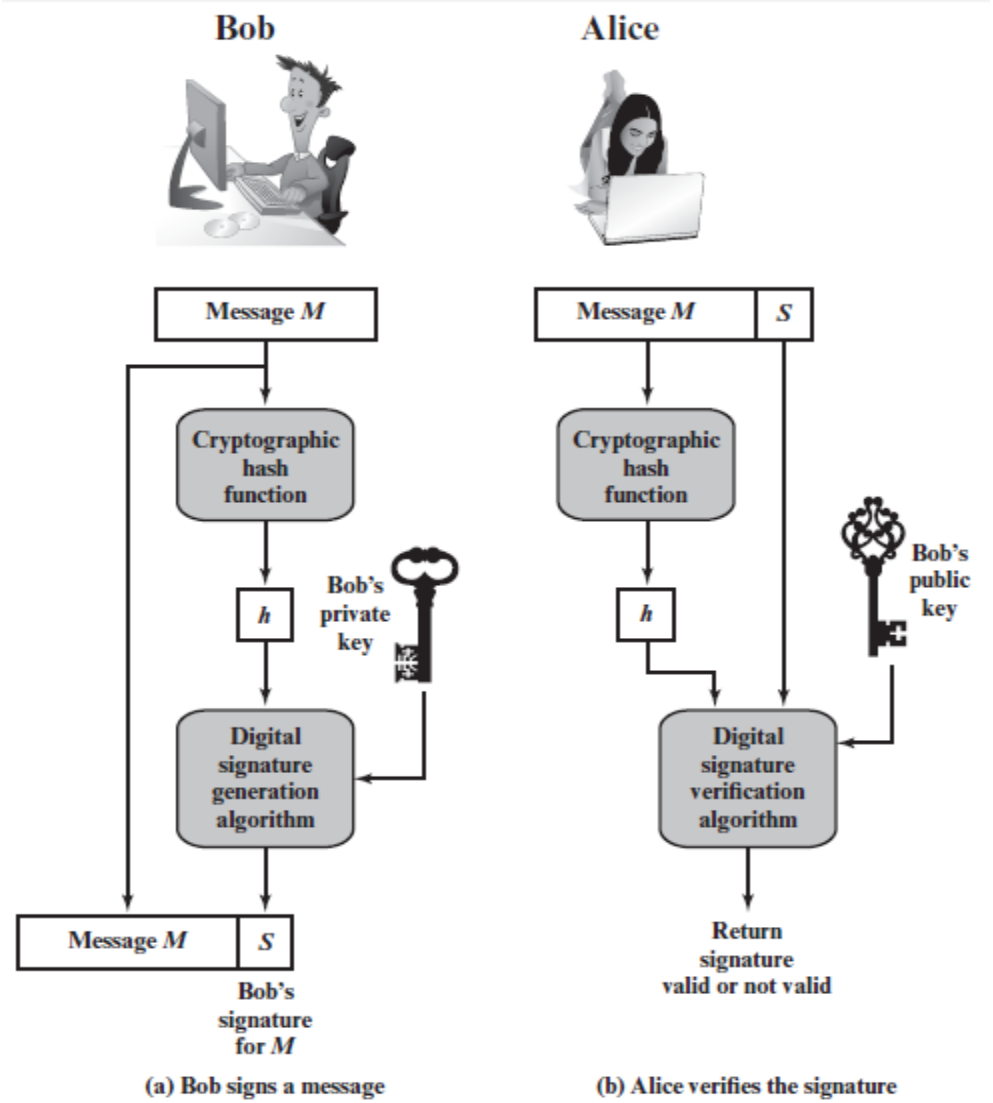
$(\Sigma_1^{512} e)$ = $\mathrm{ROTR}^{14}(e) \oplus \mathrm{ROTR}^{18}(e) \oplus \mathrm{ROTR}^{41}(e)$

$\mathrm{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits


a
b
c
d
e
f
g
h
Maj
Ch
Σ
Σ
+
W_t
K_t
512 bits

Digital signature

- Figure depicts a generic model of the process of constructing and using digital signatures.



- **Properties**
  - 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.

- **1.** Mary may forge a different message and claim that it came from John. Mary would simply have to create a message and append an authentication code using the key that John and Mary share.

- **2.** John can deny sending the message. Because it is possible for Mary to forge a message, there is no way to prove that John did in fact send the message.

