# Message Authentication and Hash Functions

Jibi Abraham

COEP TECHNOLOGICAL UNIVERSITY

Shivajinagar, Pune-411 005

(A Unitary Technological University of Govt. of Maharashtra)

# What is Authentication?

- A procedure to verify that received messages come from the alleged source and have not been altered
  - Digital Signature is one of the techniques including a countermeasure of repudiation by either source or destination

# Authentication Requirements

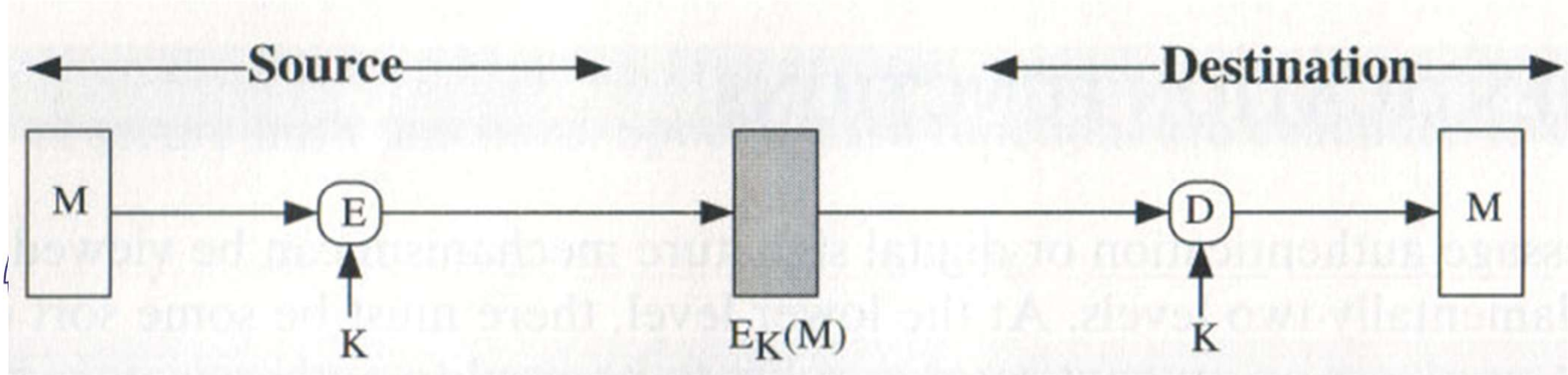
- Possible attacks
  1. Disclosure
  2. Traffic Analysis
  3. Masquerade
  4. Content Modification
  5. Sequence Modification
  6. Timing Modification
  7. Repudiation: source and destination repudiation
- Attacks#1-2 -> Confidentiality
- Attacks#3-7 -> Authentication
  - Especially #7 is related to Digital Signature

# Authentication Functions

- 3 Types of cryptographic operations related to authentication:
  - Message Encryption
  - Message Authentication Code (MAC)
  - Hash Function

# Message Encryption

- Conventional encryption provides a weak form of authentication
- If Bob can recover a message encrypted with a shared key between Alice and Bob, Bob knows that Alice sent this message
- If the message has been altered, Bob would not be able to read it
- But if the plaintext is not intelligible, it is difficult for receiver to ensure that cipher text after decryption is compromised or not



# Confidentiality and Authentication Implications of Message Encryption

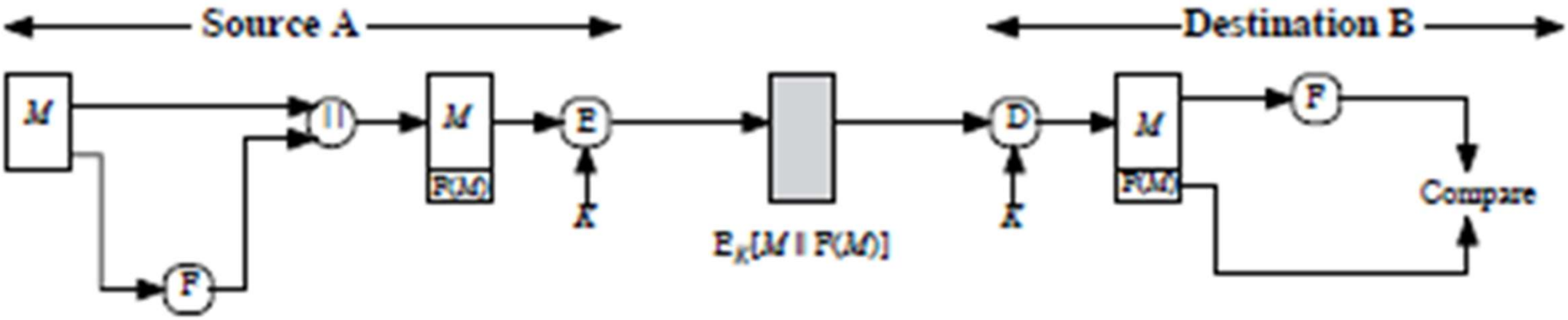
## (a) Conventional (symmetric) Encryption

$A \rightarrow B: E_K[M]$

- Provides confidentiality
  - —Only A and B share K
- Provides a degree of authentication
  - —Could come only from A
  - —Has not been altered in transit
  - —Requires some formatting/redundancy
- Does not provide signature
  - —Receiver could forge message
  - —Sender could deny message

# Checksum with Encryption

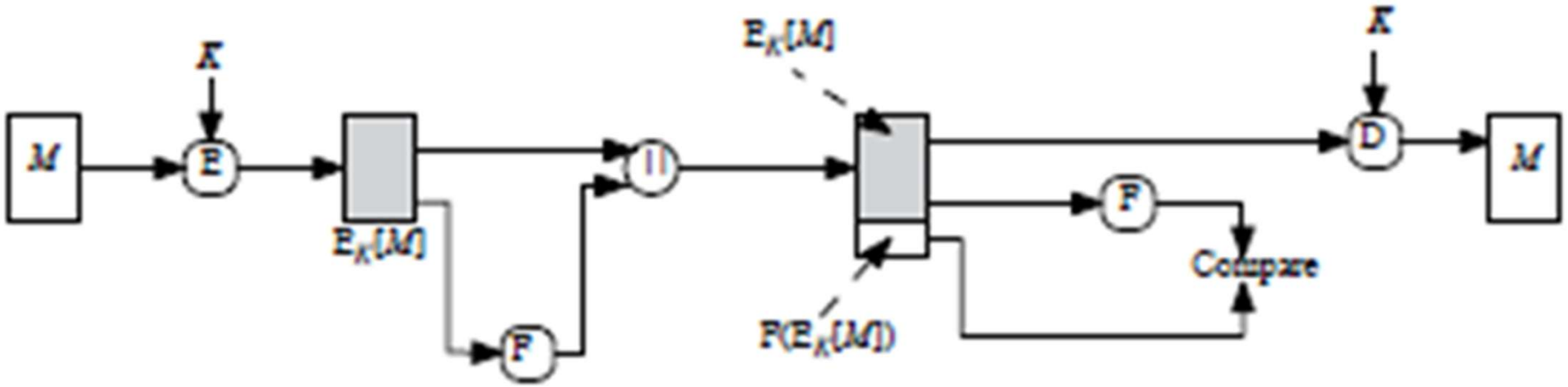
- If the message has a suitable structure, redundancy or a checksum to detect any changes
- With Internal  error control




(a) Internal error control

# Checksum with Encryption

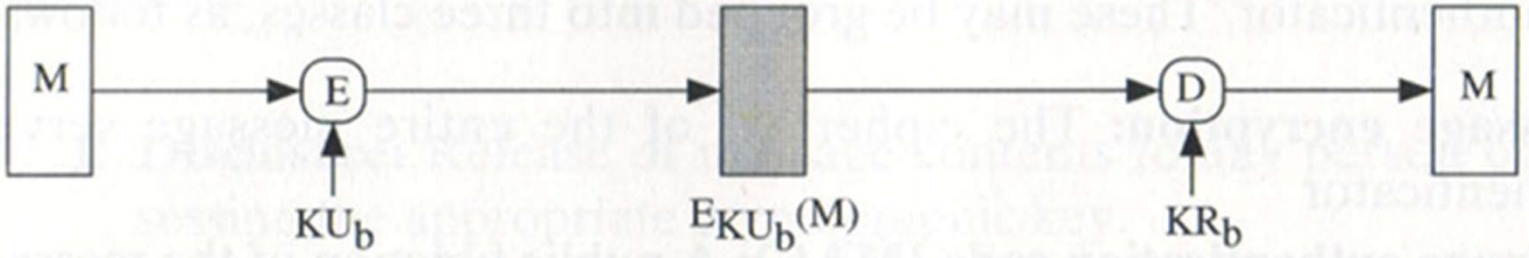
- With External error control




(b) External error control

# Public-key Encryption

- Encryption provides no confidence of sender since anyone potentially knows public-key

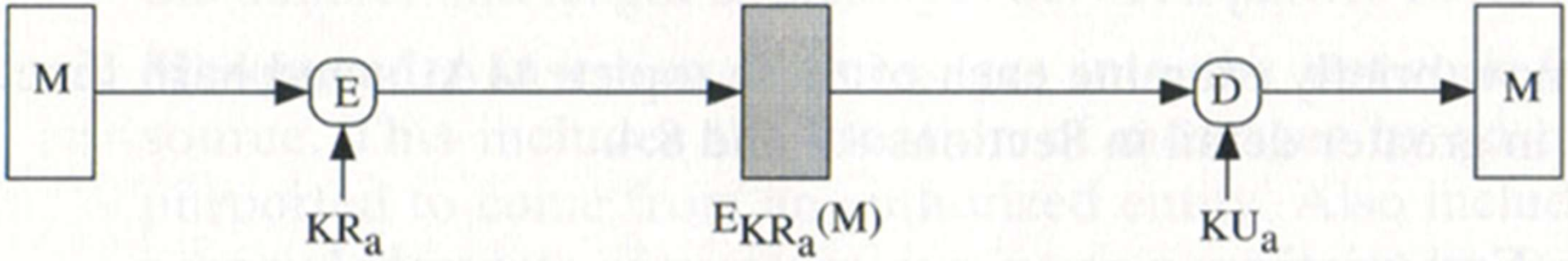



$A \rightarrow B: E_{KU_b}[M]$

- Provides confidentiality
  - —Only B has $KR_b$ to decrypt
- Provides no authentication
  - —Any party could use $KU_b$ to encrypt message and claim to be A

# Public-key Encryption

- However, if sender **signs** message using their private-key
- Anyone can read the message



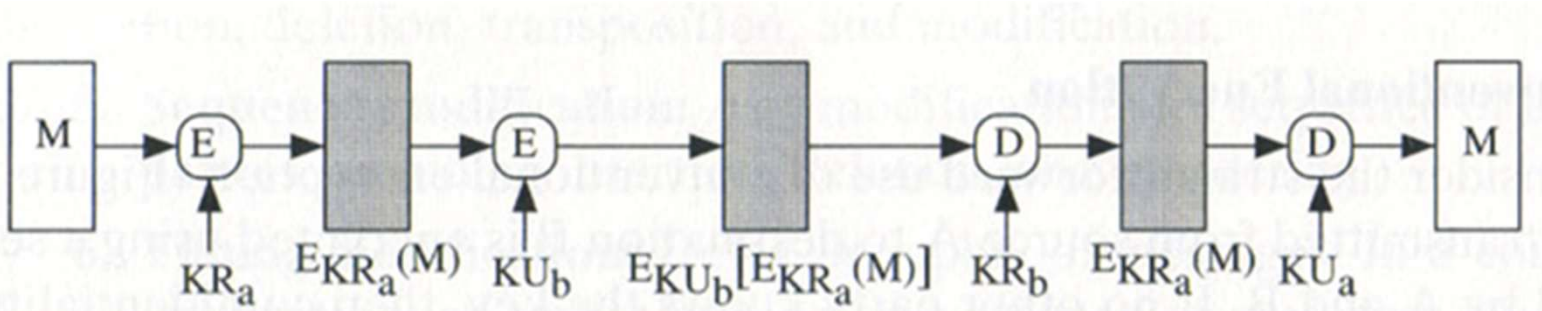

(c) Public-key encryption: authentication and signature

$A \rightarrow B: E_{KR_a}[M]$

- Provides authentication and signature
  - —Only A has $KR_a$ to encrypt
  - —Has not been altered in transit
  - —Requires some formatting/redundancy
  - —Any party can use $KU_a$ to verify signature

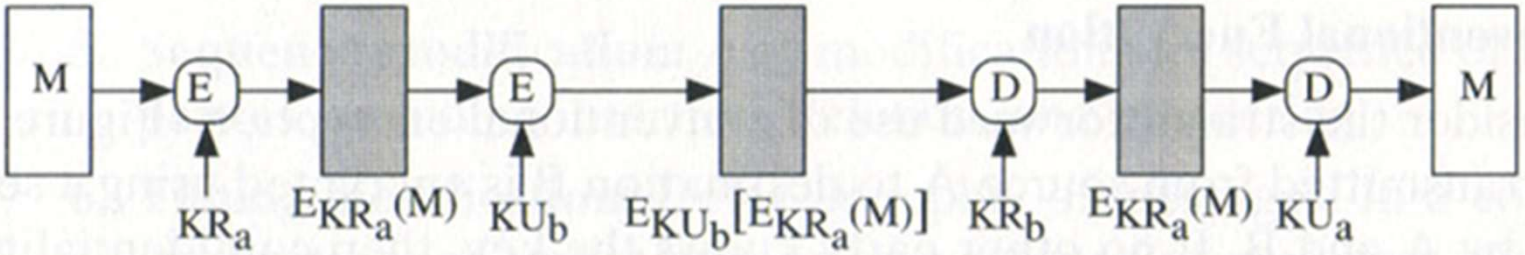
# Public-key Encryption

- If sender **signs** message using their private-key, then encrypts with recipient's public key
- have both secrecy and authentication
- Again, need to recognize corrupted messages
- But at the cost of two public-key uses on a message




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

# Public-key Encryption




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

$A \rightarrow B: E_{KU_b}[E_{KR_a}(M)]$

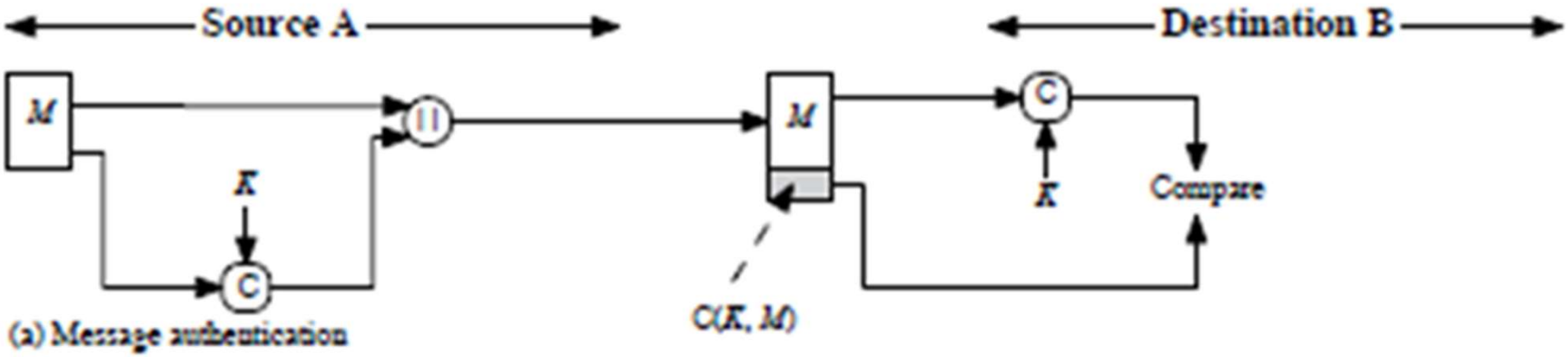
- Provides confidentiality because of $KU_b$
- Provides authentication and signature because of $KR_a$

# Hash and MAC Algorithms

- Hash Functions
  - condense arbitrary size message to fixed size
  - by processing message in blocks
  - through some compression function
  - either custom or block cipher based
- Message Authentication Code (MAC)
  - fixed sized authenticator for some message
  - to provide authentication for message
  - by using block cipher mode or hash function

# Message Authentication Code (MAC)

- Generated by an algorithm that creates a small fixed-sized block
  - Depending on both the message and some key
  - Like encryption, though need not be reversible
- Appended to the message as a **signature**
- receiver performs same computation on message and checks it matches the MAC
- MAC is irreversible, but encryption isn't.
- provides assurance that message is unaltered and comes from sender



# Message Authentication Codes (MACs)

- MAC involves the use of a secret key to generate a small fixed-size block of data.
- known as a cryptographic checksum: MAC = $C_K(M)$

  where M is a variable-length message, K is a secret key shared between sender and receiver, and $C_K$ is fixed-length authenticator
- MAC is appended to the message and sent over to receiver.
- is a many-to-one function
  - potentially many messages have same MAC
  - but finding these needs to be very difficult

# Message Authentication Code

1. Alice and Bob share the secret K1.
2. Alice calculates $MAC_1 = C_{K1}(M)$

   Alice→Bob: $\{M, MAC_1\}$
3. Bob calculates $MAC_2 = C_{K1}(M)$

   If $MAC_2 = MAC_1$, M is sent from Alice and not altered

- Confidentiality can be provided by encryption with another shared key.

