Message Authentication and Digital Signatures

4.1 Message Authentication Approaches, Hash Function, Cryptographic Hash Function Requirements, Cryptographic Hash Function Security, Cryptographic Hash Function Structure, SHA, HMAC, MD5.

4.2 Using Symmetric Encryption for Message Authentication, Message Authentication Code (MAC), Digital Authentication Algorithm (DAA)

4.3 Using Public Key for Authentication, Digital Signatures, Properties of Digital Signatures beyond Message Authentication, DSS, Authentication Applications: Kerberos, X.509 Authentication Service

Message Authentication

- message authentication is concerned with:
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- will consider the security requirements
- then three alternative functions used:
  - hash function (see Ch 11)
  - message encryption
  - message authentication code (MAC)

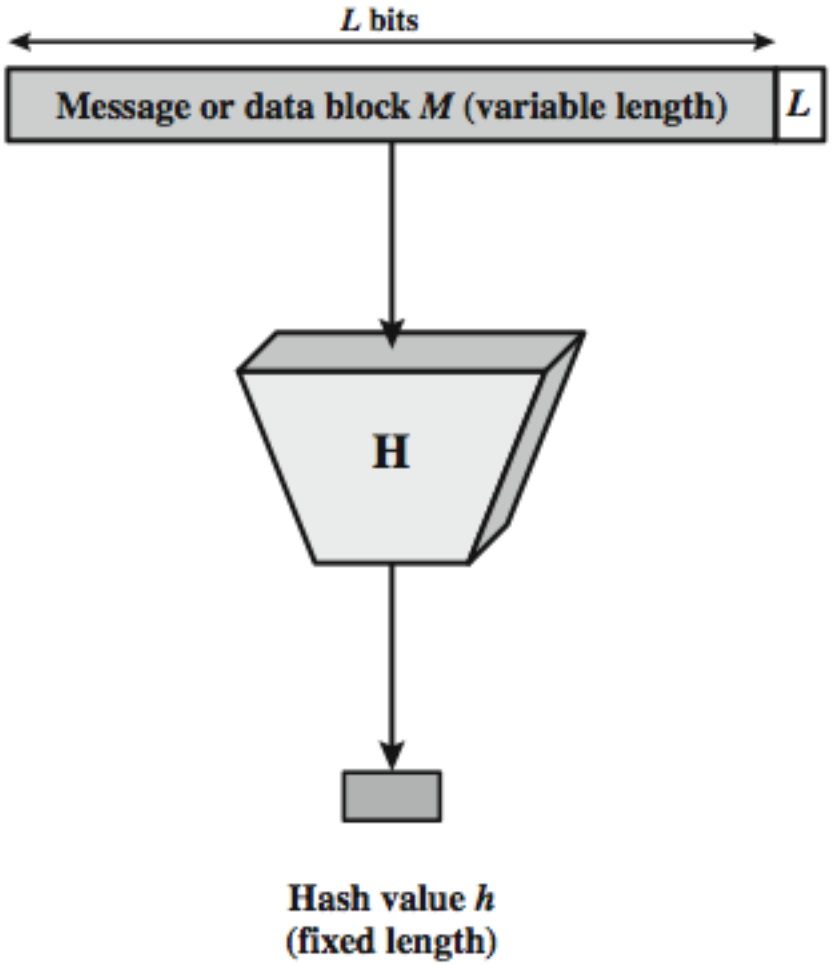
Hash Functions

- condenses arbitrary message to fixed size

```
h = H(M)
```

- usually assume hash function is public
- hash used to detect changes to message
- want a cryptographic hash function
  - computationally infeasible to find data mapping to specific hash (one-way property)
  - computationally infeasible to find two data to same hash (collision-free property)