- 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.

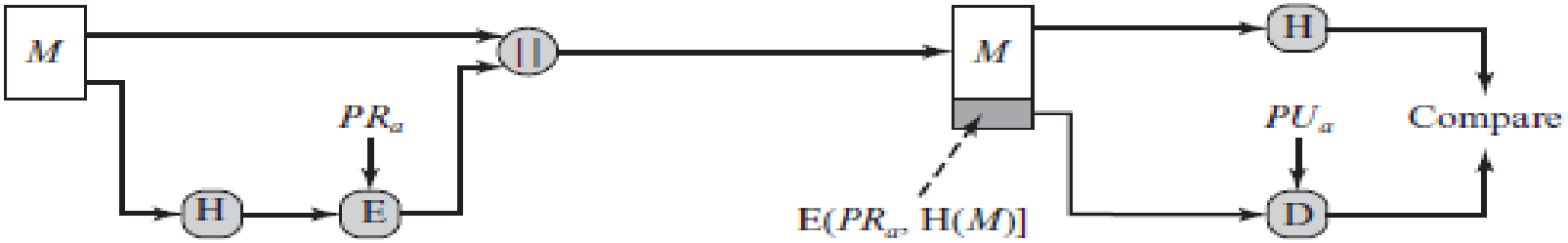
- **Digital Signature Requirements:** On the basis of the properties, we can formulate the following requirements for a digital signature.
  - The signature must be a bit pattern that depends on the message being signed.
  - The signature must use some information only known to the sender to prevent both forgery and denial.
  - It must be relatively easy to produce the digital signature.
  - It must be relatively easy to recognize and verify the digital signature.
  - It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
  - It must be practical to retain a copy of the digital signature in storage.

- A secure hash function, embedded in a scheme such as that of Figure above, provides a basis for satisfying these requirements.

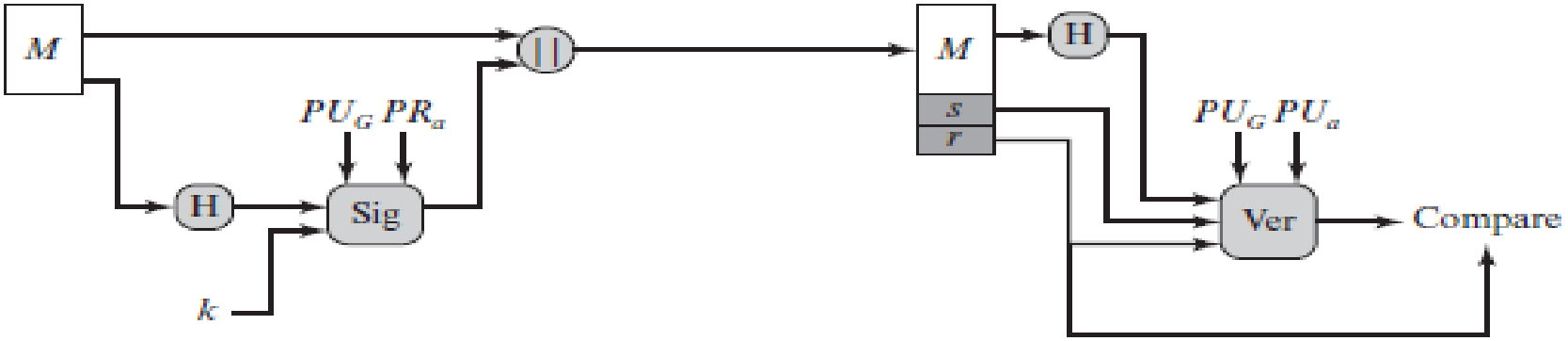
- The DSA makes use of the Secure Hash Algorithm (SHA).

- 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.

- Nevertheless, it is a public-key technique.