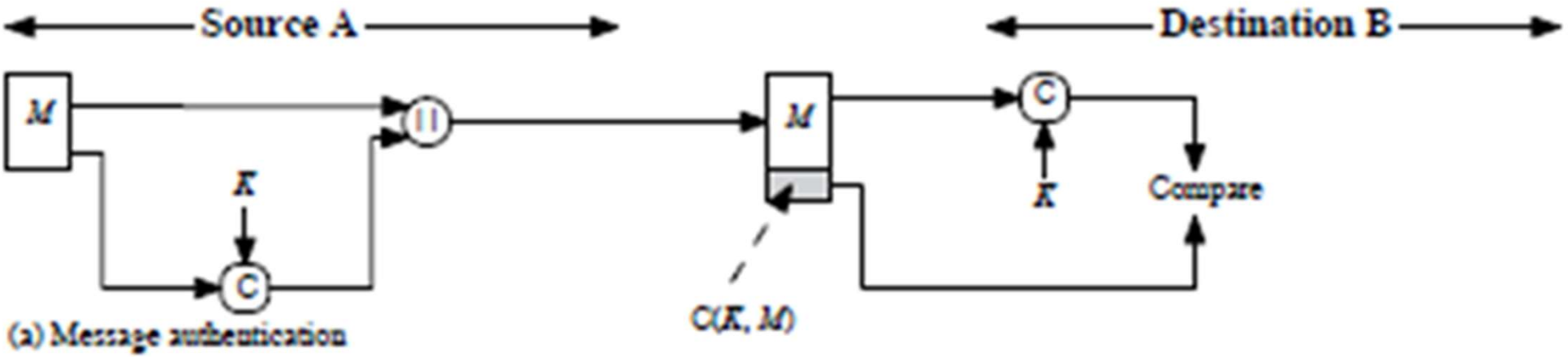
  Alice→Bob: $\{M, MAC_1\}_{K2}$

# MAC applications

- Why MAC is required, where conventional encryption could provide authentication?
  - Same message need to be broadcasted to many destinations
  - Heavy messages, can not afford encryption
  - To preserve the integrity of a computer program or massages like SNMP, better MAC than encrypt and save
  - To prolong the period of protection

# Basis uses of MAC

- To Provide authentication

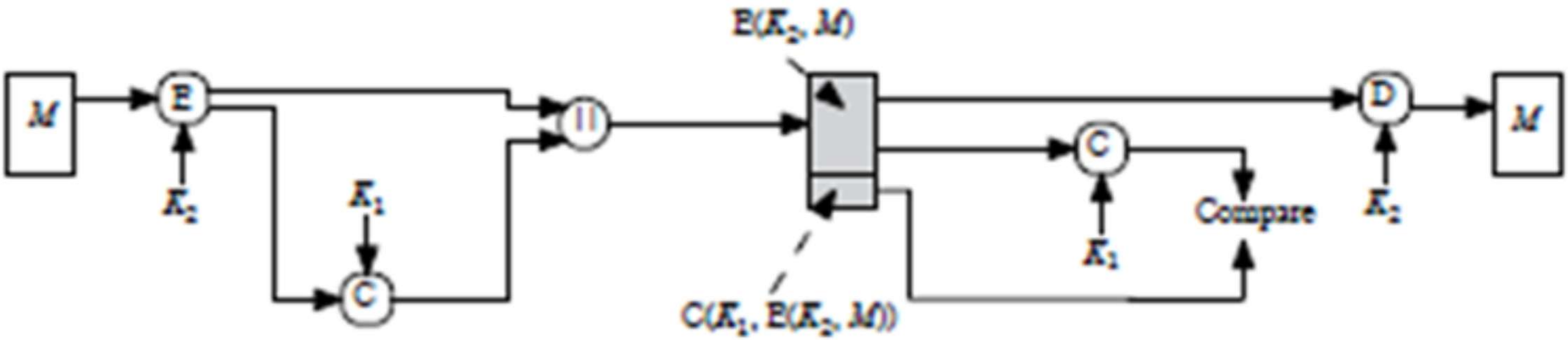


# Basis uses of MAC

- Message authentication and confidentiality
- Authentication tied to plaintext

# Basis uses of MAC

- Message authentication and confidentiality
- Authentication tied to ciphertext

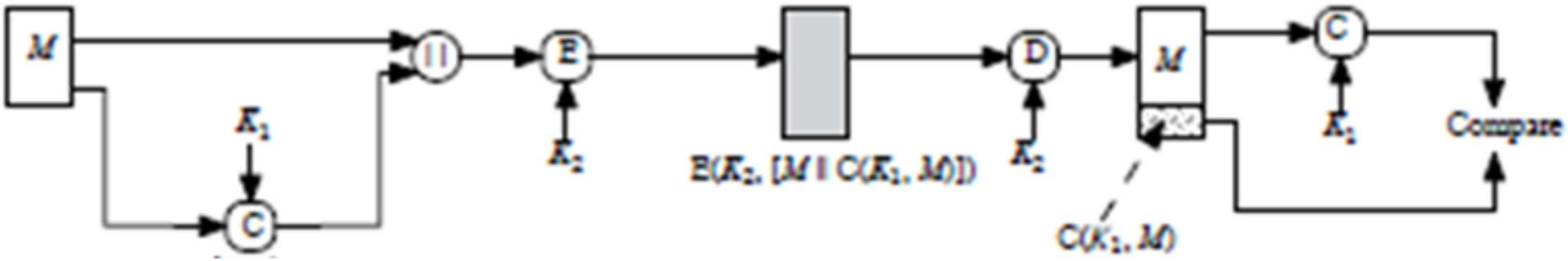


# Requirements for MACs

- Taking into account the types of attacks
- need the MAC to satisfy the following:
  1. knowing a message and MAC, it is infeasible to find another message with same MAC
  2. MACs should be uniformly distributed
  3. MAC should depend equally on all bits of the message

# Brute Force Attack on MAC

- MAC functions are many to one
- n bit MAC, $2^n$ possible MACs, N possible messages, $N >> 2^n$
- Potentially many messages have same MAC



# Brute Force Attack on MAC

- Brute force attack to recover key
  - Suppose key size k > n MAC size
  - Round1:
    - Cryptanalyst perform $MAC_1 = C_{K1}(M_1)$ for all possible K1
    - $2^{(k-n)}$ keys produce same MAC
  - Round 2:
    - Cryptanalyst perform $MAC_2 = C_{K2}(M_2)$ for remaining $2^{(k-n)}$
    - $2^{(k-2n)}$ produces same match
  - On average $\alpha$ rounds required, where $k = \alpha$ X n
  - More harder than decryption

# Security of MACs

- like block ciphers have brute-force attacks exploiting
  - strong collision resistance hash have cost $2^{m/2}$
    - 128-bit hash looks vulnerable, 160-bits better
  - MACs with known message-MAC pairs
    - can either attack keyspace (cf. key search) or MAC
    - at least 128-bit MAC is needed for security

# Properties of MAC function

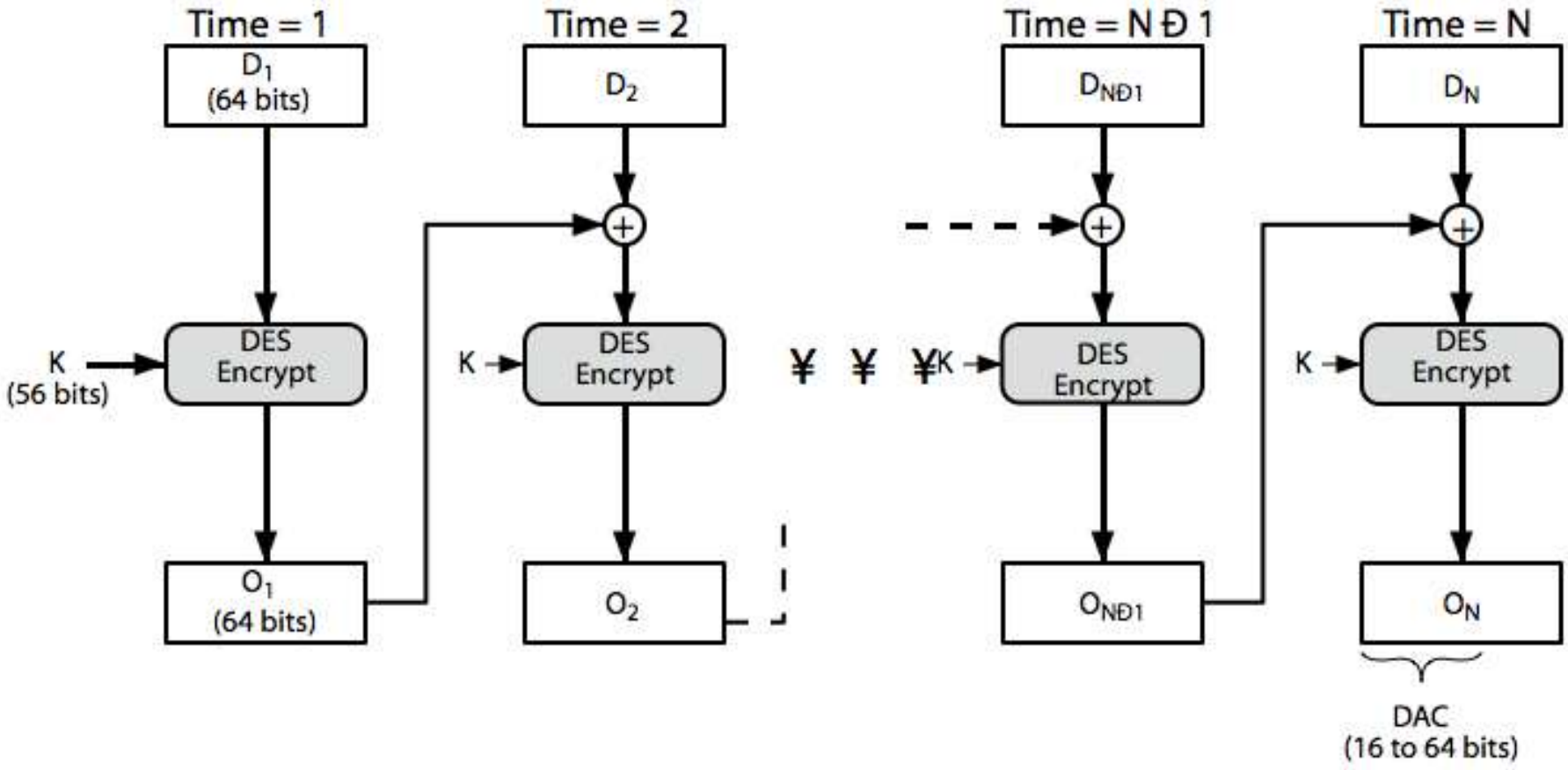
- Simple attack
  - $\Delta(M) = x_1 \oplus x_2 \oplus \ldots \oplus x_m$, $C_K(M) = E_K(\Delta(M))$
  - Intruder replaces $X_1$ through $X_{m-1}$ with $Y_1$ through $Y_{m-1}$ and $X_m$ with $Y_m$, where

    $$y_m = y_1 \oplus y_2 \oplus \ldots \oplus y_{m-1} \oplus \Delta(M)$$
- Properties
  - Computationally infeasible to construct a message M' to replace M, where $C_K(M')=C_K(M)$
  - Randomly chosen messages M and M', probability of $C_K(M')=C_K(M)$ is $2^{-n}$, n is MAC size
  - If M'=f(M), probability of $C_K(M')=C_K(M)$ is $2^{-n}$, n is MAC size

# Using Symmetric Ciphers for MACs

- can use any block cipher chaining mode and use final block as a MAC
- **Data Authentication Algorithm (DAA)** is a widely used MAC based on DES-CBC
  - using IV=0 and zero-pad of final block
  - encrypt message using DES in CBC mode
  - and send just the final block as the MAC
    - or the leftmost M bits (16≤M≤64) of final block
- but final MAC is now too small for security

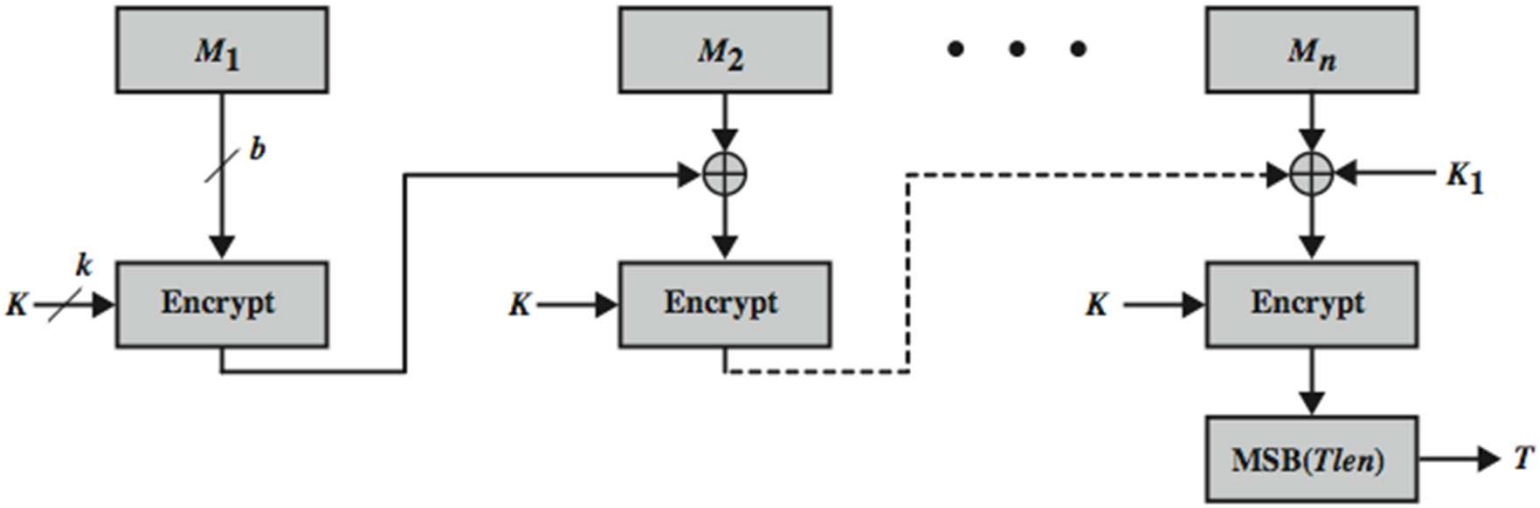
# Data Authentication Algorithm



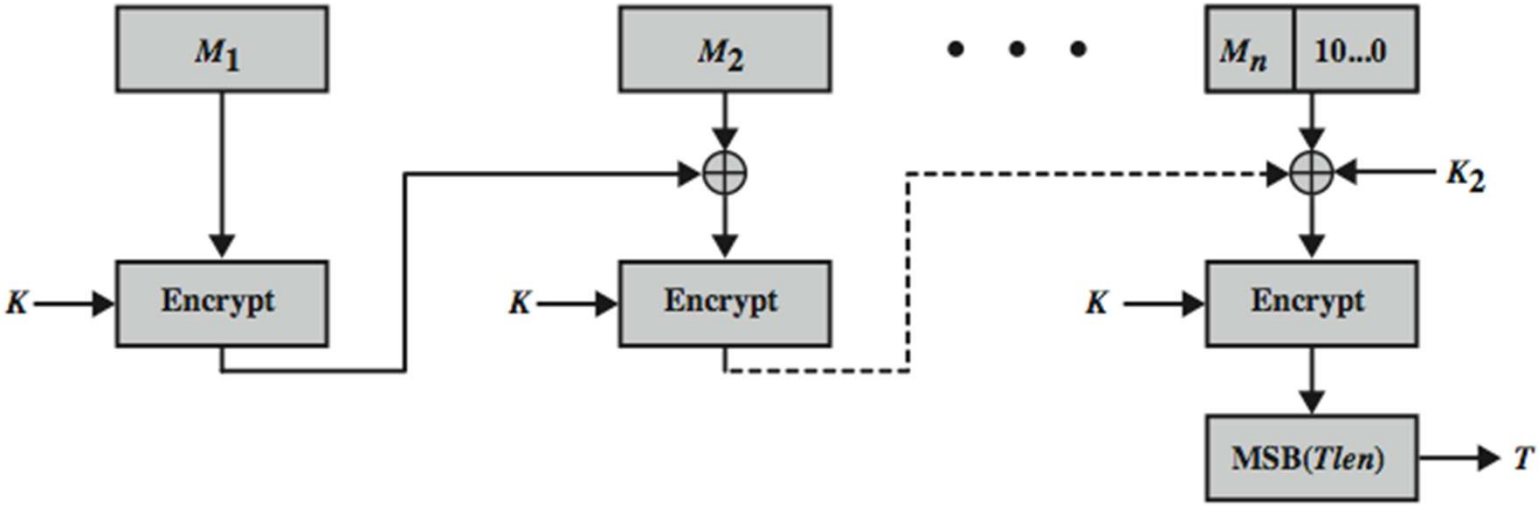
# CMAC

- DAA was widely used in govt and industry, but has message size limitation
- Can overcome using 2 keys and padding, thus forming the Cipher-based Message Authentication Code (CMAC)
- adopted by NIST SP800-38B
- Uses the blocksize of the underlying cipher (ie 128-bits for AES or 64-bits for triple-DES).

# CMAC Overview




(a) Message length is integer multiple of block size




(b) Message length is not integer multiple of block size

# CMAC

a) If the size of the input message block is equal to a positive multiple of the block size (namely, 128 bits), the last block shall be exclusive-OR'ed with K1 before processing

b) Otherwise, the last block shall be padded with 10^i and exclusive-OR'ed with K2.