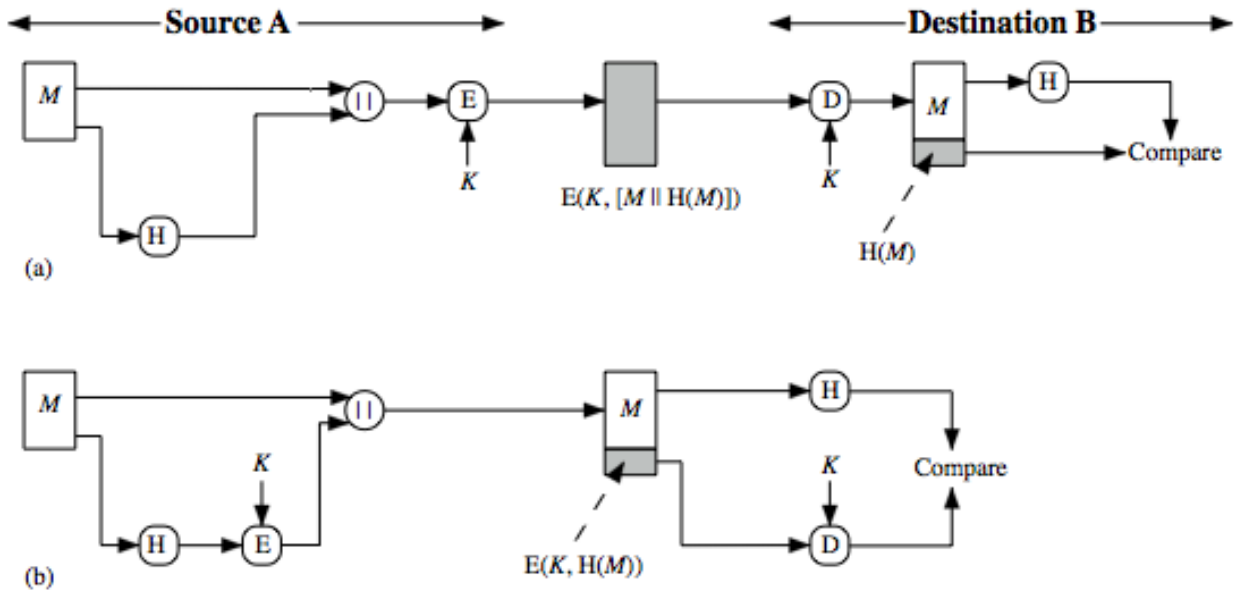
Cryptographic Hash Function



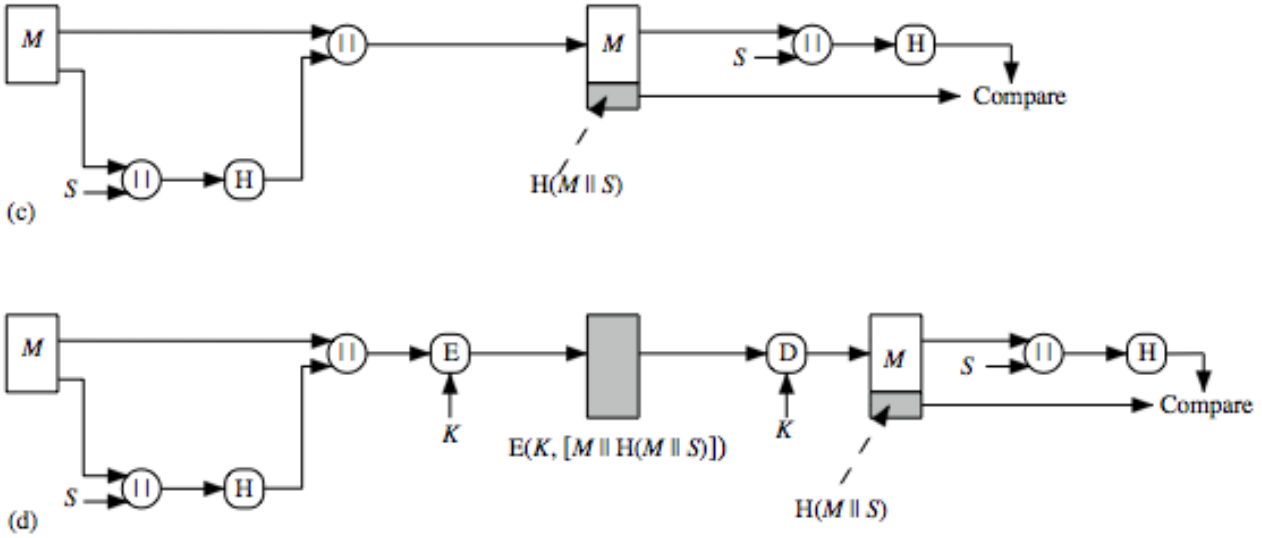
Hash Function Uses

- Message Integrity Check (MIC)
  - send hash of message (digest)
  - MIC always encrypted, message optionally
- Message Authentication Code (MAC)
  - send keyed hash of message
  - MAC, message optionally encrypted
- Digital Signature (non-repudiation)
  - Encrypt hash with private (signing) key
  - Verify with public (verification) key

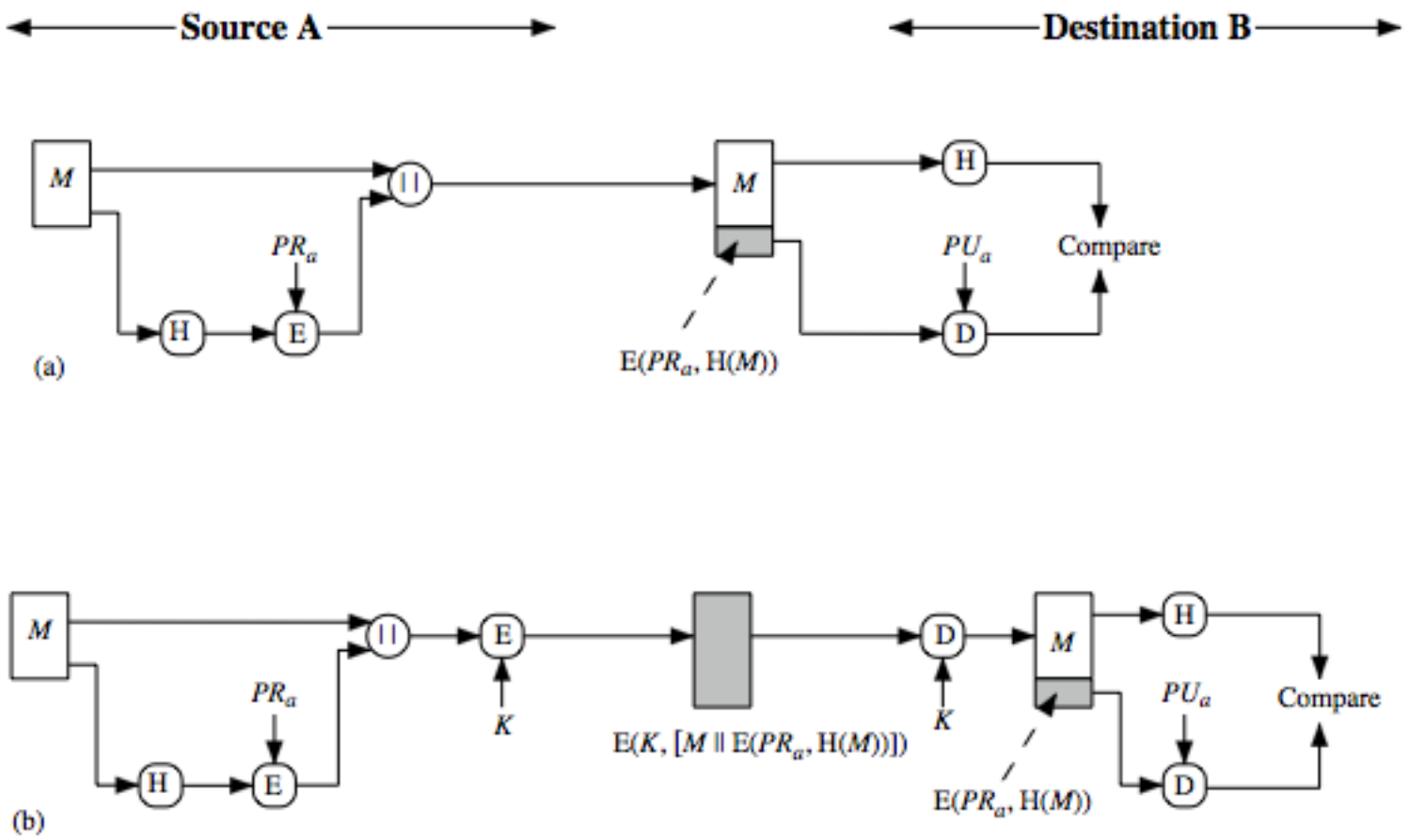
Hash Functions & Message Authentication



Hash Functions & Message Authentication



Hash Functions & Digital Signatures - PKCS



Other Hash Function Uses

- pseudorandom function (PRF)
  - Generate session keys, nonces
  - Produce key from password
  - Derive keys from master key cooperatively
- pseudorandom number generator (PRNG)
  - Vernam Cipher/OTP
  - S/Key, proof of “what you have” via messages