(a) RSA approach

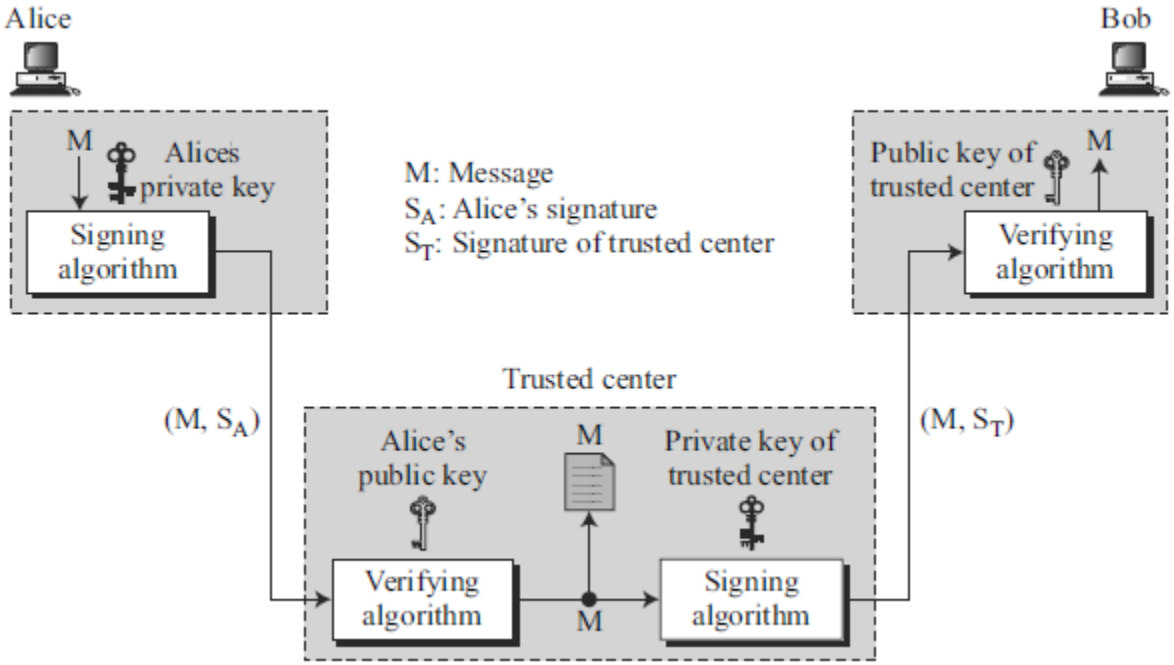


(b) DSA approach

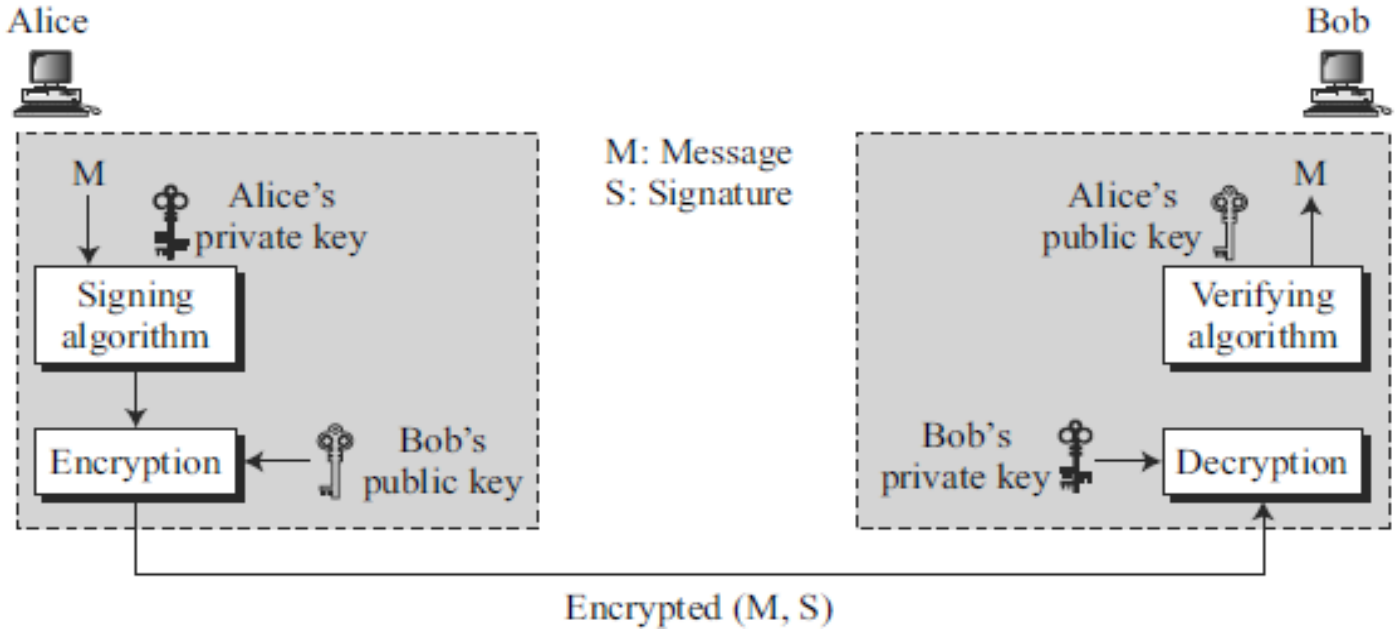
- A digital signature needs a public-key system.
- The signer signs with her private key; the verifier verifies with the signer’s public key.
- A cryptosystem uses the private and public keys of the receiver: a digital signature uses the private and public keys of the sender.