- K1 is the subkey for the case (a), and
- K2 is the subkey for the case (b).
- K1 and K2 are generated by the subkey generation algorithm from the key K

# HMAC

- Original proposal:

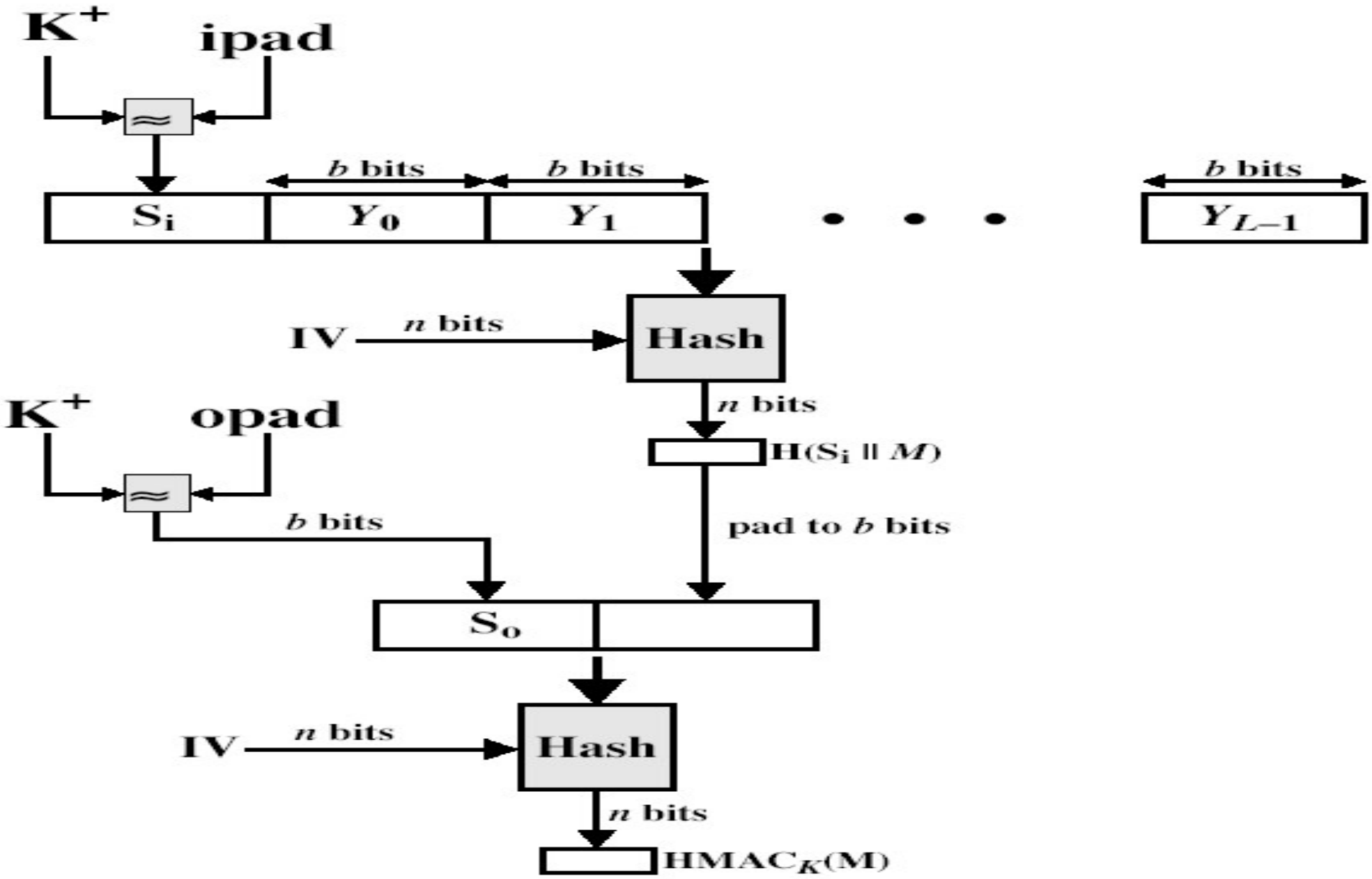
  `KeyedHash = Hash(Key|Message)`

  – extension attack
- Eventually led to development of HMAC (RFC2104)

$$HMAC_K = Hash[(K^+ \text{ XOR } opad) \,||\, Hash[(K^+ \text{ XOR } ipad)||M)]]$$

- where $K^+$ is the key padded out to size
- opad, ipad are specified padding constants
- overhead is just 2 more hash iterations than the message needs alone
- any of MD5, SHA-1, RIPEMD-160 can be used

# HMAC Overview

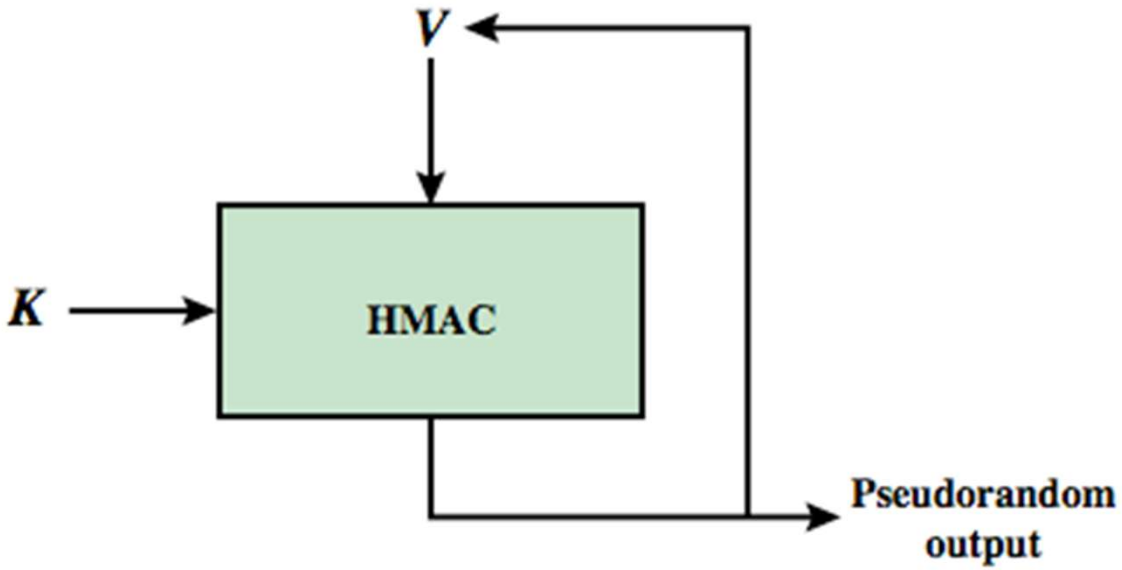


# HMAC Security

- Security of HMAC relates to that of the underlying hash algorithm
- Attacking HMAC requires either:
  - Brute force attack on the key used
  - Birthday attack (but since keyed would need to observe a huge number of messages)
- Choose a hash function based on speed versus security constraints

# PRNG using a MAC

- MAC PRNGs in SP800-90, IEEE 802.11i, TLS
  - use key
  - input based on last hash in various ways



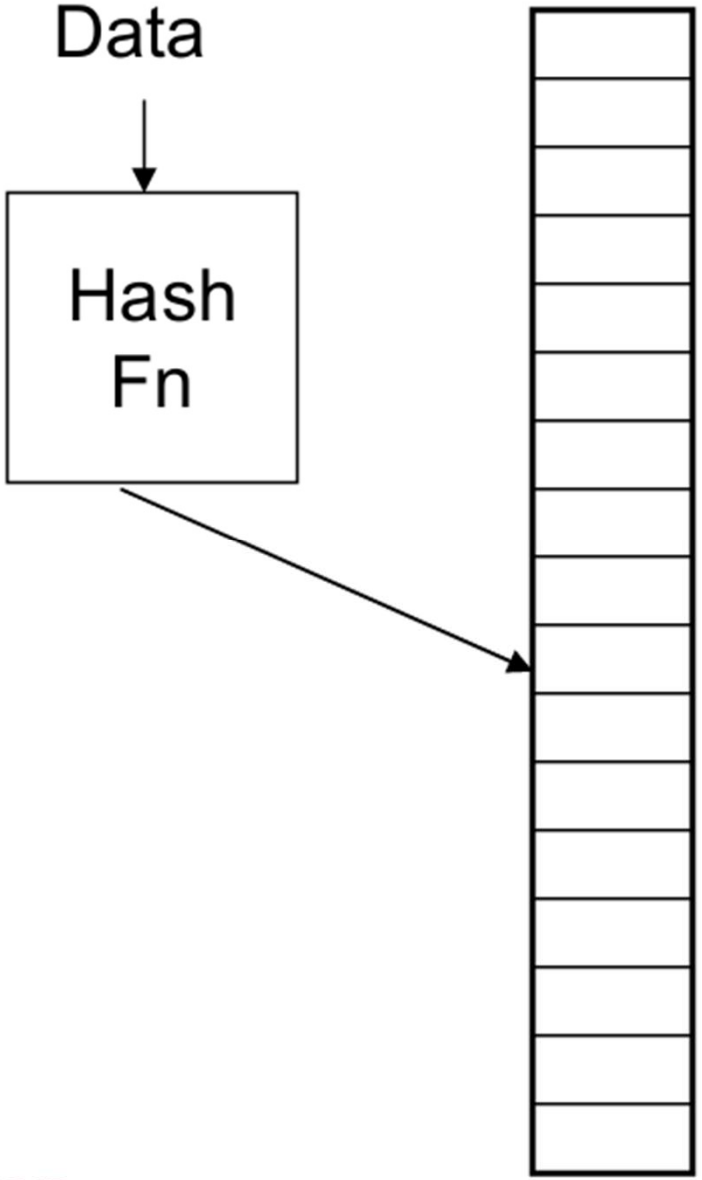

(b) PRNG using HMAC

# Hash Algorithms

- A **hash algorithm** is a function that converts input data of any size into a fixed-size string of characters , known as the hash value or hash code, is unique to each unique input
- Widely used in various applications, including data storage, retrieval, and security
- Hash is used to detect changes to message
- A hash function is public and is not keyed
- Can use in various ways, with message most often to create a digital signature

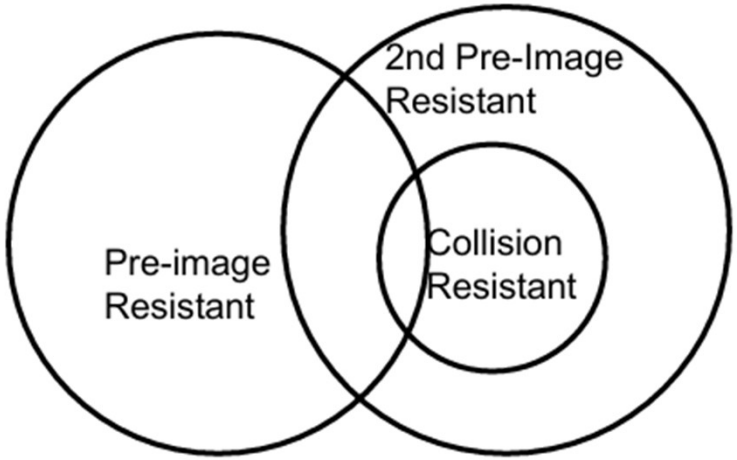
# Hash Function

- Hash tables used in data searches
- Hash function should
  - Take variable size input
  - Produce fixed output size (Size of the table)
  - Be easy to compute
  - Be pseudorandom so that it distributes uniformly over the table
  - Minimizes collisions



# Cryptographic Hash Functions

- Variable size input M
- Fixed size output h
- Efficient computation
- Pseudorandom
- Pre-image Resistant = one-way
  - Given h, it is not possible to find M,
- 2nd Pre-image Resistant: = Weak Collision Resistant
  - h(x) is known, it is not possible to find a y, such that h(y)= h(x)
- Strong Collision Resistant:
  - It is not possible to find any two x and y, such that h(y)=h(x)



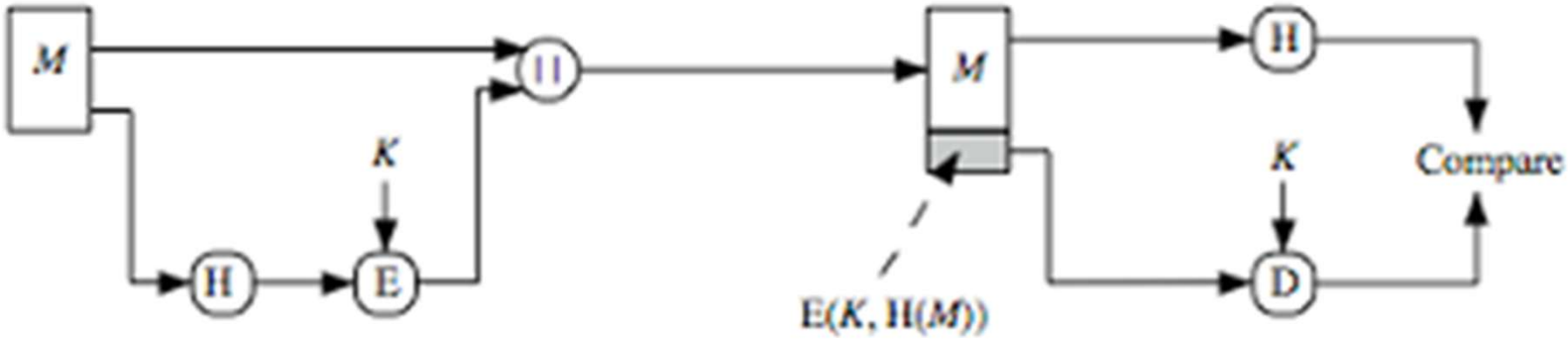
# Examples of Crypto Hash Functions

- MD4 = Message Digest 4 [RFC 1320] →128bit hash
  - Fast on 32bit machines
- MD5 = Message Digest 5 [RFC 1321] →128bit hash
  - 32bit operations (but differs in internal structure from MD4)
- SHA = Secure hash algorithm [NIST]
- SHA-1 = Updated SHA
- SHA-2 = SHA-224, SHA-256, SHA-384, SHA-512
  - SHA-512 uses 64-bit operations
- SHA-3 in 2015: SHA3-224, SHA3-256, SHA3-384, and SHA3-512

# Applications of Crypto Hash Function

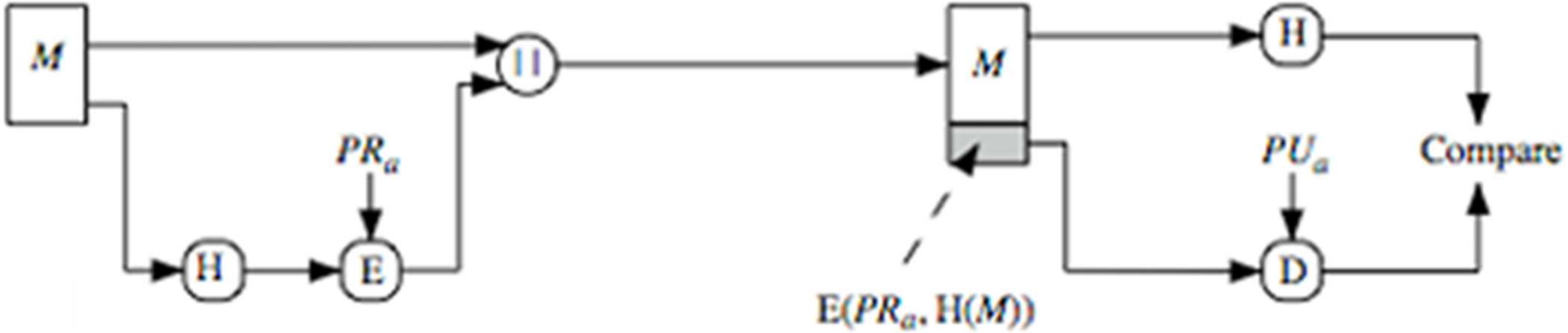
1. Message Authentication = Integrity
   - Can encrypt Message, hash, or both for confidentiality


   - Only hash is encrypted. Reduces the processing burden for application who donot require message confidentiality



# Message Authentication

## 2. Digital Signature




## 3. Digital Signature and confidentiality

# Applications of Crypto Hash Function

4. Password storage: Hash of the user's password is compared with that in the storage. Hackers can not get password from storage
5. Pseudorandom number generation: Hash an IV, Hash the hash, …, repeat

# Hash Function Properties

- Hash Function produces a fingerprint of some file/message/data `h = H(M)`
- Condenses a variable-length message M to a fixed-sized fingerprint
- Assumed that the algorithm is a known public

# Requirements for Hash Functions

1. Can be applied to any sized message M
2. Produces fixed-length output h
3. It is easy to compute h=H(M) for any message M
4. One-way property
   - given h is infeasible to find x such that H(x)=h
5. Weak collision resistance
   - given x is infeasible to find y such that H(y)=H(x)
6. Strong collision resistance
   - is infeasible to find any x, y such that H(y)=H(x)

# Simple Hash Function

| | bit 1 | bit 2 | • • • | bit *n* |
|---|---|---|---|---|
| block 1 | $b_{11}$ | $b_{21}$ | | $b_{n1}$ |
| block 2 | $b_{12}$ | $b_{22}$ | | $b_{n2}$ |
| | • | • | • | • |
| | • | • | • | • |
| | • | • | • | • |
| block *m* | $b_{1m}$ | $b_{2m}$ | | $b_{nm}$ |
| hash code | $C_1$ | $C_2$ | | $C_n$ |

- For normal text file, high order bit of each octant is zero. So 128 bits hash code, 16 bits will have zero. So effectiveness is $2^{-112}$ instead of $2^{-128}$
- One-bit circular shift on the hash value after each block is processed would improve

# Birthday Problem

- What is the probability that two people have the same birthday (day and month)?
- there are 365 possible birthdays
- $1^{st}$ person can have 365 days
- $2^{nd}$ person can have 364 days
- **365 × 364** represents the number of ways two people can have different birthdays

# Birthday Problem

- What is the probability that k people do not have the same birthday (day and month)?

| k | Total | Number of ways different people having different birthday |
|---|---|---|
| 2 | $365^2$ | $365 \times 364$ |
| 3 | $365^3$ | $365 \times 364 \times 363$ |
| | | $\cdots$ |
| k | $365^k$ | $365 \times 364 \times 363 \times \cdots \times (365 - k + 1)$ |

$$P(\text{No common day}) = \frac{365 \times 364 \times 363 \times \ldots \times (365 - k + 1)}{365^k}$$

$$= \frac{365!}{365^k (365 - k)!}$$

# Birthday Problem (Cont)

- With 22 people in a room, there is better than 50% chance that two people have a common birthday
- With 40 people in a room there is almost 90% chance that two people have a common birthday
- If there k people, there are k(k-1)/2 pairs

$$\mathrm{P}(1 \text{ pair having common birthday}) = \frac{k(k-1)}{2 \times 365}$$

$$k \geq \sqrt{365} \Rightarrow P > 0.5$$

- In general, n possibilities
  - $\sqrt{n}$ trials to find a collision

| k | P |
|---|---|
| 2 | .01 |
| 3 | .02 |
| 4 | .03 |
| ... | ... |
| 19 | .41 |
| 20 | .44 |
| 21 | .48 |
| 22 | .51 |
| 23 | .54 |
| ... | ... |
| 38 | .88 |
| 39 | .89 |
| 40 | .90 |

# Birthday Attack- A letter with Variations

Dear Anthony,

{This letter is / I am writing} to introduce {you to / to you} {Mr. / --} Alfred {P. / --}
Barton, the {new / newly appointed} {chief / senior} jewellery buyer for {our / the}
Northern {European / Europe} {area / division}. He {will take / has taken} over {the / --}
responsibility for {all / the whole of} our interests in {watches and jewellery / jewellery and watches}
in the {area / region}. Please {afford / give} him {every / all the} help he {may need / needs}
to {seek out / find} the most {modern / up to date} lines for the {top / high} end of the
market. He is {empowered / authorized} to receive on our behalf {samples / specimens} of the
{latest / newest} {watch and jewellery / jewellery and watch} products, {up / subject} to a {limit / maximum}
of ten thousand dollars. He will {carry / hold} a signed copy of this {letter / document}
as proof of identity. An order with his signature, which is {appended / attached}
{authorises / allows} you to charge the cost to this company at the {above / head office}
address. We {fully / --} expect that our {level / volume} of orders will increase in
the {following / next} year and {trust / hope} that the new appointment will {be / prove}
{advantageous / an advantage} to both our companies.