More Hash Function Uses

- to create a one-way password file
  - store hash of password not actual password
  - e.g., Unix, Windows NT, etc.
  - salt to deter precomputation attacks
  - Rainbow tables
- for intrusion detection and virus detection
  - keep & check hash of files on system
  - e.g., Tripwire

Two Simple Insecure Hash Functions

- consider two simple insecure hash functions
- bit-by-bit exclusive-OR (XOR) of every block
  - $C_i = b_{i1}$ *xor* b_{i2} *xor . . . xor* b_{im}
  - a longitudinal redundancy check
  - reasonably effective as data integrity check
- one-bit circular shift on hash value
  - for each successive *n-bit* block
    - rotate current hash value to left by1bit and XOR block
  - good for data integrity but useless for security

What is a hash function?

- Compression: A function that maps arbitrarily long binary strings to fixed length binary strings
- Ease of Computation: Given a hash function and an input it should be easy to calculate the output
- Often used to create a hash table where the hash function computes the index into the table

Hash Function Properties

It is mathematically impossible to extract the original message from the digest.

- Hashing is sometimes referred to as one-way encryption:
  - message can be encrypted but is impossible to decrypt.
  - This is accomplished using one-way functions within the hashing algorithm.
- It is impossible to derive ‘hello’ knowing only a resulting hash value of ’52’. Mostly because there could be thousands of messages that result in the identical output.

Hash Function

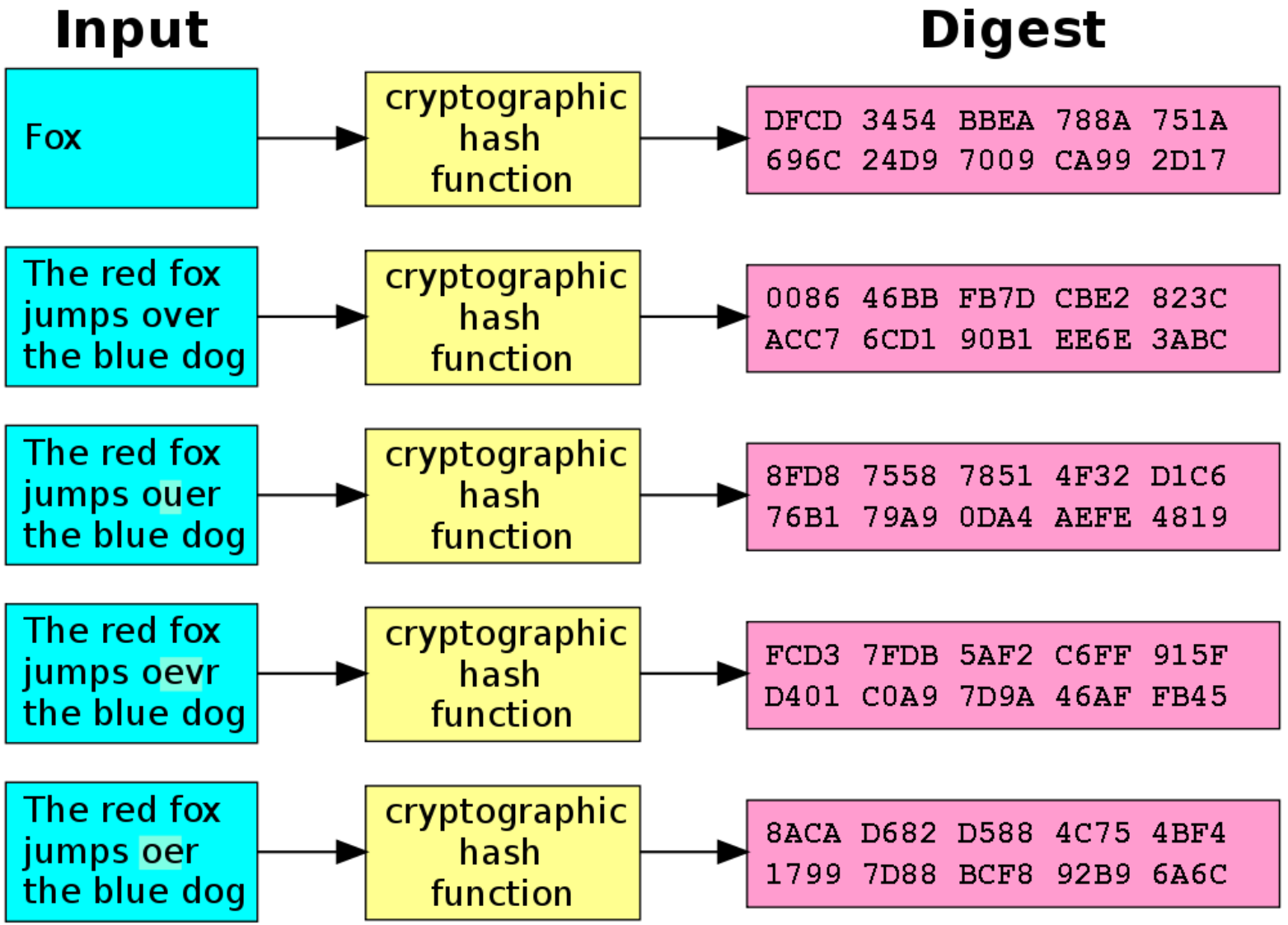
A slight change to the original message causes a drastic change in the resulting digest.