- Services of DS
  - Message Authentication
  - Message Integrity
  - Nonrepudiation

- Nonrepudiation can be provided using a trusted party.



- Confidentiality
  - A digital signature does not provide confidential communication.
  - If confidentiality is required, the message and the signature must be encrypted using either a secret-key or public-key cryptosystem.



DSA

- DSS uses a digital signature algorithm (DSA).
- The DSA approach 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.

Global Public-Key Components

p prime number where $2^{L-1} < p < 2^L$
for $512 \leq L \leq 1024$ and L a multiple of 64;
i.e., bit length L between 512 and 1024 bits
in increments of 64 bits

q prime divisor of $(p - 1)$, where $2^{N-1} < q < 2^N$
i.e., bit length of N bits

g $= h(p - 1)/q$ is an exponent mod p,
where h is any integer with $1 < h < (p - 1)$
such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x random or pseudorandom integer with $0 < x < q$

User's Public Key

$y = g^x \bmod p$

User's Per-Message Secret Number

k random or pseudorandom integer with $0 < k < q$

Signing

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

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

Signature $= (r, s)$

Verifying

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

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

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

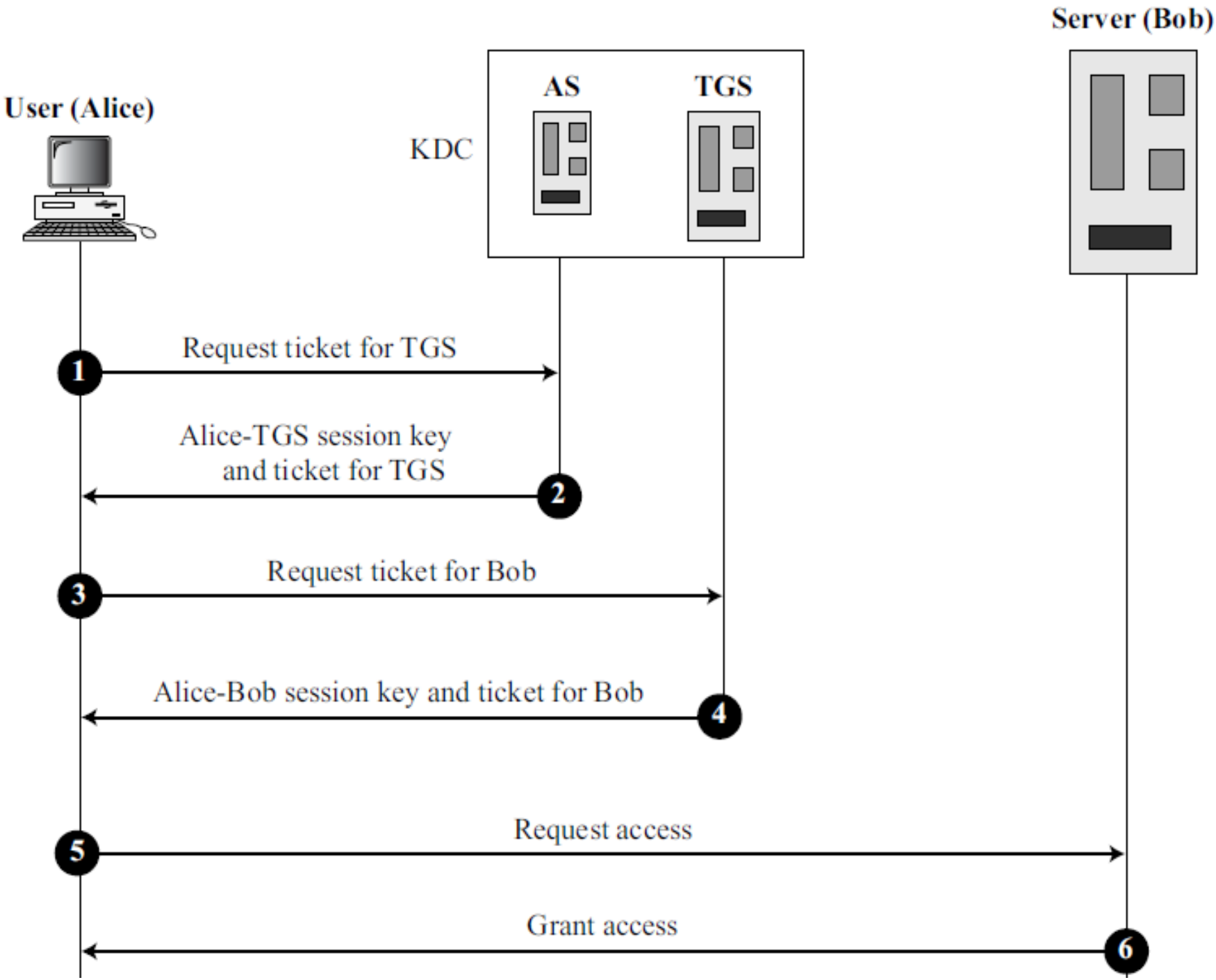
$v = [(g^{u1}y^{u2}) \bmod p] \bmod q$

TEST: $v = r'$

M = message to be signed
H(M) = hash of M using SHA-1
M', r', s' = received versions of M, r, s

Authentication Protocol: **Kerberos**

- Kerberos is an authentication protocol, and at the same time a KDC, that has become very popular.
- It is named after the three-headed dog in Greek mythology that guards the gates of Hades.
- Originally designed at MIT, it has gone through several versions.
- We'll cover version 4 and its difference with version 5.


Server (Bob)
AS
TGS
User (Alice)
KDC
Request ticket for TGS
1
Alice-TGS session key
and ticket for TGS
2
Request ticket for Bob
3
Alice-Bob session key and ticket for Bob
4
Request access
5
Grant access
6