# Birthday Attacks

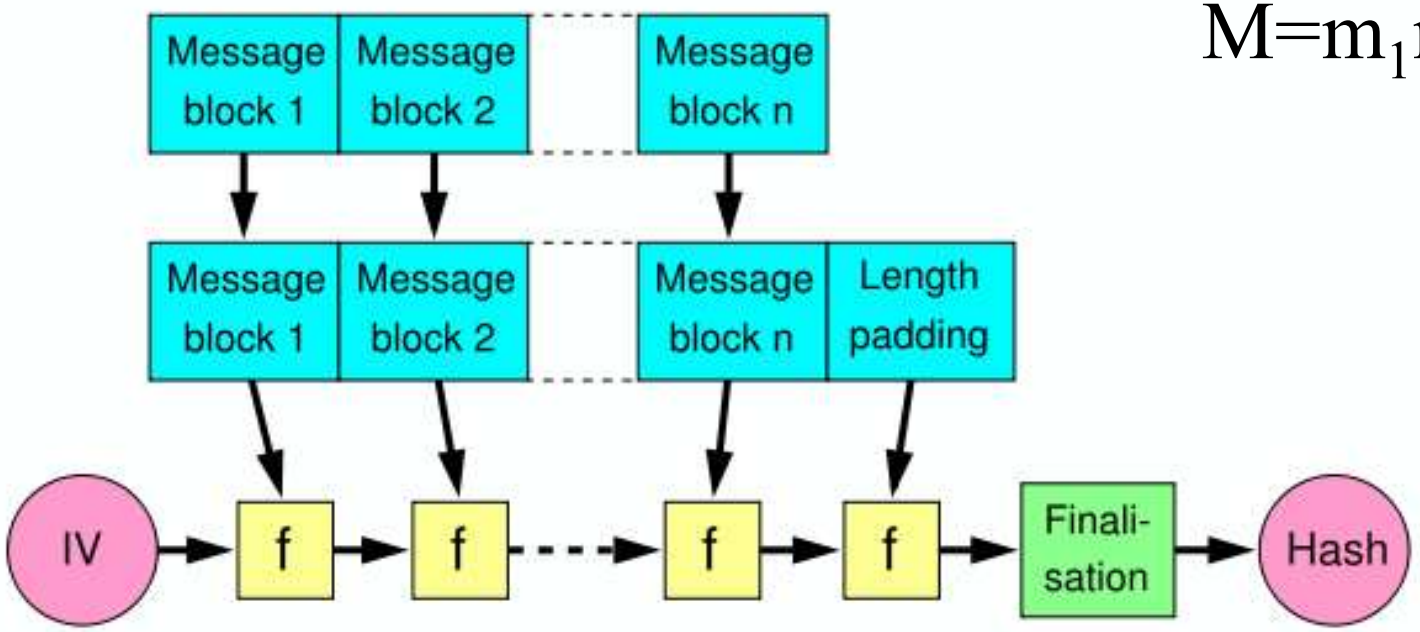
1. The source, A, is prepared to "sign" a message by appending the appropriate $m$-bit hash code and encrypting that hash code with A's private key
2. The opponent generates $2^{m/2}$ variations on the message, all of which convey essentially the same meaning. The opponent prepares an equal number of messages, all of which are variations on the fraudulent message to be substituted for the real one.
3. 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.
4. 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.

# Probability of Hash Collisions

- Arbitrary length message → Fixed length hash
- Many messages may map to the same hash
- Given message size=1000 bit → $2^{1000}$ messages
- 128 bit hash → $2^{128}$ possible hashes
- $2^{1000}/2^{128} = 2^{872}$ messages/hash value
- n-bit hash → Need avg $2^{n/2}$ tries to find two messages with same hash
- 64 bit hash → $2^{32}$ tries (feasible)
- 128 bit hash → $2^{64}$ tries (not feasible)

# Merkle-Damgard Construction for Hash Functions

- Message is divided into fixed-size blocks and padded
- Uses a compression function f, which takes a chaining variable (of size of hash output) and a message block, and outputs the next chaining variable
- Final chaining variable is the hash value



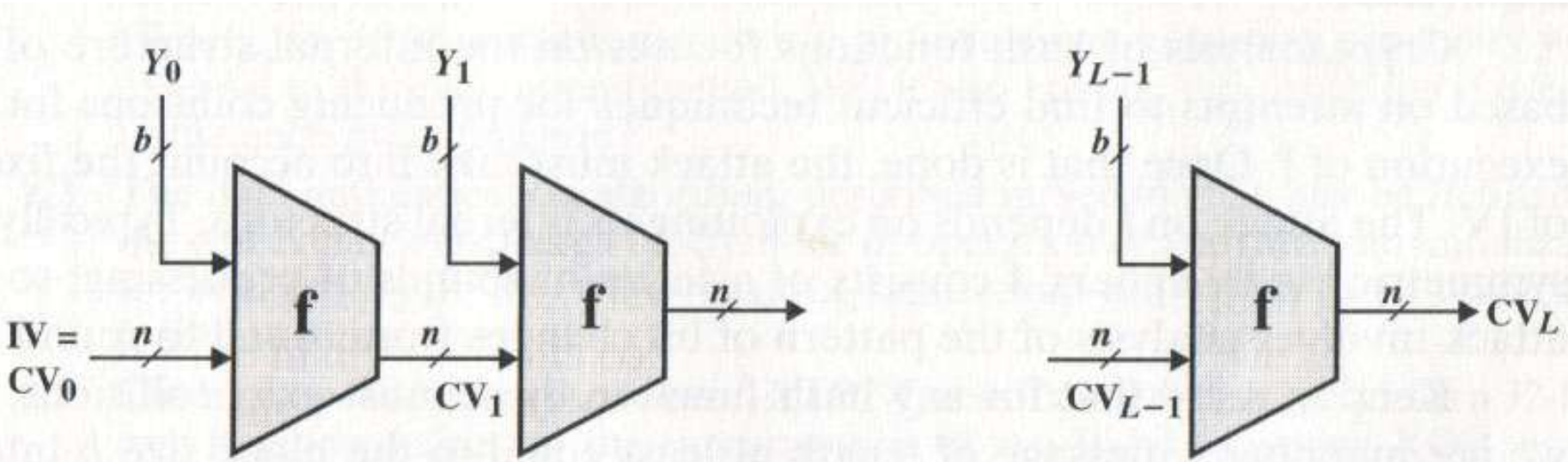

$$M=m_1 m_2 \ldots m_n;$$

$$C_0=IV,$$

$$C_{i+1}=f(C_i,m_i);$$

$$H(M)=C_n$$

# General Structure of Hash Function




IV = initial value
CV = chaining variable
$Y_i$ = *i*th input block
f = compression algorithm
*L* = number of input blocks
*n* = length of hash code
*b* = length of input block

f: compression function taking two inputs and producing n-bit output

$CV_0$ = IV = initial n-bit value

$CV_i = f(CV_{i-1}, Y_{i-1}), 1 \le i \le L$

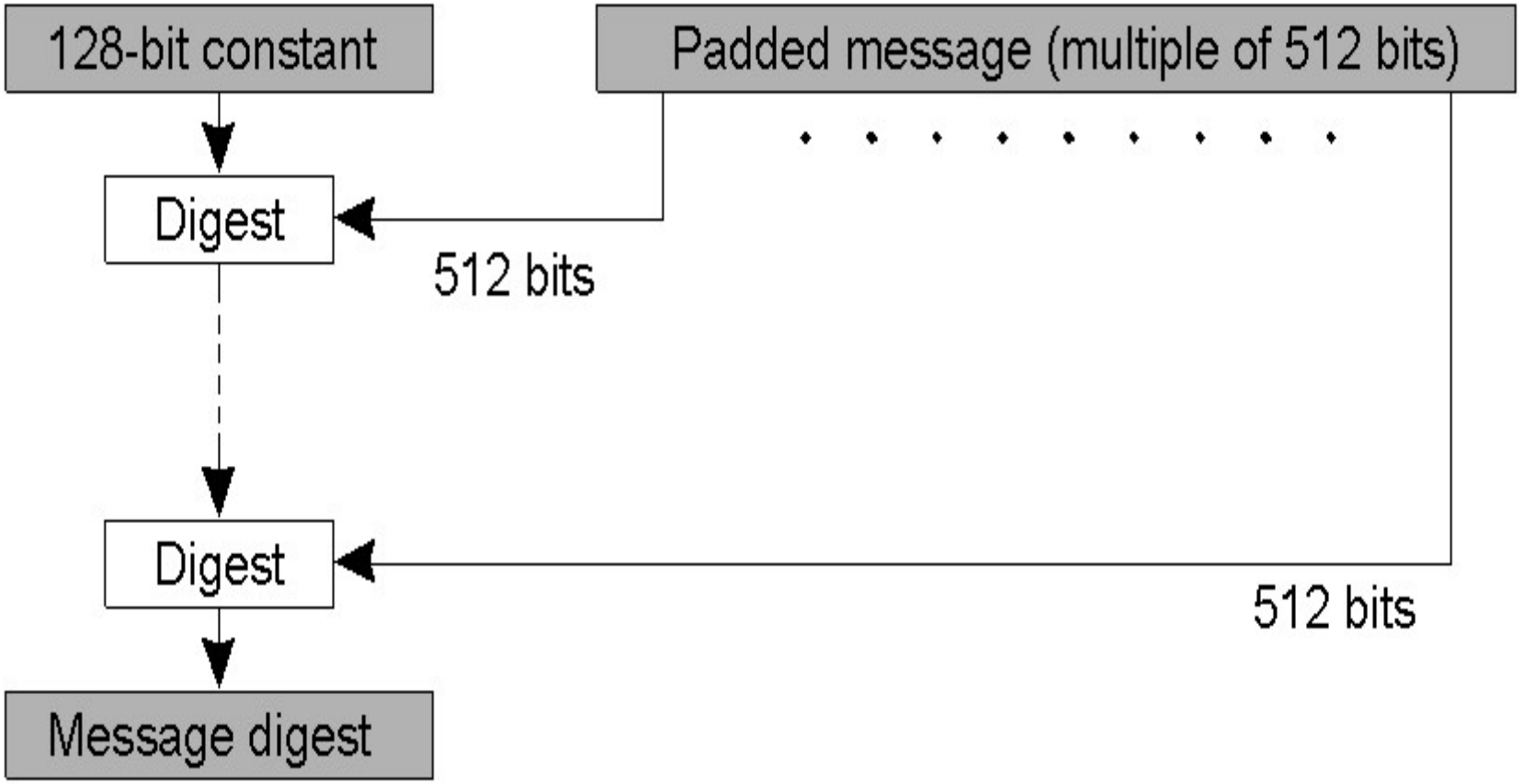
$H(M) = CV_L$

**Figure 8.10** General Structure of Secure Hash Code.

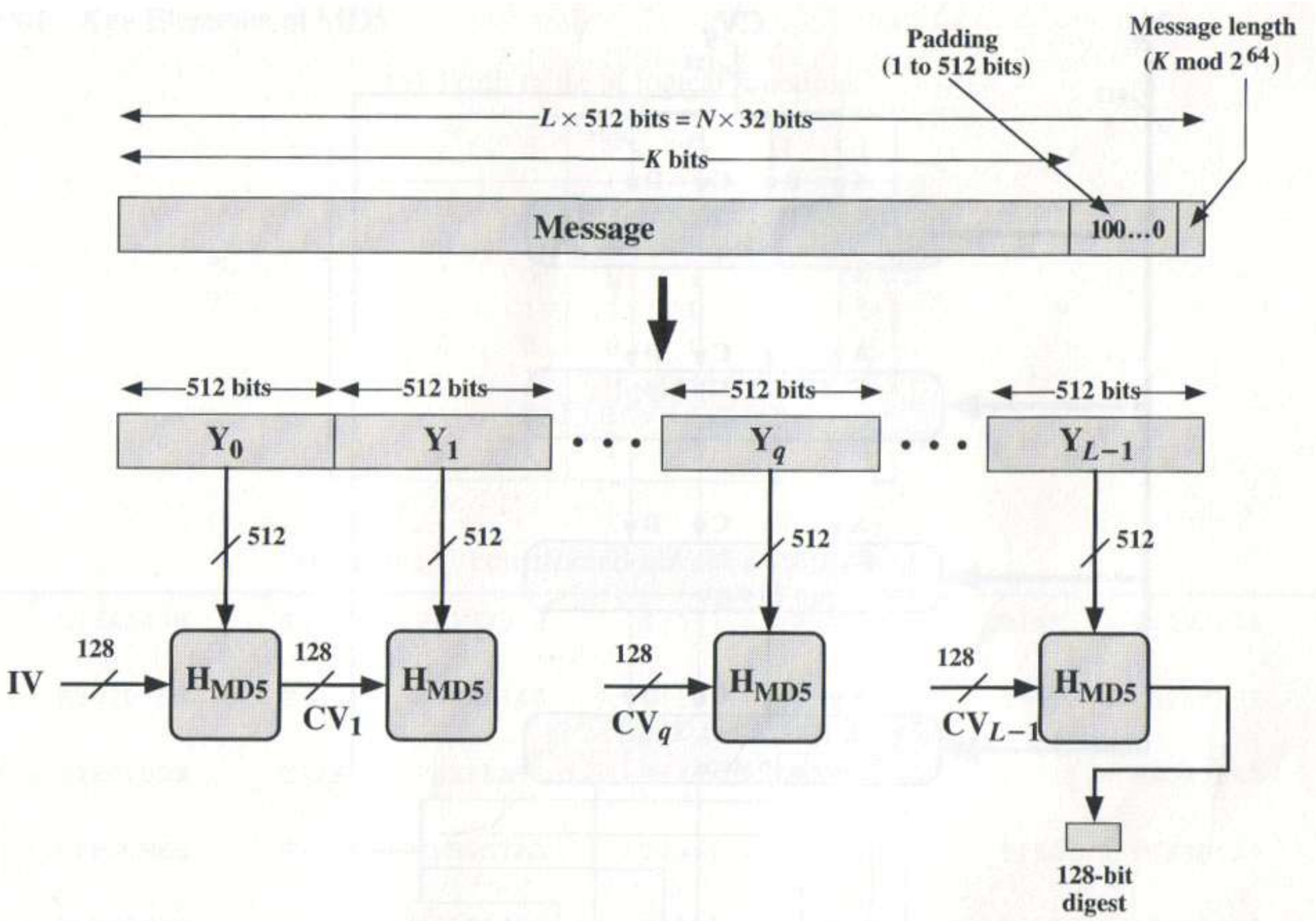
# MD5

- Designed by Ronald Rivest (the R in RSA)
- Latest in a series of MD2, MD4
- Produces a 128-bit hash value
- Until recently was the most widely used hash algorithm
  - in recent times have both brute-force and cryptanalytic concerns
- Specified as Internet standard RFC1321