- Any minor modification – even as small as changing a single character – to the original Message should greatly alter the computed hash result
- This is sometimes referred to as the Avalanche effect.
- If for two different messages, hash to same output, then this term is known as Collision.

Hash Function

The result of the hashing algorithm is always the same length.

- It is vital for the resulting hash value to not provide any hints or clues about the original Message – including its length.
- A hash result should not grow in size as the length of the Message increases.



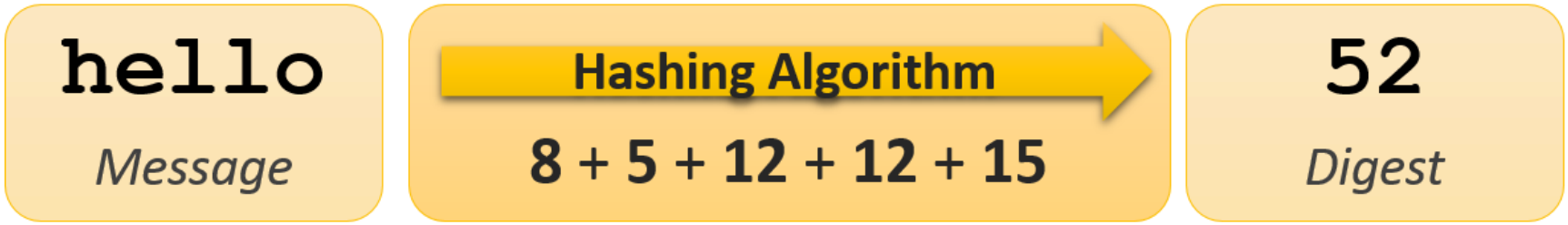
A slight change in input causes larger change in output

Image source: https://en.wikipedia.org/wiki/Cryptographic_hash_function#/media/File:Cryptographic_Hash_Function.svg

Hash Function

It is infeasible to construct a message which generates a given digest.

- In this example, if hash value result = 52 is given, it would not be overly difficult to generate a list of words that might have been the original message.



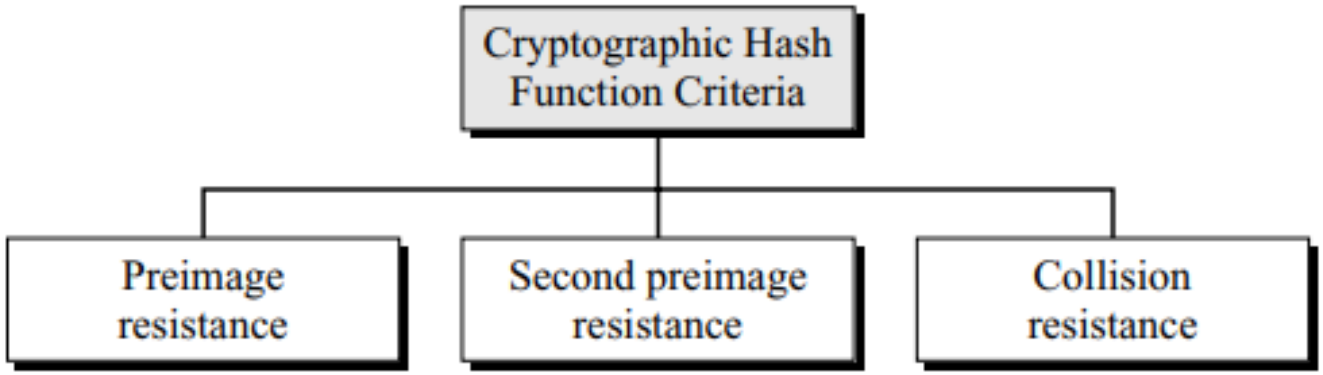
PRACTICAL NETWORKING .NET

Cryptographic Hash Function properties

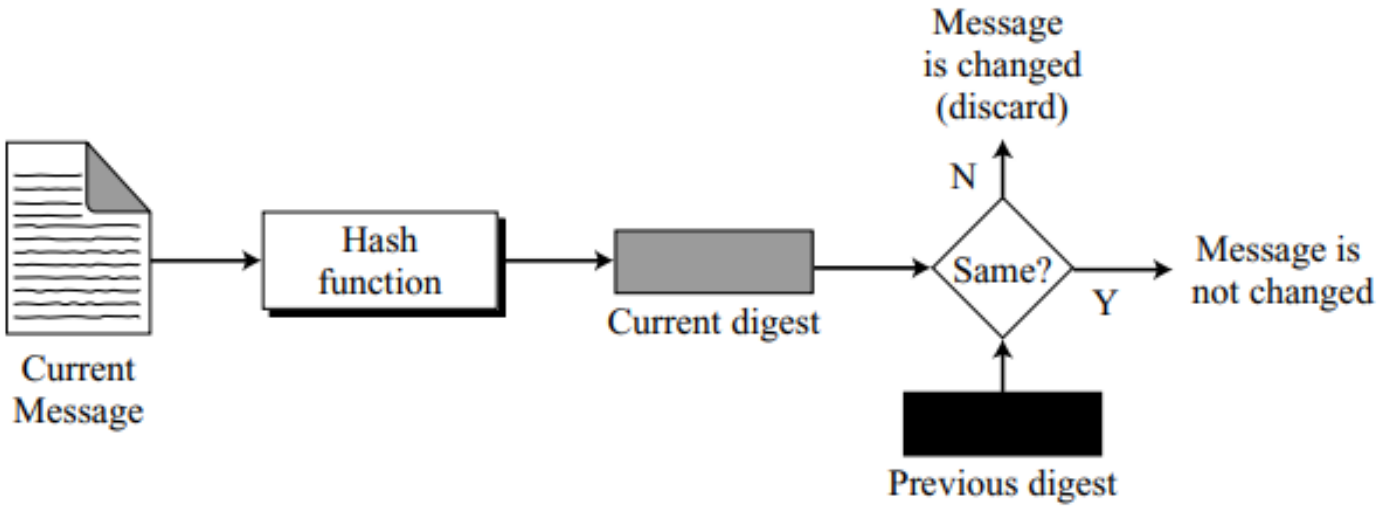
- It is computationally infeasible to find either
  - A data object that maps to a pre-specified hash result (the one-way property)
  - Two data objects that map to the same hash result (the collision-free property)