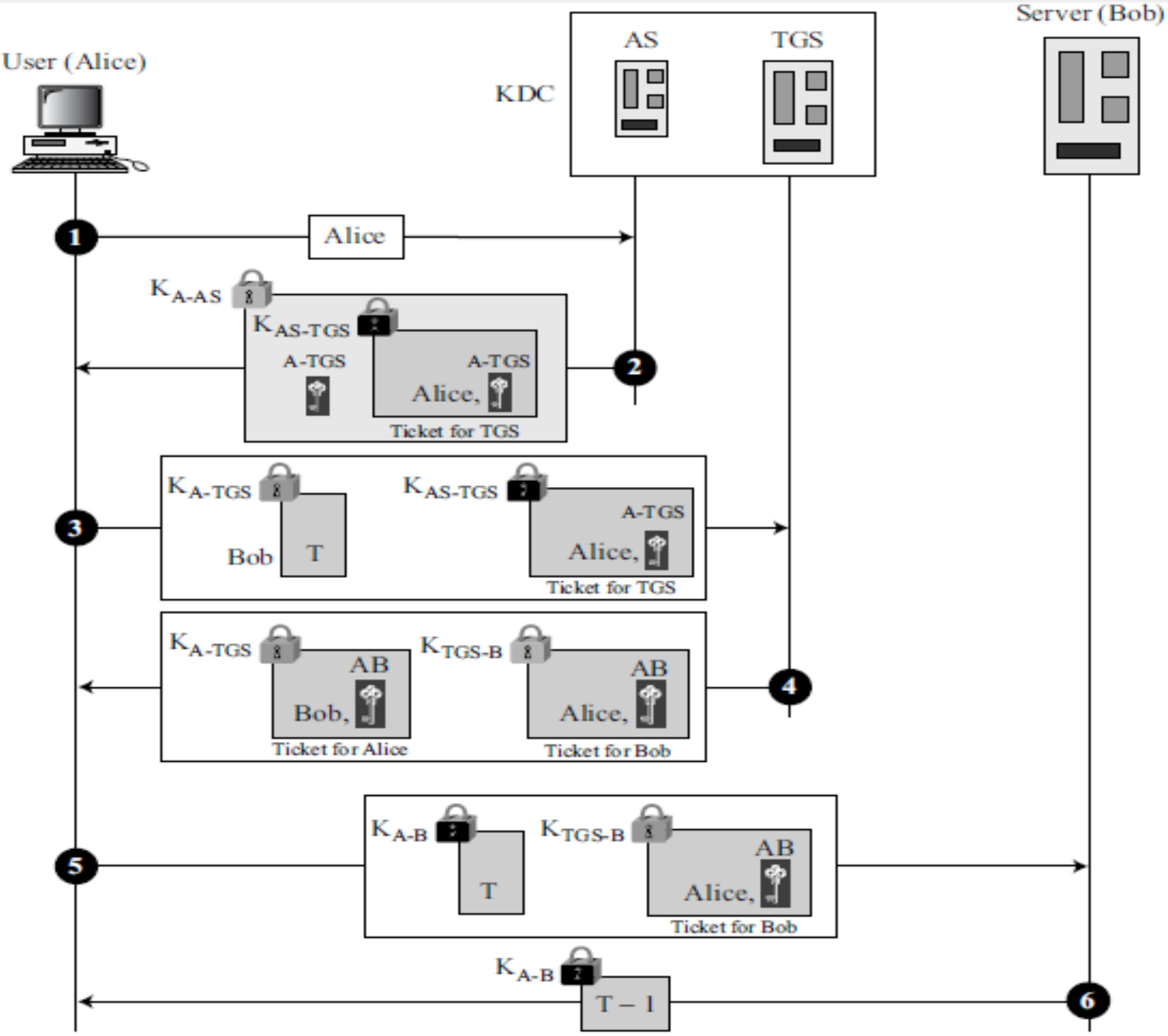
- Three servers are involved in the Kerberos protocol: an authentication server (AS), a ticket-granting server (TGS), and a real (data) server that provides services to others.

- In our examples and figures, Bob is the real server and Alice is the user requesting service.

- Authentication Server (AS)
  - The authentication server (AS) is the KDC in the Kerberos protocol.
  - Each user registers with the AS and is granted a user identity and a password.
  - The AS has a database with these identities and the corresponding passwords.
  - The AS verifies the user, issues a session key to be used between Alice and the TGS, and sends a ticket for the TGS.

- Ticket-Granting Server (TGS)
  - The ticket-granting server (TGS) issues a ticket for the real server (Bob).
  - It also provides the session key (K_{AB}) between Alice and Bob. Kerberos has separated user verification from the issuing of tickets.
  - In this way, though Alice verifies her ID just once with the AS, she can contact the TGS multiple times to obtain tickets for different real servers.