# MD5 Algorithm Structure



# MD5 Message Digest Algorithm



# MD5 Steps

1. **Padding**: The input message K is padded (1 to 512 bits) so its length becomes a multiple of 512 bits. Padding includes a 1 bit followed by 0s
2. Append length: the original message length (in bits) appended as a 64-bit value. 64-bit representation of the length in bits. If the message is longer than $2^{64}$ bits, only low-order 64 bits of the length are used.
   - Message length = K mod $2^{64}$. The message is represented as a sequence of 512-bit blocks $Y_0, Y_1, \ldots, Y_{L-1}$
   - So, we have L blocks of 512 bits
   - Each block is divided into 16 32-bit words.
   - Total number of words in the message is N represented by M[0,…, N-1]

$$N = L \times 16$$

# MD5 Steps (cont.)

3. Initialize MD buffer

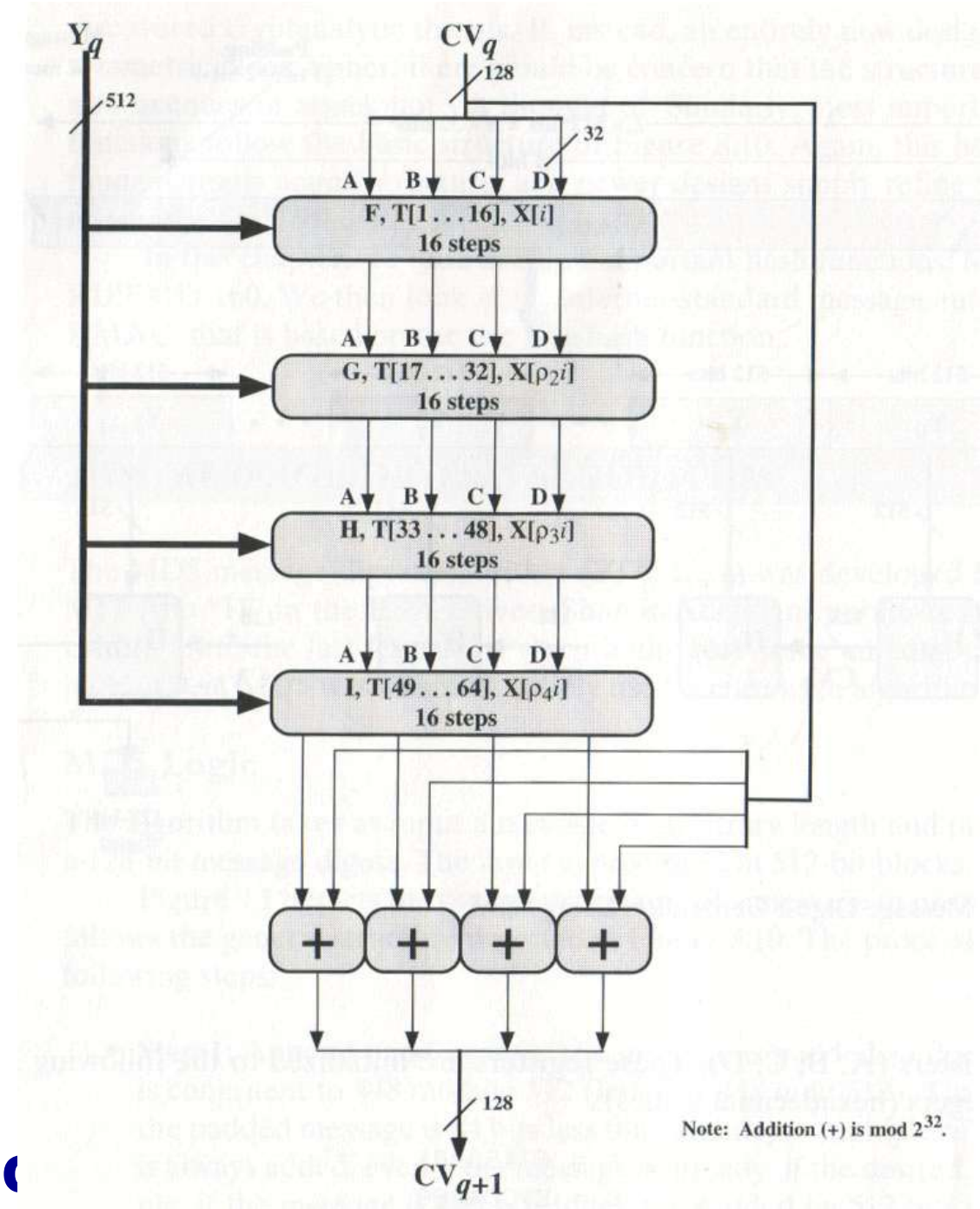
- The buffer is represented as 4 32-bit registers (A, B, C, D)
- Initialization value (in HEX)

  A: 01 23 45 67 (32 bits)

  B: 89 AB CD EF

  C: FE DC BA 98

  D: 76 54 32 10



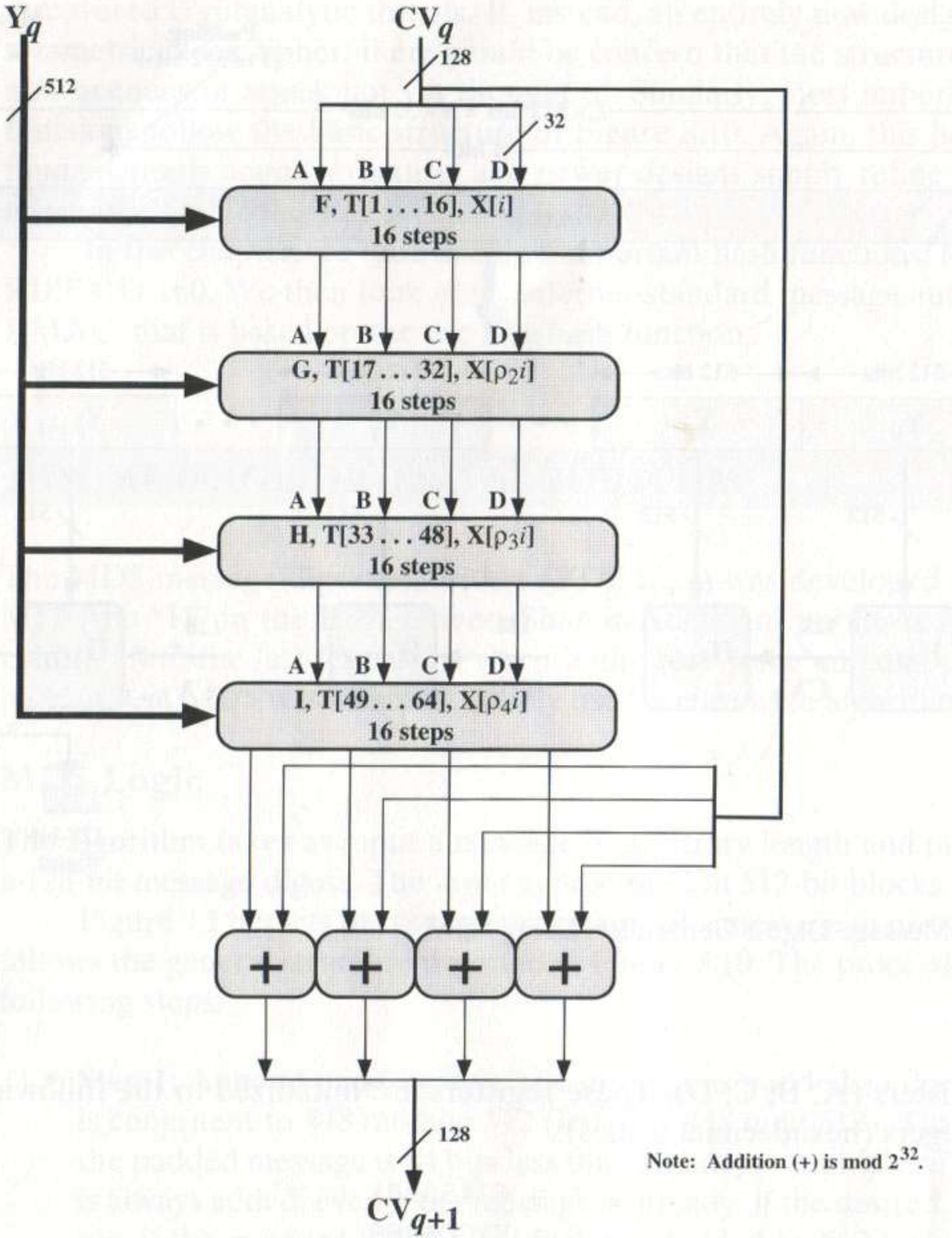
# MD5 Steps (cont.)

4. Process message in 512-bit (16-word) blocks

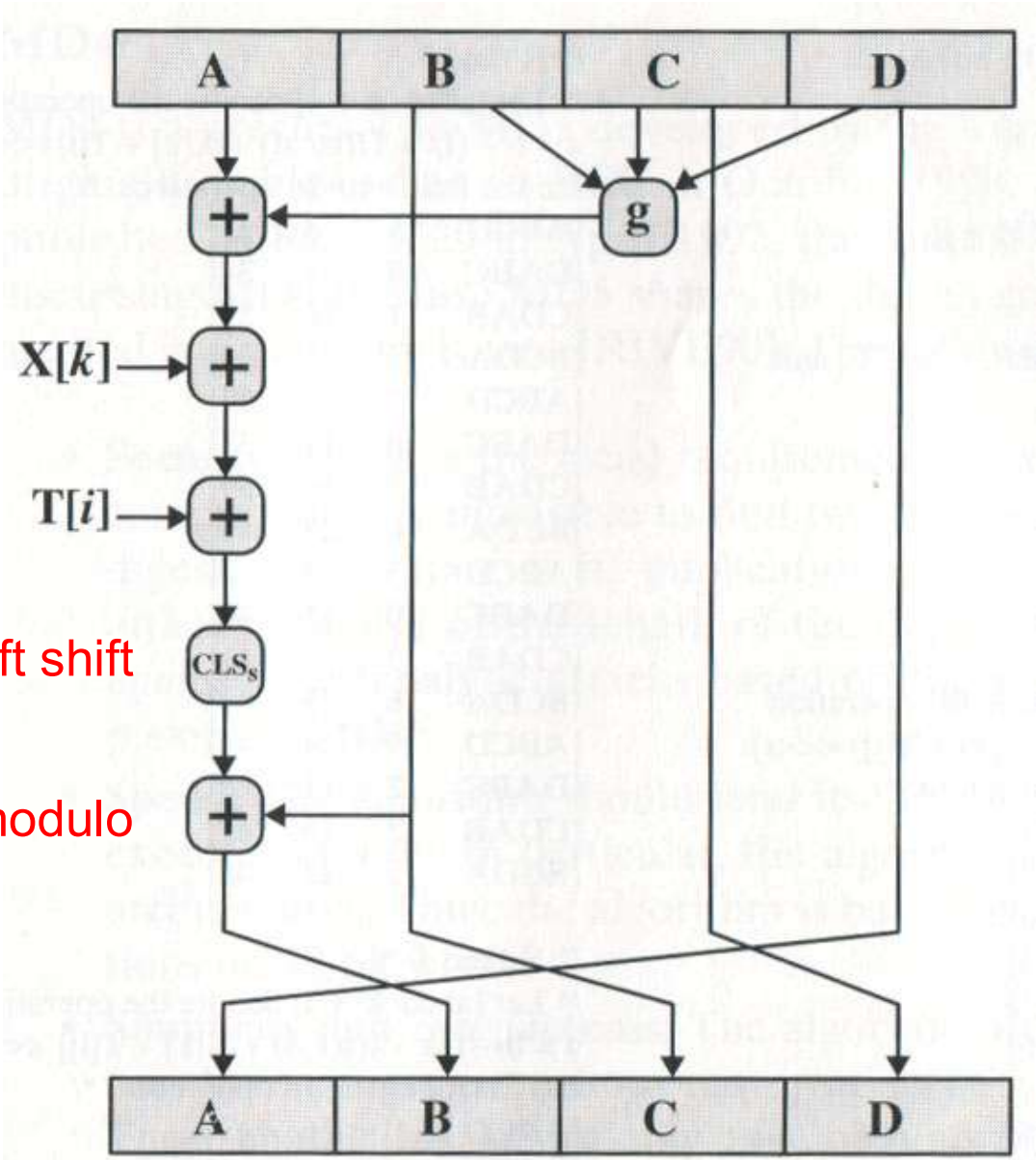
- F(b, c, d) = (b∧c)∨(~b ∧ d)
- G(b, c, d) = (b ∧ d) ∨(b ∧~d)
- H(b, c, d) = b⊕c⊕ d
- I(b, c, d) = c⊕(b ∧ ~d)

(a) Truth table of logical functions

| b | c | d | F | G | H | I |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |



# MD5 Compression Function




S-bit circular left shift

Addition modulo

**Figure 9.3** Elementary MD5 Operation (single step).

# MD5 Compression Function (cont.)

- Each step is in the form:

$b = b + ((a + g(b,c,d)) + X[k] + T[i] <<< s)$

a,b,c,d = four words of the buffer

g = one of the primitive functions F,G,H,I

<<<s = s-bit circular left shift

X[k] = M[q x 16 + k] = the $k^{th}$ 32-bit word in the $q^{th}$ 512-bit-block of the message

T[i] = the $i^{th}$ 32-bit word in matrix T

+ = addition modulo $2^{32}$

(b) Table T, constructed from the sine function

| | | | |
|---|---|---|---|
| T[1] = D76AA478 | T[17] = F61E2562 | T[33] = FFFA3942 | T[49] = F4292244 |
| T[2] = E8C7B756 | T[18] = C040B340 | T[34] = 8771F681 | T[50] = 432AFF97 |
| T[3] = 242070DB | T[19] = 265E5A51 | T[35] = 699D6122 | T[51] = AB9423A7 |
| T[4] = C1BDCEEE | T[20] = E9B6C7AA | T[36] = FDE5380C | T[52] = FC93A039 |
| T[5] = F57COFAF | T[21] = D62F105D | T[37] = A4BEEA44 | T[53] = 655B59C3 |
| T[6] = 4787C62A | T[22] = 02441453 | T[38] = 4BDECFA9 | T[54] = 8F0CCC92 |
| T[7] = A8304613 | T[23] = D8A1E681 | T[39] = F6BB4B60 | T[55] = FFEFF47D |
| T[8] = FD469501 | T[24] = E7D3FBC8 | T[40] = BEBFBC70 | T[56] = 85845DD1 |
| T[9] = 698098D8 | T[25] = 21E1CDE6 | T[41] = 289B7EC6 | T[57] = 6FA87E4F |
| T[10] = 8B44F7AF | T[26] = C33707D6 | T[42] = EAA127FA | T[58] = FE2CE6E0 |
| T[11] = FFFF5BB1 | T[27] = F4D50D87 | T[43] = D4EF3085 | T[59] = A3014314 |
| T[12] = 895CD7BE | T[28] = 455A14ED | T[44] = 04881D05 | T[60] = 4E0811A1 |
| T[13] = 6B901122 | T[29] = A9E3E905 | T[45] = D9D4D039 | T[61] = F7537E82 |
| T[14] = FD987193 | T[30] = FCEFA3F8 | T[46] = E6DB99E5 | T[62] = BD3AF235 |
| T[15] = A679438E | T[31] = 676F02D9 | T[47] = 1FA27CF8 | T[63] = 2AD7D2BB |
| T[16] = 49B40821 | T[32] = 8D2A4C8A | T[48] = C4AC5665 | T[64] = EB86D391 |

# MD5 Steps (cont.)

5. Output

$CV_0 = IV$

$CV_{q+1} = SUM_{32}(CV_q, RF_I[Y_q, RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]])$

$MD = CV_L$

IV = initial value of ABCD buffer

$Y_q$ = the $q^{th}$ 512-bit block of the message

L = the number of blocks in the message

$CV_q$ = chaining variable processed with $q^{th}$ message block

$RF_x$ = round function using primitive function x

MD = final message digest value

$SUM_{32}$ = Addition modulo $2^{32}$ performed separately on each word of the pair of inputs

# Strength of MD5

- MD5 hash is dependent on all message bits
- Rivest claims security is good as can be
- Known attacks are:
  - Berson 92 attacked any 1 round using differential cryptanalysis (but can't extend)
  - Boer & Bosselaers 93 found a pseudo collision (again unable to extend)
  - Dobbertin 96 created collisions on MD compression function (but initial constants prevent exploit)
- Conclusion is that MD5 looks vulnerable

# Secure Hash Function (SHA)

| | SHA-1 | SHA-224 | SHA-256 | SHA-384 | SHA-512 |
|---|---|---|---|---|---|
| **Message Digest Size** | 160 | 224 | 256 | 384 | 512 |
| **Message Size** | $< 2^{64}$ | $< 2^{64}$ | $< 2^{64}$ | $< 2^{128}$ | $< 2^{128}$ |
| **Block Size** | 512 | 512 | 512 | 1024 | 1024 |
| **Word Size** | 32 | 32 | 32 | 64 | 64 |
| **Number of Steps** | 80 | 64 | 64 | 80 | 80 |

*Note:* All sizes are measured in bits.