Cryptographic Hash Function Criteria

- A cryptographic hash function must satisfy three criteria:
  - preimage resistance,
  - Second preimage resistance,
  - collision resistance,



Checking message integrity



Properties of a Hash Function

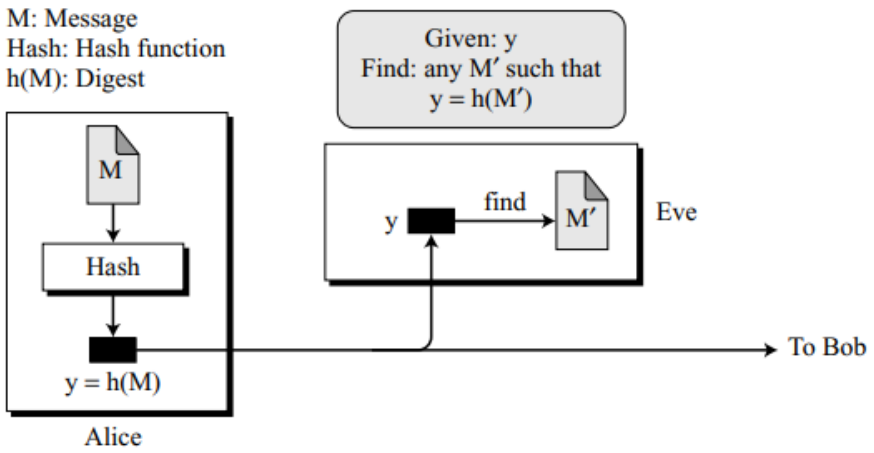
- Preimage Resistance (One Way): For essentially all pre-specified outputs, it is computationally infeasible to find input which hashes to given output.
- Second Preimage Resistance (Weak Col. Res.): It is computationally infeasible to find any second input which has the same output as any specified input.
- Collision Resistance (Strong Col. Res.): It is computationally infeasible to find any two distinct inputs which hash to the same output.

1. Preimage Resistance

- A cryptographic hash function must be preimage resistant.
- Given a hash function h and Y = h(M), it must be extremely difficult for Eve to find the message, M, such that h(M)=Y.

Preimage Attack

Given: y = h(M) **Find: M′ such that y = h(M′)**



Given only a message value, one should not be able to find another input (preimage) that generates same hash result

Preimage Resistance

- If the hash function is not preimage resistant, Eve can intercept the digest h(M) and create a message M′.
- Eve can then send M′ to Bob pretending it is M

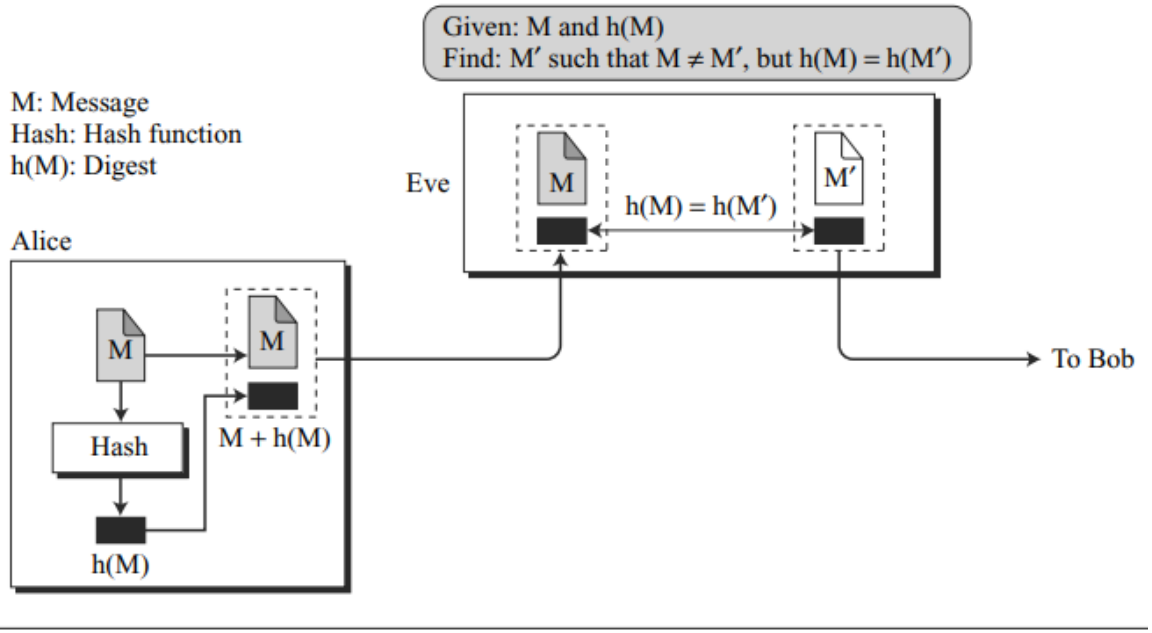
Preimage Resistance

- Computationally hard to reverse a hash function.
- In other words, if a hash function h produced a hash value z, then it should be a difficult process to find any input value x that hashes to z.
- This property protects against an attacker who only has a hash value and is trying to find the input.

2. Second Preimage Resistance

- Second preimage resistance, ensures that a message cannot easily be forged.
- If Alice creates a message and a digest and sends both to Bob, this criterion ensures that Eve cannot easily create another message that hashes to the exact same digest.
- In other words, given a specific message and its digest, it is impossible (or at least very difficult) to create another message with the same digest.

2. Second Preimage Resistance



- Eve intercepts (has access to) a message M and its digest h(M).
- She creates another message M′≠ M, but h(M) = h(M′).
- Eve sends the M′ and h(M′) to Bob.
- Eve has forged the message.