- Real Server
  - The real server (Bob) provides services for the user (Alice). Kerberos is designed for a client-server program, such as FTP, in which a user uses the client process to access the server process.
  - Kerberos is not used for person-to-person authentication.


User (Alice)
KDC
AS
TGS
Server (Bob)
1
Alice
K_{A-AS}
K_{AS-TGS}
A-TGS
A-TGS
Alice,
Ticket for TGS
2
3
K_{A-TGS}
Bob
T
K_{AS-TGS}
A-TGS
Alice,
Ticket for TGS
4
K_{A-TGS}
AB
Bob,
Ticket for Alice
K_{TGS-B}
AB
Alice,
Ticket for Bob
5
K_{A-B}
T
K_{TGS-B}
AB
Alice,
Ticket for Bob
K_{A-B}
T – 1
6

- Kerberos Version 5
  - The minor differences between version 4 and version 5 are briefly listed below:
  - 1. Version 5 has a longer ticket lifetime.
  - 2. Version 5 allows tickets to be renewed.
  - 3. Version 5 can accept any symmetric-key algorithm.
  - 4. Version 5 uses a different protocol for describing data types.
  - 5. Version 5 has more overhead than version 4.

- Realms
  - Kerberos allows the global distribution of ASs and TGSs, with each system called a realm.
  - A user may get a ticket for a local server or a remote server.