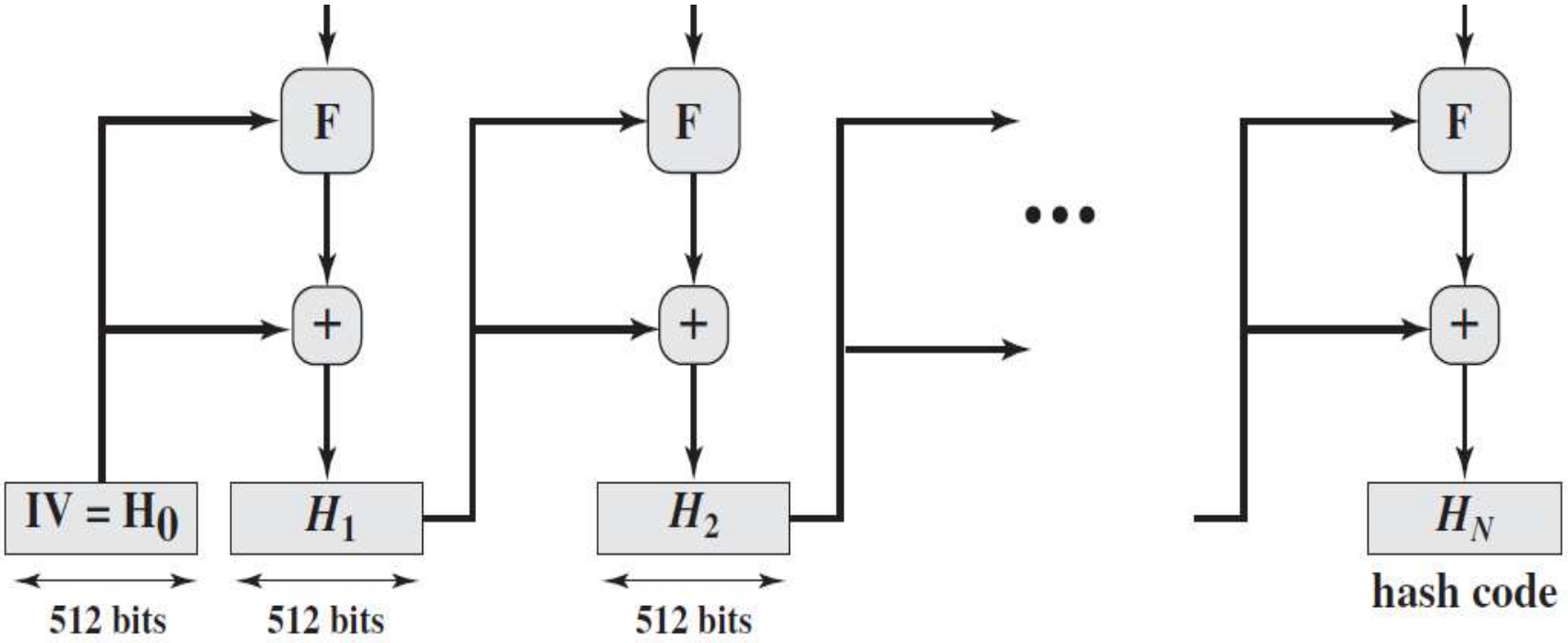
# Requirements for SHA-3

- Plug-compatible with SHA-2 in current apps
- Support digital signatures, hash-based MACs, PRFs, RNGs, KDFs, etc.
- Required security properties
  - Collision resistance of approximately n/2 bits,
  - Preimage resistance of approximately n bits,
  - Second-preimage resistance of approximately n-k bits for any message shorter than $2^k$ bits,
  - Resistance to length-extension attacks.

# SHA-1 Features

- Support hash value lengths: 224, 256, 384, 512 bits.
- Can process comparatively small blocks (512, 1024 bits) at a time instead of requiring that the entire message be buffered in memory before
- Security: strength of SHA-3 to resist any potentially successful attack on SHA-2 functions
- Cost: both time and memory efficient over a range of hardware platforms.
- Algorithm and implementation characteristics:
  - Flexibility (e.g., tunable parameters for security/performance tradeoffs, opportunity for parallelization, and so on)
  - Simplicity (which makes it easier to analyze the security properties of the algorithm)

# The Secure Hash Function (SHA)



# SHA Algorithm

## Step 1: Append padding bits

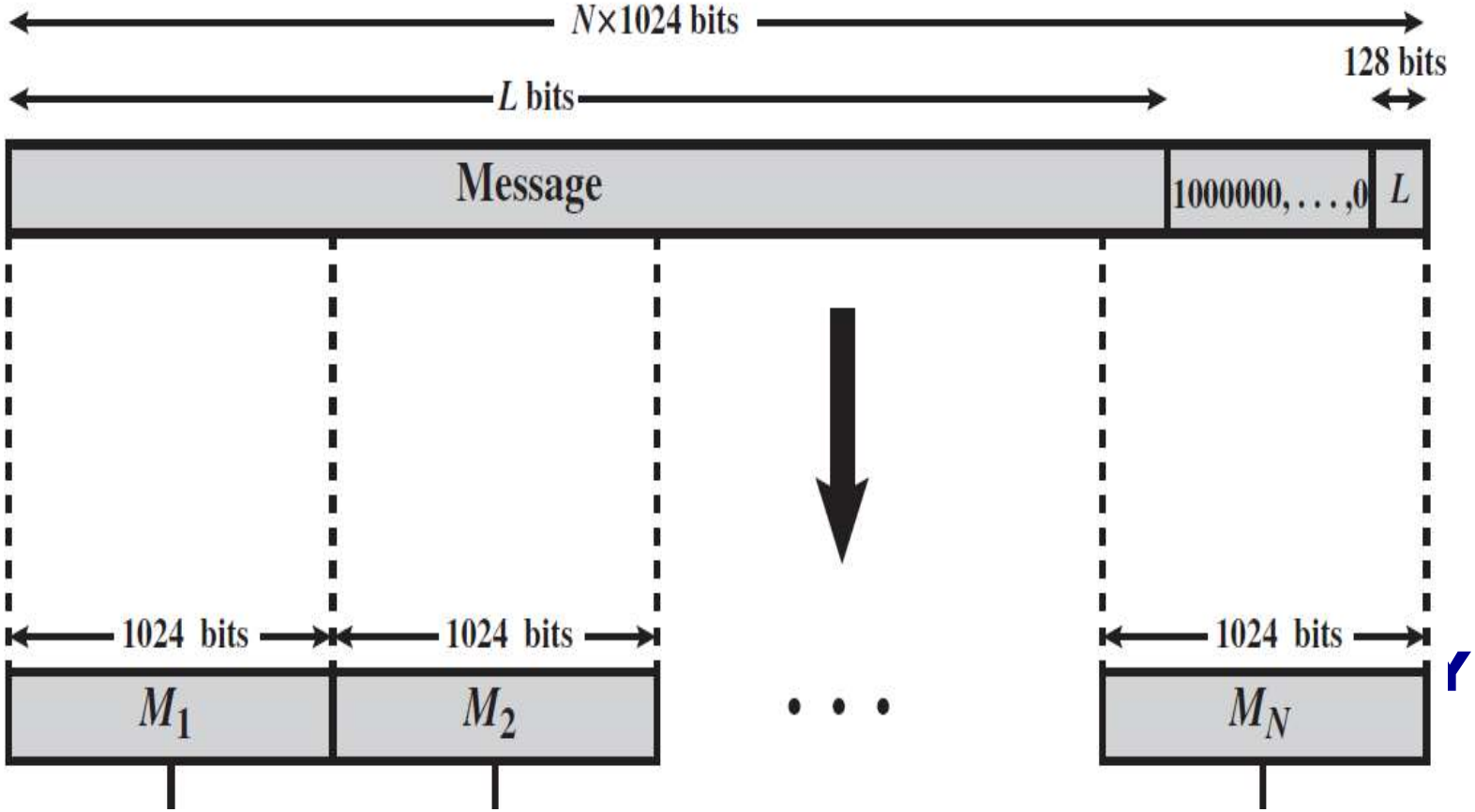
- Padding is added, even if the message is already of the desired length. No. of Padding bits = [1 to 1024]
- Padding consists of a single 1 bit followed by the necessary number of 0 bits.

## Step 2: Append length

- A block of 128 bits is appended to the message.
- This block is treated as an unsigned 128-bit integer and contains the length of the original message (before the padding).

# SHA Algorithm

- Outcome of first two steps yields a message an integer multiple of 1024 bits in length.
- Total length of the expanded message is N × 1024 bits as the expanded message is a sequence of 1024-bit blocks $M_1$, $M_2$, . . ., $M_N$.



# SHA Algorithm

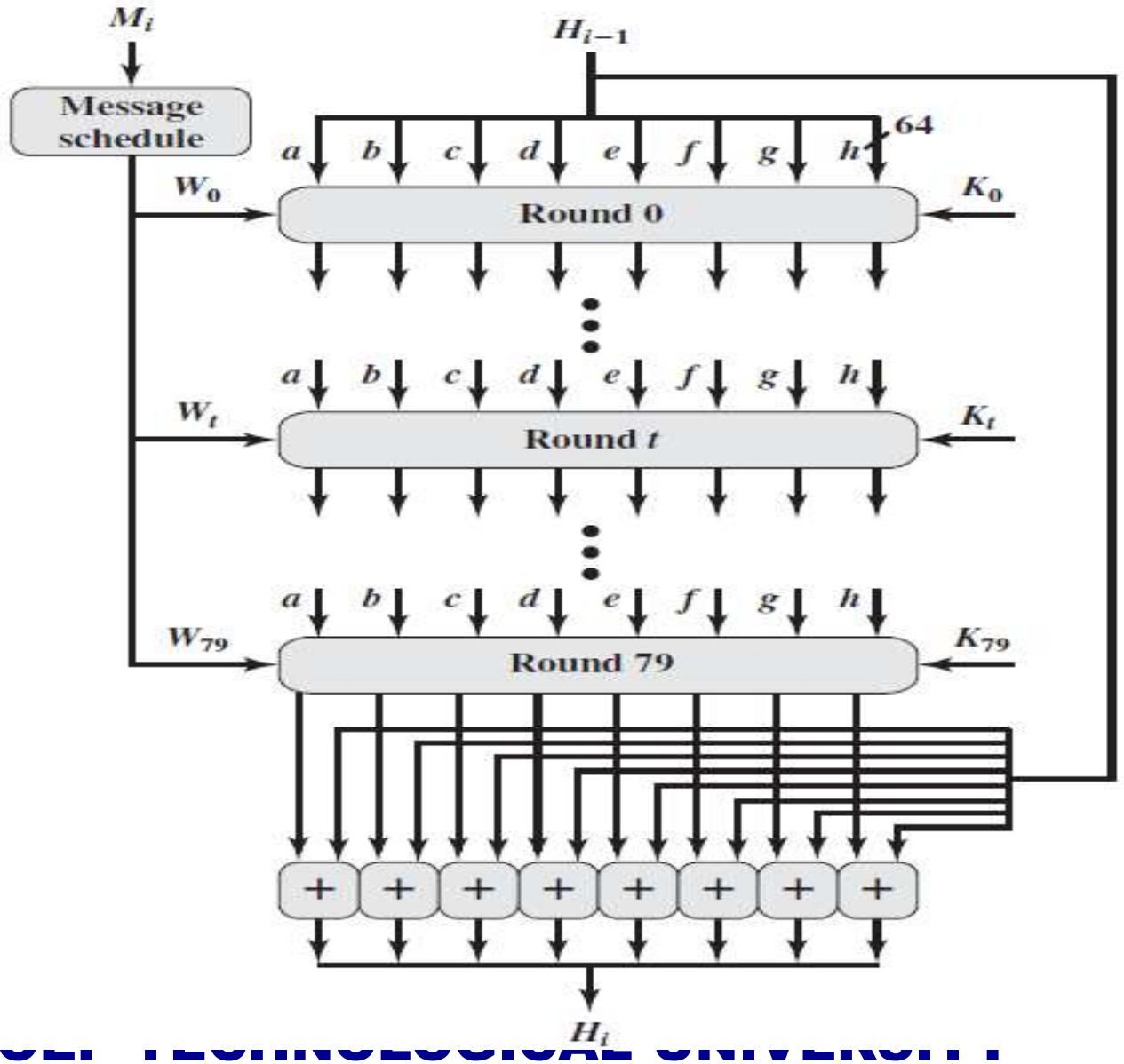
## Step 3: Initialize hash buffer

- 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*).
- Initialize these registers by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

$$a = \texttt{6A09E667F3BCC908} \qquad e = \texttt{510E527FADE682D1}$$
$$b = \texttt{BB67AE8584CAA73B} \qquad f = \texttt{9B05688C2B3E6C1F}$$
$$c = \texttt{3C6EF372FE94F82B} \qquad g = \texttt{1F83D9ABFB41BD6B}$$
$$d = \texttt{A54FF53A5F1D36F1} \qquad h = \texttt{5BE0CD19137E2179}$$

# The Secure Hash Function (SHA)

## Processing o



# SHA Algorithm

Step 4: Process message in 1024-bit (128-word) blocks

- The module labeled F consists of 80 rounds.
- Each round takes as input the 512-bit buffer value *a,b,c,d,e,f,g,h* and updates the contents of the buffer.
- At input to the first round, the buffer has the value of the intermediate hash value, $H_{i-1}$.
- Each round **t** makes use of a 64-bit value $W_t$ derived from the current 1024-bit block being processed ($M_i$).

# SHA Algorithm

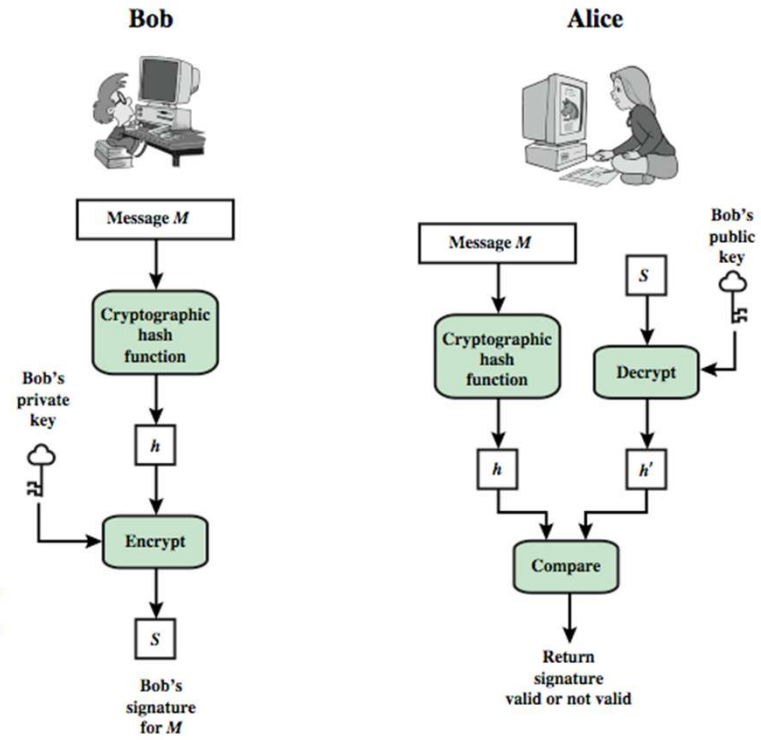
- Each round also makes use of an additive constant $K_t$, where t = 0 … … 79.
- The constants eliminate any regularities in the input data.
- The output of the 80th round is added to the input to the first round ($H_{i-1}$) to produce $H_i$ .

## Step 5 Output:

- After all N 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.
- In 2012, NIST formally published SHA-3

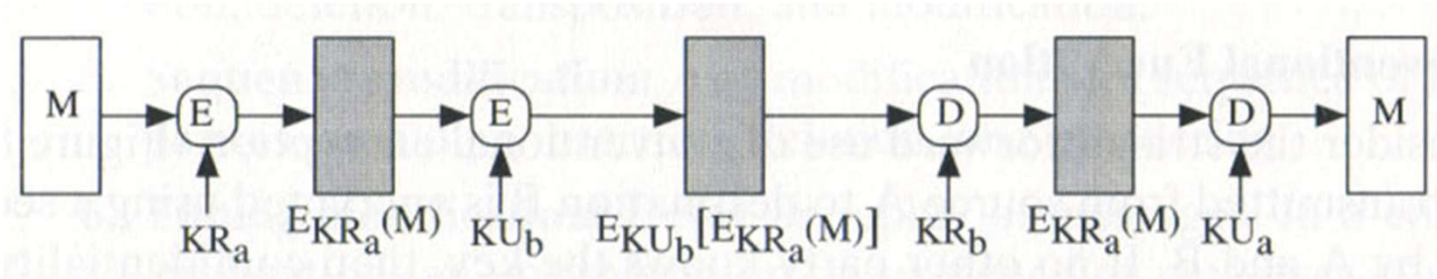
# Digital Signatures

- Digital signatures provide the ability to:
  - Verify author, date and time of signature
  - Authenticate message contents
  - Be verified by third parties to resolve disputes (non-repudiation)
- Hence include authentication function with additional capabilities



# Direct Digital Signatures

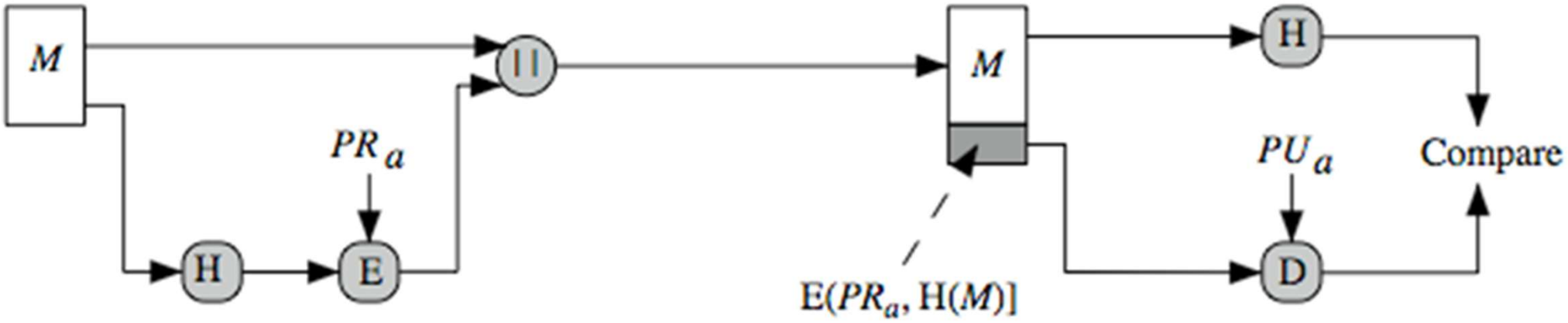
- Involve only sender and receiver
- Assumed receiver has sender's public-key
- Digital signature made by sender signing entire message or hash with private-key
- Can encrypt using receiver's public-key
- Important that sign first then encrypt message and signature
- Security depends on sender's private-key



# Digital Signature Properties

- Must depend on the message signed
- Must use information unique to sender
  - to prevent both forgery and denial
- Must be relatively easy to produce
- Must be relatively easy to recognize and verify
- Be computationally infeasible to forge
  - with new message for existing digital signature
  - with fraudulent digital signature for the given message
- Be practical save digital signature in storage