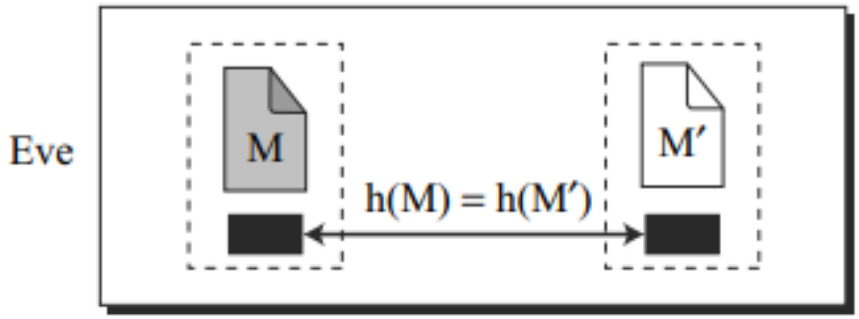
Second preimage resistance

- The property says the following: if I give you an input and the digest it hashes to, you should be unable to find a different input that hashes to the same digest.
- Given message m1, it is difficult to produce another message m2 such that , H(m1) = H(m2).

3. Collision

M: Message
Hash: Hash function
h(M): Digest

Find: M and M′ such that M ≠ M′, but h(M) = h(M′)



Collision Attack	
Given: none	**Find: M′ ≠ M such that h(M) = h(M′)**

- First preimage attack — an attacker can find the original input from a hash.
- Second preimage attack — an attacker is given one input and can find a matching input that results in the same hash

Summary

- An attacker is provided information and must find other information that meets certain criteria
- Preimage Resistance
  - Given: H(M)
  - Find: M
- Second Preimage Resistance
  - Given: H(M) and M
  - Find: M´ such that H(M) = H(M´) and M ≠ M ´
- Collision Resistance
  - Given: nothing
  - Find: M and M´ such that H(M) = H(M´) and M ≠ M

Table 11.1 Requirements for a Cryptographic Hash Function H

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) such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

 Image source: https://www.brainkart.com/article/Requirements-and-Secu

Attacks on Hash Functions

- have brute-force attacks and cryptanalysis
- a preimage or second preimage attack
  - find y s.t. $H(y)$ equals a given hash value
- collision resistance
  - find two messages x & y with same hash so $H(x) = H(y)$
- hence value $2^{m/2}$ determines strength of hash code against brute-force attacks
  - 128-bits inadequate, 160-bits suspect

Birthday Attacks

- might think a 64-bit hash is secure
- but by **Birthday Paradox** is not
- **birthday attack** works thus:
  - given user prepared to sign a valid message x
  - opponent generates $2^{m/2}$ variations x' of x, all with essentially the same meaning, and saves them
  - opponent generates $2^{m/2}$ variations y' of a desired fraudulent message y
  - two sets of messages are compared to find pair with same hash (probability > 0.5 by birthday paradox)
  - have user sign the valid message, then substitute the forgery which will have a valid signature
- conclusion is that need to use larger MAC/hash

Birthday Attacks

Find i and j such that $H(y'_j)=H(x'_i)$

Table takes $O(N^2)$ time

Faster …

Sorted lists take O(NlogN) time

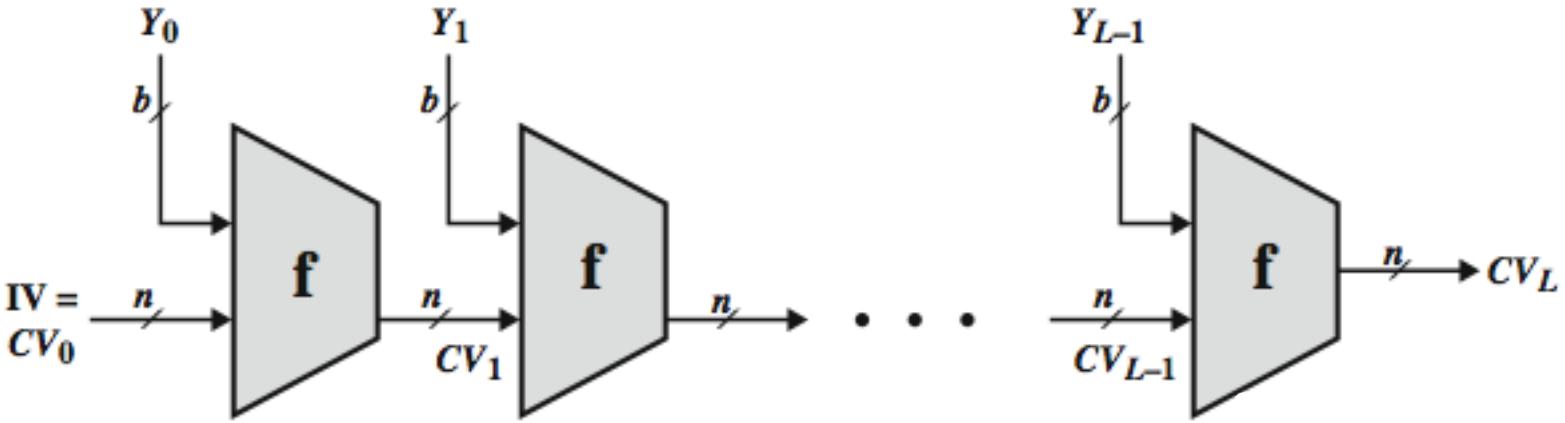
	y	y'_1	y'_2	…	y'_j	…	y'_N
x	≠	≠	≠		≠		≠
x'_1	≠	≠	≠		≠		≠
x'_2	≠	≠	≠		≠		≠
…							
x'_i	≠	≠	≠		=		≠
…							
x'_N	≠	≠	≠		≠		≠

Birthday Attacks

- What are chances we get a match?
- N distinct values, k randomly chosen ones
  - P(N,i) = prob(i randomly selected values from 1..N have at least one match)
  - P(N,2) = 1/N
  - P(N,i+1) = P(N,i)+(1-P(N,i))(i/N)
- For P(N,k)>0.5, need $k \approx N^{1/2}$
- Need double # bits in hash value

Hash Function Cryptanalysis

- cryptanalytic attacks exploit some property of alg so faster than exhaustive search
- hash functions use iterative structure
  - process message in blocks (incl length)
- attacks focus on collisions in function f



Cipher Block Chaining as Hash Functions

- can use block ciphers as hash functions
  - using H_0=0 and zero-pad of final block
  - compute: $H_i = E_{M_i} [H_{i-1}]$
  - and use final block as the hash value
  - similar to CBC but without a key
- resulting hash is too small (64-bit)
  - both due to direct birthday attack
  - and to “meet-in-the-middle” attack
- other variants also susceptible to attack