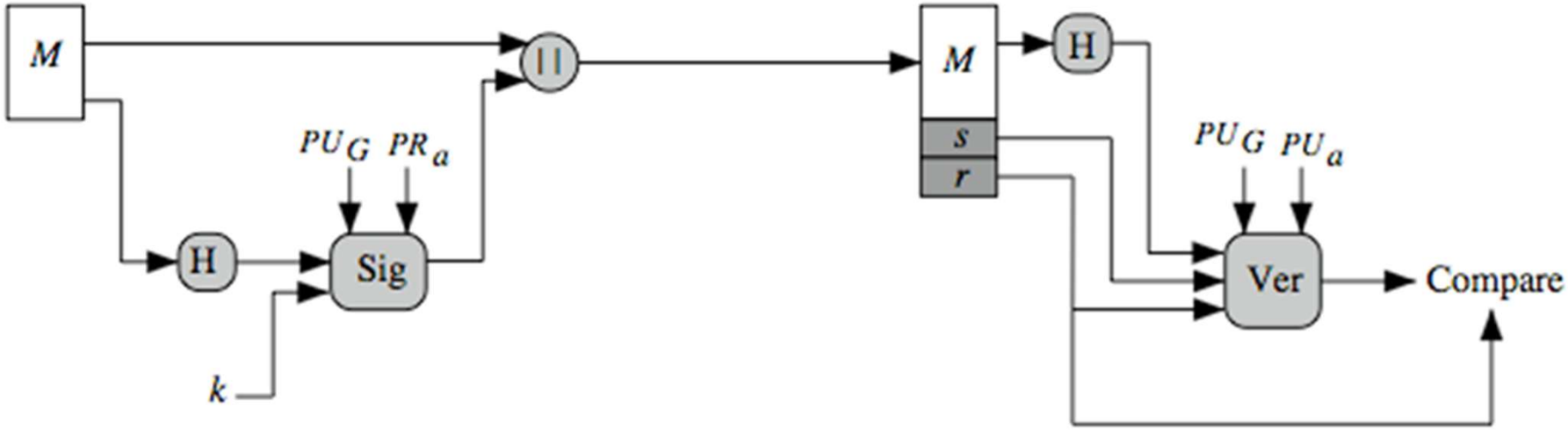
# RSA Signatures



# Digital Signature Standard (DSS)

- US Govt approved signature scheme FIPS 186
- uses the SHA hash algorithm
- designed by NIST and NSA in early 90's
- DSS is the standard, DSA is the algorithm
- A variant on ElGamal and Schnorr schemes
- Creates a 320 bit signature, but with 512-1024 bit security
- Security depends on difficulty of computing discrete logarithms

# DSS Signatures




- Hash code from M is provided as input to a signature function along with a random number *k* generated for this particular signature
- A global public key, $PU_G$ is a parameter for a group communication
- Result is a signature consisting of two components, labeled s and r

# Digital Signature Algorithm (DSA)

- DSA is the algorithm for Digital Signature generation and verification
- Smaller and faster than RSA
- Security depends on difficulty of computing discrete logarithms
- Variant of ElGamal and Schnorr schemes

# Global parameters and Key Generation

**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 of between 512 and 1024 bits
in increments of 64 bits

q prime divisor of $(p - 1)$, where $2^{159} < q < 2^{160}$;
i.e., bit length of 160 bits

g $= h^{(p-1)/q} \bmod p$,
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**

$y = g^x \bmod p$

# Signature Generation

**User's Per-Message Secret Number**

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

**Signing**

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

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

Signature = $(r, s)$

# Signature Verification

**Verifying**

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

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

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

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

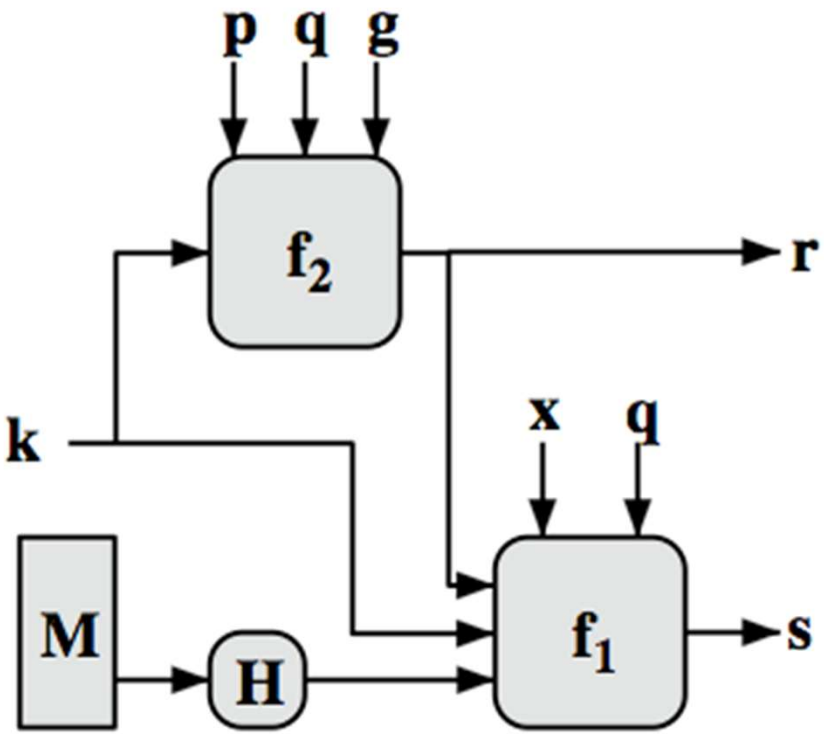
TEST: $v = r'$

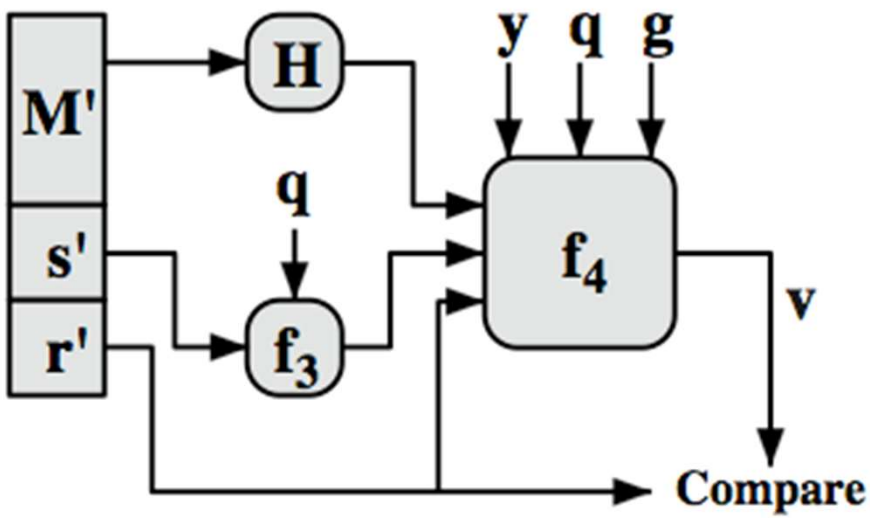
$M$ = message to be signed

$H(M)$ = hash of M using SHA-1

$M', r', s'$ = received versions of $M, r, s$

# DSS Overview




$s = f_1(H(M), k, x, r, q) = (k^{-1} (H(M) + xr)) \bmod q$

$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$

**(a) Signing**

$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M'), w, r')$

$= ((g^{(H(M')w) \bmod q} y^{r'w \bmod q}) \bmod p) \bmod q$

**(b) Verifying**

# Public Key Infrastructure (PKI)

- Digital certificates alone are not enough to establish security
- Need control over certificate issuance and management
- Certification authorities issue certificates
- Who verifies the identity of certification authorities?
- Naming of entities
- Certification Practice Statement (Statement by a CA of the policies and procedures it uses to issue certificates)
- Certificate Revocation List
- The metafunctions of certificate issuance form the Public Key Infrastructure

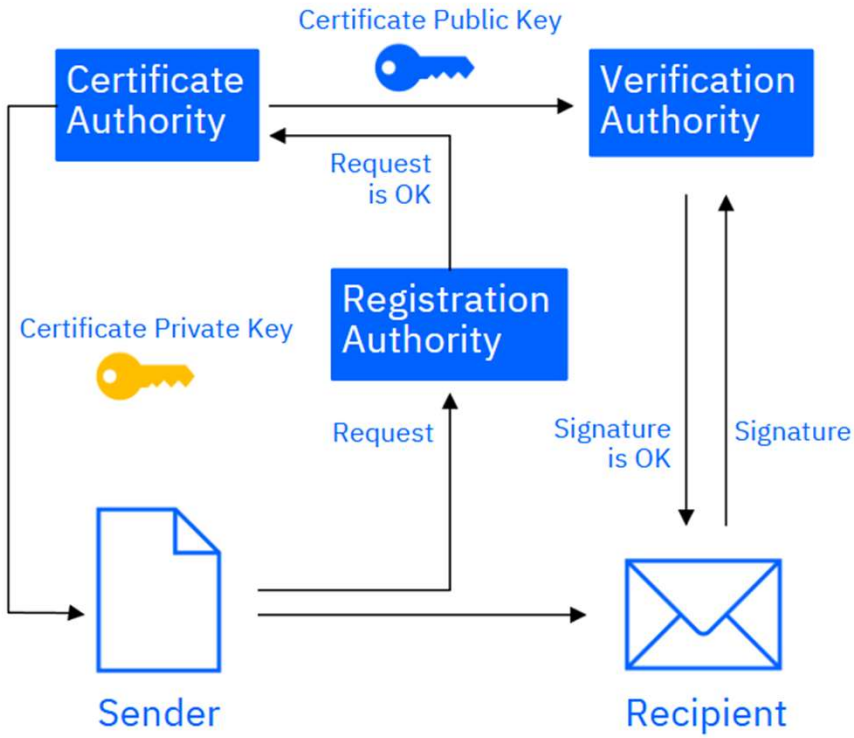
# PKI

- Some Trusted Agency is required which certifies the association of an individual with the key pair.

  *Certifying Authority (CA)*

- This association is done by issuing a certificate to the user by the CA

  *Public key certificate (PKC)*

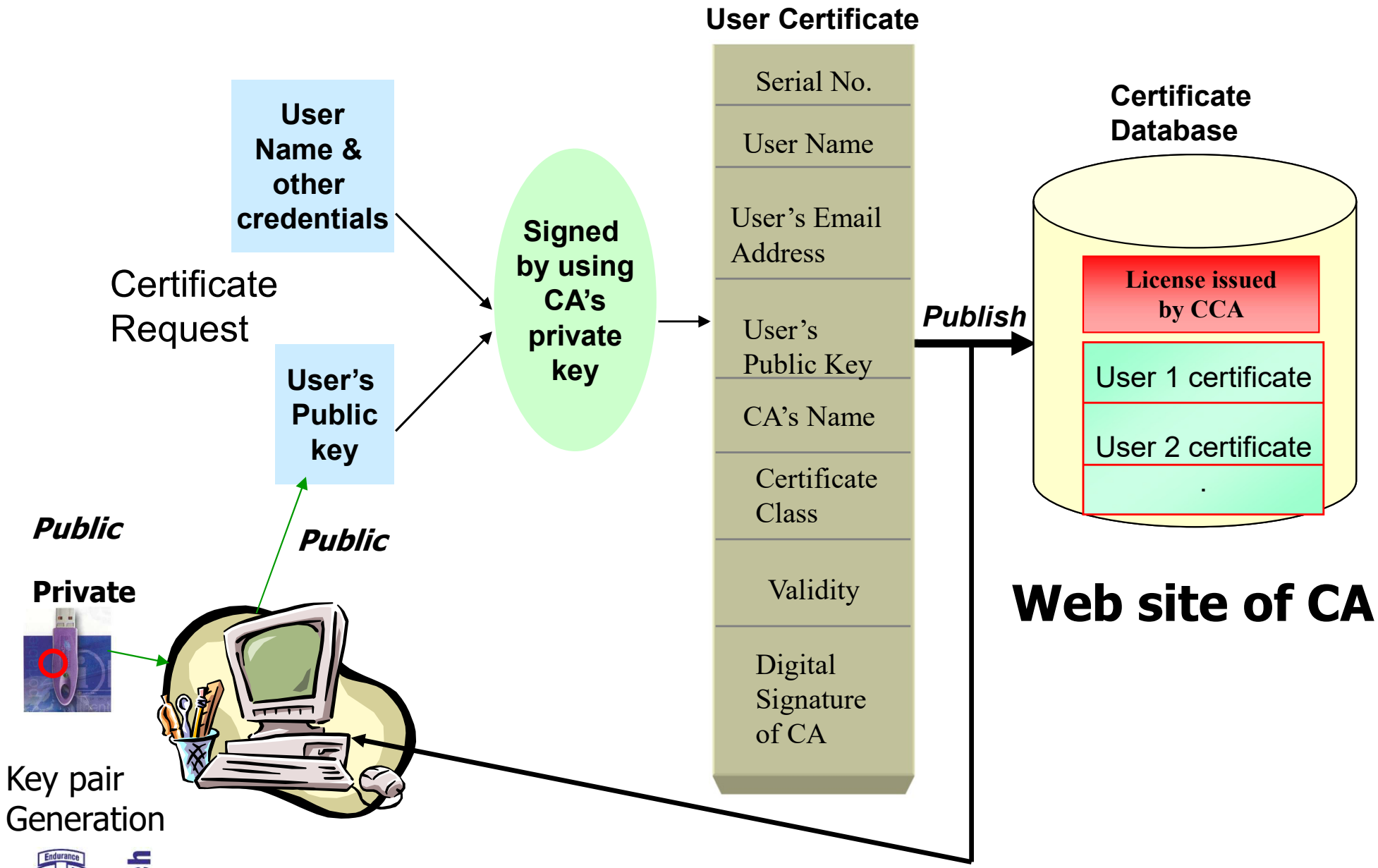



- All public key certificates are digitally signed by the CA
- View Verisign Certification Practice Statement
- View Verisign CRL
- Verisign CRL usage agreement

# Functions of a CA

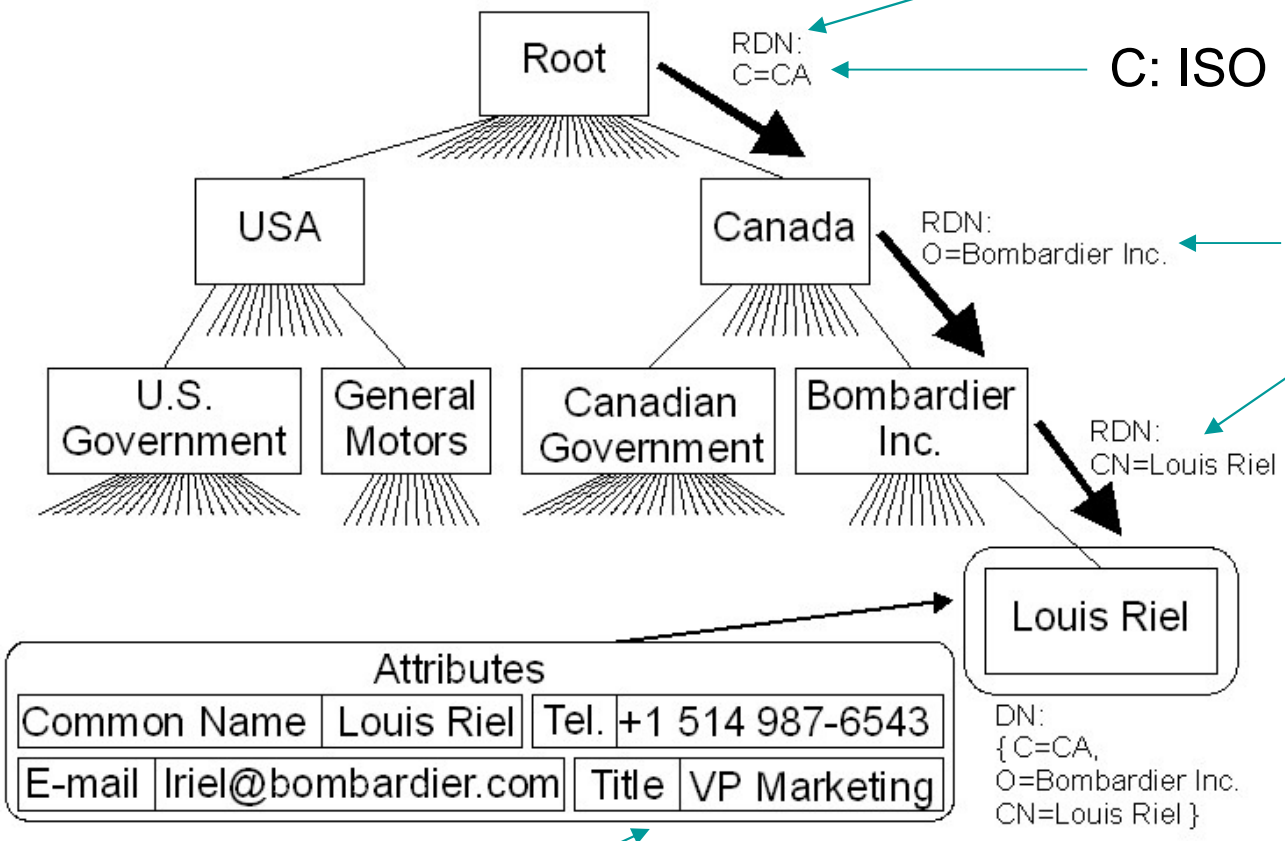
- Must be widely known and trusted
- Must have well defined Identification process before issuing the certificate
- Provides online access to all the certificates issued
- Provides online access to the list of certificates revoked
- Displays online the license issued by the Controller
- Displays online approved Certification Practice Statement (CPS)
- Must adhere to IT Act/Rules/Regulations and Guidelines

# Public-Key Certification



# ISO X.500 Directory Standard

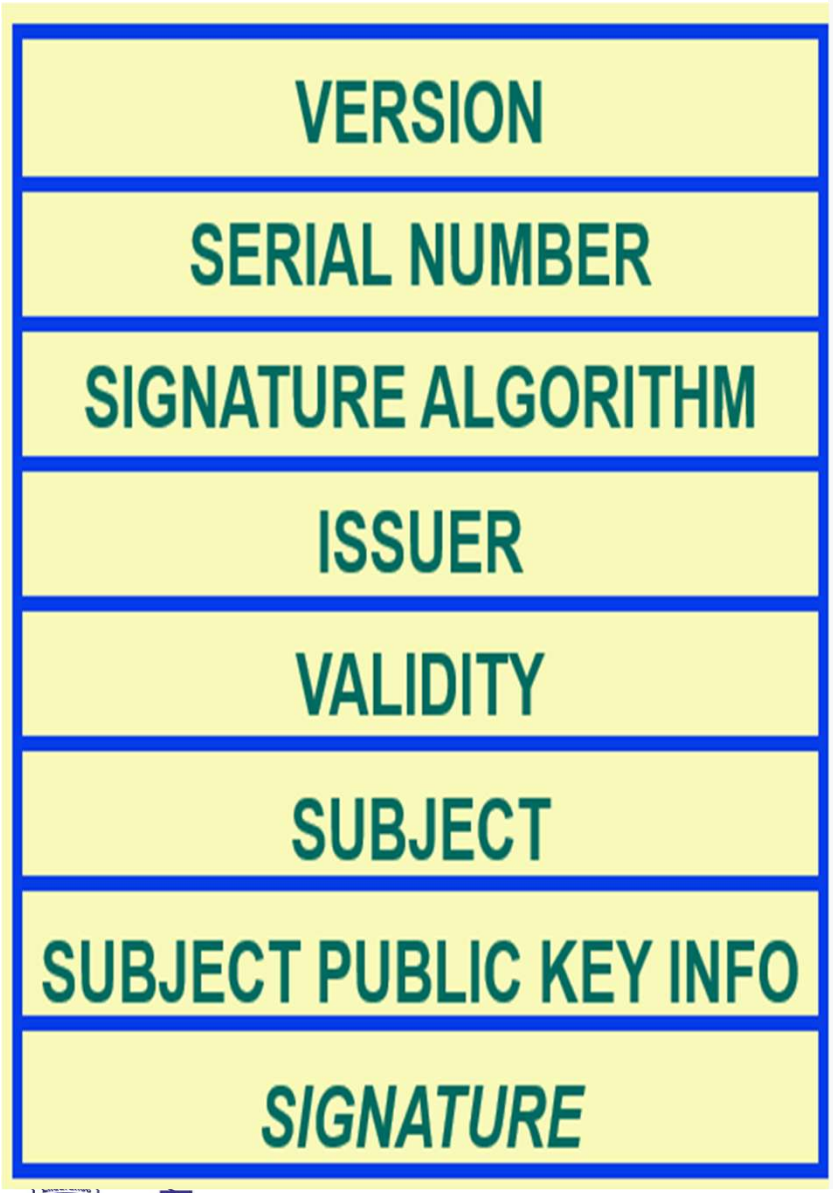
STANDARD FOR HIERARCHICAL DIRECTORIES

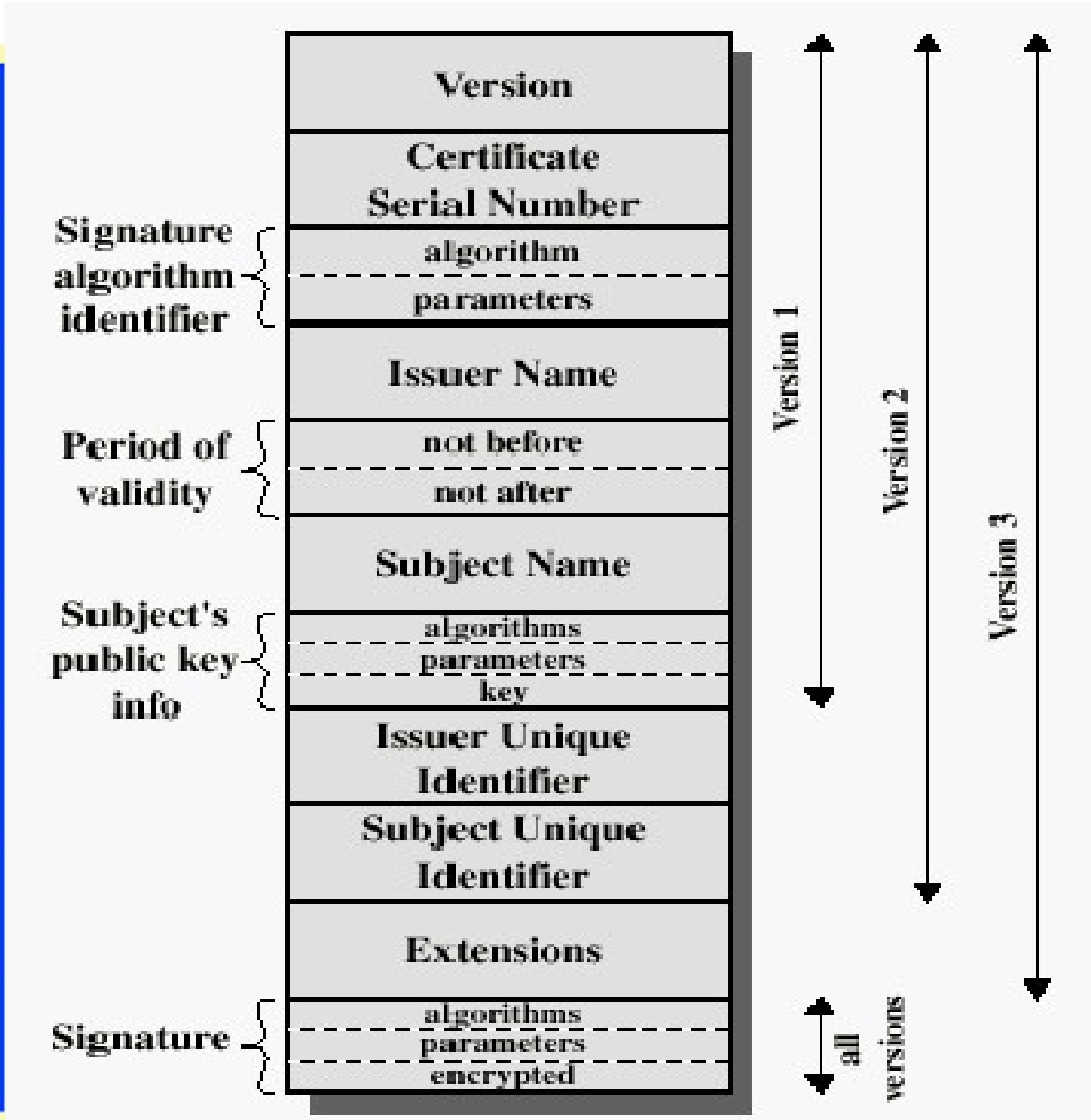


# X.509 Certificates

- Each certificate contains the public key of a user and is signed with the private key of a CA
- Is used in S/MIME, IP Security, SSL/TLS and SET
- CA<<A>> denotes certificate for A signed by CA
- Version 1, 2, 3
- Protocols Supporting X.509 Certificates
  - Transport Layer Security (SSL/TLS)
  - IPSec
  - Secure Multipurpose Internet Mail Extensions (S/MIME)
  - Smartcard
  - SSH
  - HTTPS
  - LDAPv3

# X.509 Certificates



# Obtaining certificates

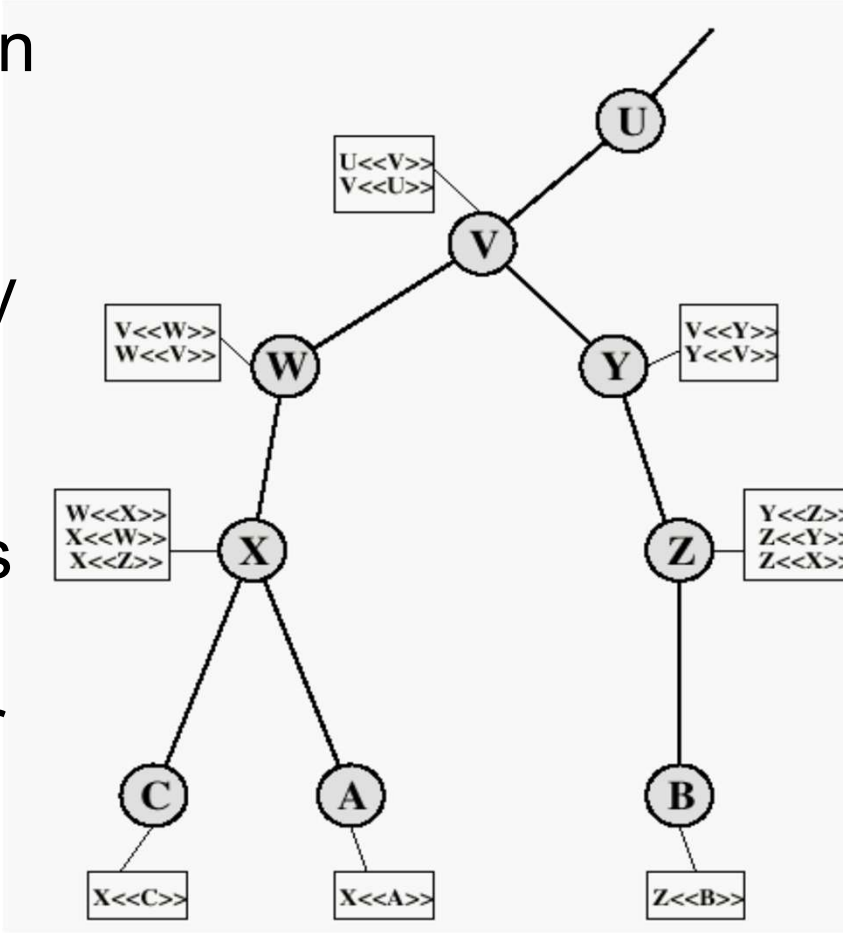
- Any user with access to CA public key can get any certificate from it
- CA can delegate some of the tasks to Registration Authority
  - Verifying registration information of new user, generating keys on behalf of end users, accepting and authorizing requests for certificate revocation
- Because cannot be forged, certificates can be placed in a public directory like X.500

# Digital Certificate Verification

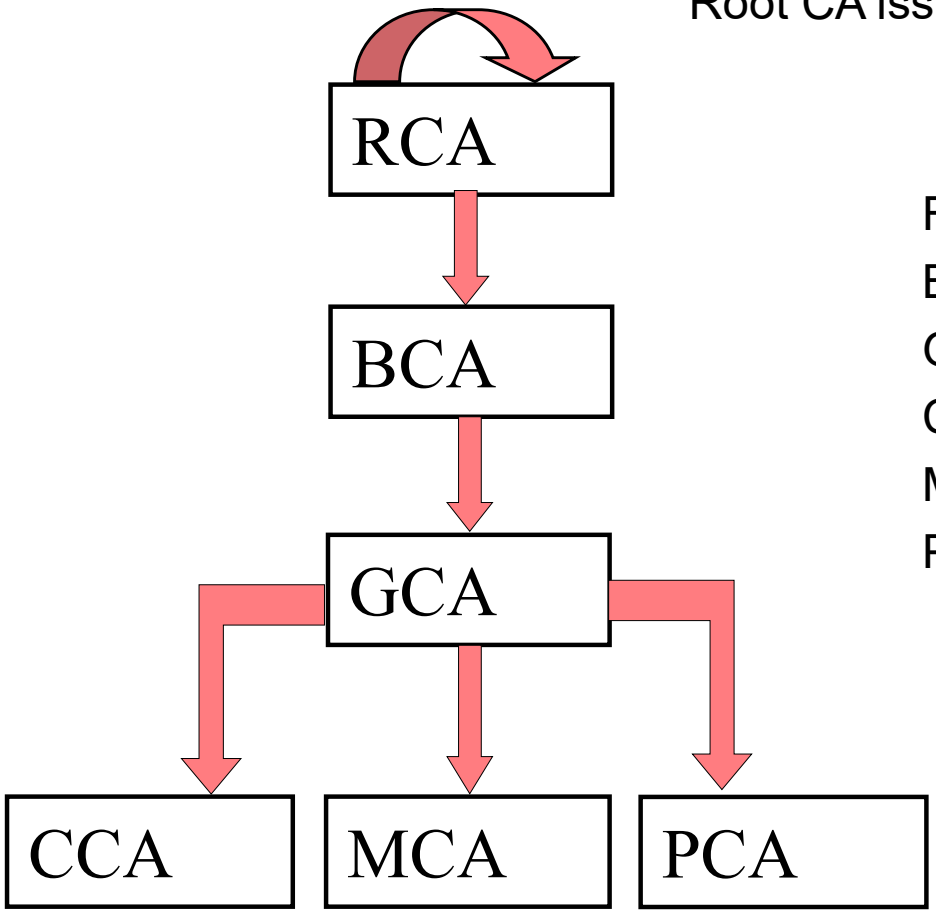
- CA's digital signature and CA's public key are input to Signature Verification algorithm to produce message digest1
- All fields of the certificate except CA's digital signature is input to hash algorithm to produce message digest2
- Compare the message digests

# CA Hierarchy

- If both users share a common CA then they are assumed to know its public key
- Otherwise CA's must form a hierarchy
- Each client trusts parents' certificates
- Enable verification of any certificate from one CA by users of all other CAs in the hierarchy
- Root CA is the top in the hierarchy for each country
- Certificate of root CA is self-signed and comes as part of web browser or web server
- Cross-certification allows CAs and end users from different PKIs to interact



# Certificate Authority Hierarchy



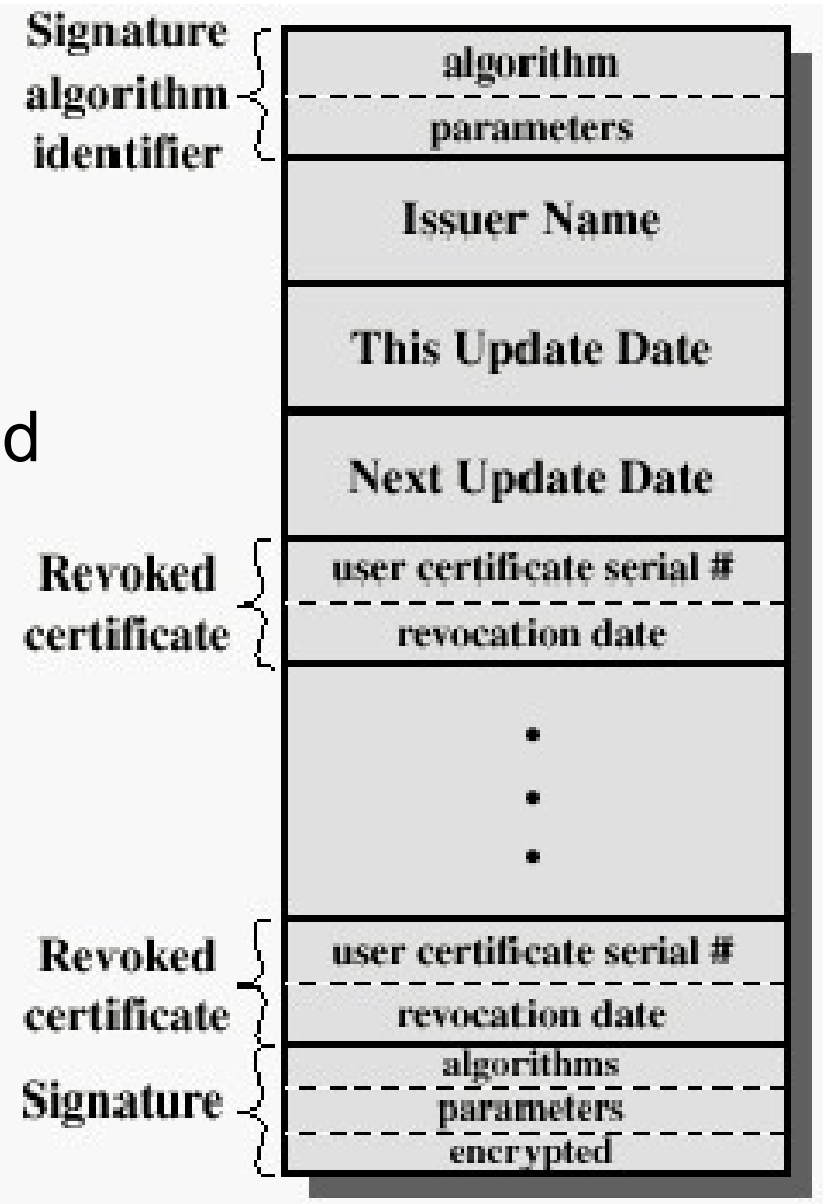

Root CA issues its own certificate!

RCA : Root Certificate Authority
BCA : Brand Certificate Authority
GCA : Geo-political Certificate Authority
CCA : Cardholder Certificate Authority
MCA : Merchant Certificate Authority
PCA : Payment Gateway
Certificate Authority

CERTIFICATE ISSUANCE

# Certificate Revocation

- Certificates have a period of validity
- May need to revoke before expiry, eg:
  1. User's private key is compromised
  2. User is no longer certified by this CA
  3. CA's certificate is compromised
- CA's maintain list of revoked certificates – CRL



# Validating certificate

- Check certificate is not expired
- Verify the chain of certificates
- Make sure that serial no of certificate does not appear in CRL

# Private Key Management

- How to protect private key?
- Mechanisms
  - Password protection
  - Personal Computer Memory Card International Association cards stores the key
  - Tokens stores the key and access through one-time password
  - Biometrics
  - Smart cards