Block Ciphers as Hash Functions

Block cipher key length B

Pad Message M to multiple of B

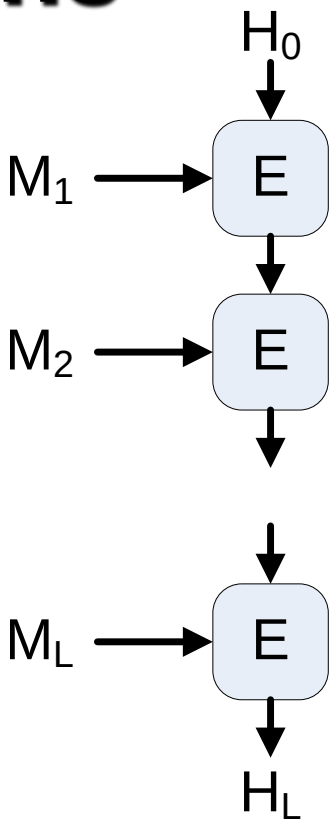
Break padded M into L blocks

L = |M|/B

$M = M_1 M_2 \ldots M_L$

Use blocks of M as keys in block cipher, iteratively encrypt state value starting with constant H_0 resulting in hash value

$H = H_L = E(M_L,\ldots.E(M_2,E(M_1,H_0))\ldots)$



Secure Hash Algorithm

- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- based on design of MD4 with key differences
- produces 160-bit hash values
- 2005 results on security of SHA-1 raised concerns on its use in future applications

Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

Hash function families

- Two widely used families the MD family and the SHA family
- Rivest and RSA laboratories developed MD4 and now MD5.
- The original MD was never published; MD2 was the first of the family to appear, and it was followed by MD4.
- The NSA developed SHA-1 and SHA-2. Around February 2005, problems with SHA-1 became public.

Hash function families

- Message digest family
  - Designed by Ron Rivest
  - MD2, MD4, MD5
- Secure hash algorithm family
  - Designed by NIST
  - SHA-0, SHA-1 AND SHA-2
    - SHA-2: SHA-224, SHA-256, SHA-384, SHA-512
    - SHA-3 : new standard in competition, released in August 2015
- Race Integrity Primitive Evaluation Message Digest (RIPEMD)

MD Family

MD4 family of Hash Functions

Algorithm	Output [bit]	Input [bit]	No. of rounds	Collisions found
MD5	128	512	64	yes
SHA-1	160	512	80	not yet
SHA-224	224	512	64	no
SHA-256	256	512	64	no
SHA-384	384	1024	80	no
SHA-512	512	1024	80	no

SHA versions

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
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
# of steps	80	64	64	80	80

Why need of a new hash standard?

- Md5 and SHA-0 are already broken
- SHA-1 is not fully yet broken
  - But it is similar to MD5 and SHA-0, so can be broken
  - So considered insecure
- SHA-2, especially SHA-512 seems secure
  - Shares same structure and mathematical operations as predecessors, so is matter of concern

MD5

- A cryptographic hash function
- commonly used to verify
  - data integrity and
  - generate checksums.
- Steps:
  - Input Padding
  - Dividing input into blocks
  - IV initialization
  - Block processing
  - Update hash
  - Final hash

MD 5 working

1. Message Input: a text string, a file, or any other set of binary data.
2. Padding:
   - MD5 works on 512-bit (64-byte) blocks of data.
   - Padding ensures that the data can be divided into fixed-size blocks.
3. Dividing the Message: The padded message is divided into 512-bit blocks.
4. Initialize Constants:
   - MD5 uses a set of constants to mix and transform data during the hashing process.
   - These constants are predefined within the MD5 algorithm.
5. Initialization Vector (IV): predefined hexadecimal numbers.

MD 5 working

6. Processing Blocks:

- Each block is processed one at a time.
- block is divided into 32-bit words.
- A set of complex bitwise and logical operations is performed on these words in multiple rounds (usually 64 rounds).
- 16 operations in 4 rounds
- Different function on each round i.e. for the 1st round we apply the F function, for the 2nd G function, 3rd for the H function, and 4th for the I function.
- Operations: bitwise operations (AND, OR, XOR), modular addition, and specific logical functions.
- Each round combines the previous round's result with the current block's data.

Rounds	Process P
1	(b AND c) OR ((NOT b) AND (d))
2	(b AND d) OR (c AND (NOT d))
3	b OR c OR d
4	c OR (b OR (NOT d))

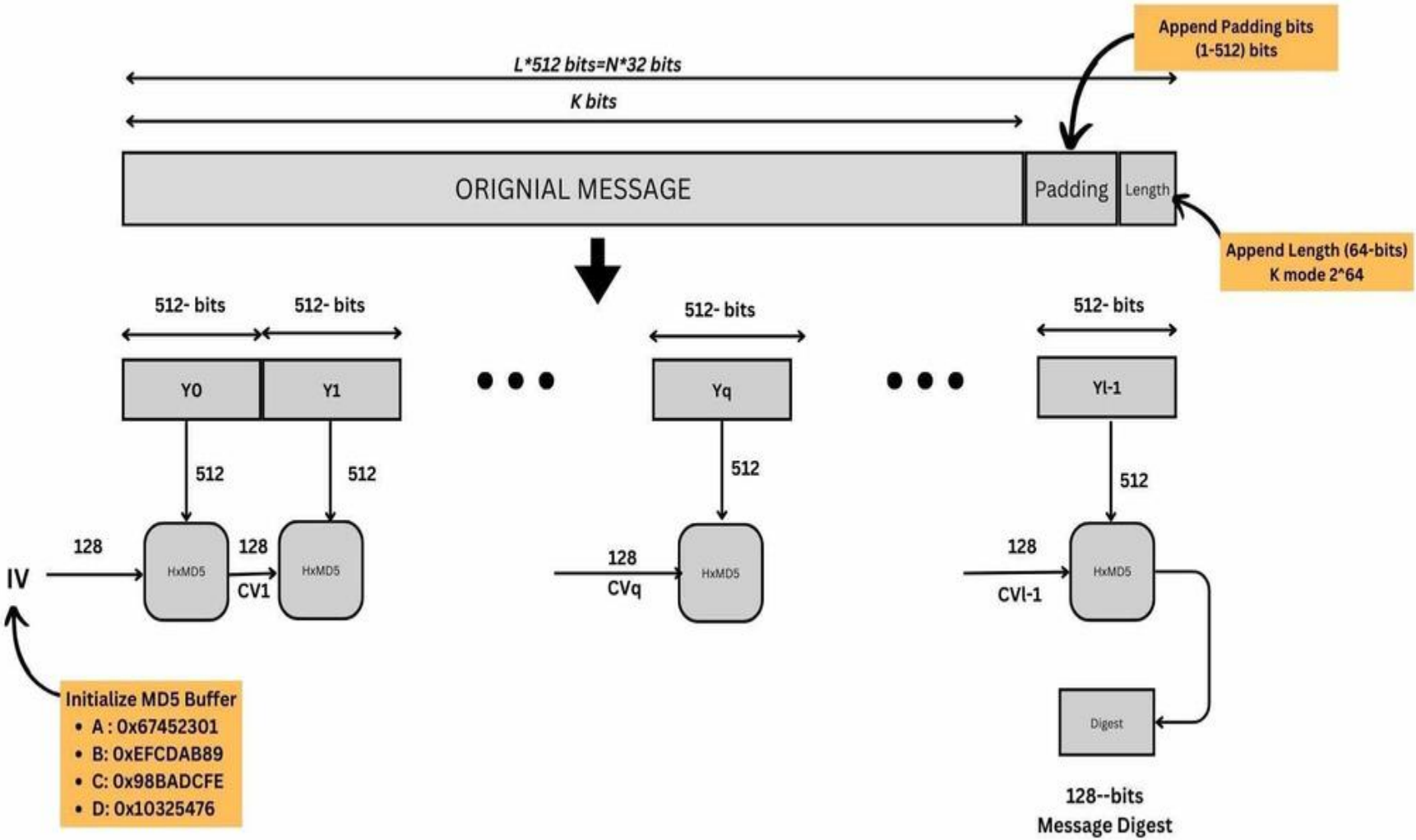
MD 5 working

7. Updating the Hash:

- As each block is processed, the hash value is updated.
- MD5 has four 32-bit variables (A, B, C, and D) that represent the current hash value.
- These variables are updated in each round of processing.

8. Final Hash:

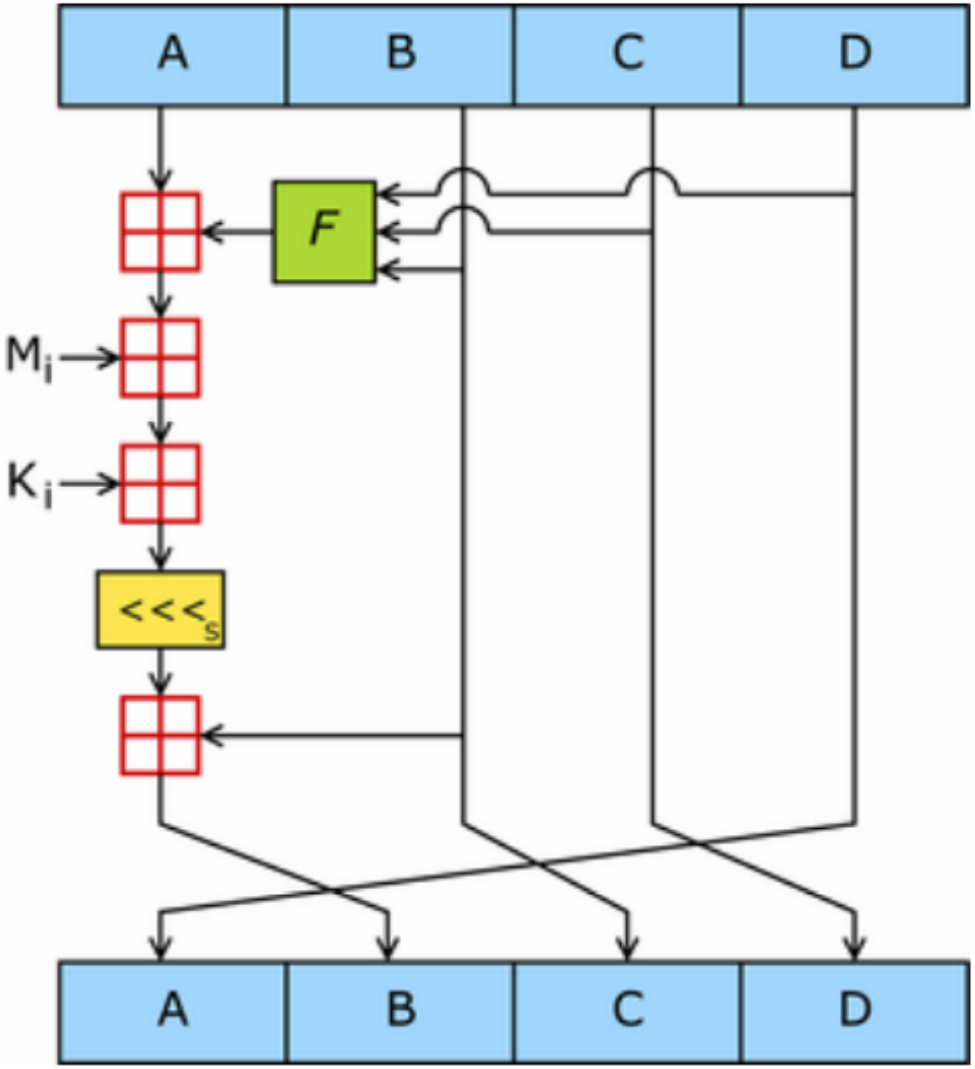
- final hash value is a 128-bit (16-byte) hexadecimal number.
- This value is often represented as a 32-character string of hexadecimal digits.


Append Padding bits
(1-512) bits
L*512 bits=N*32 bits
K bits
ORIGNIAL MESSAGE
Padding
Length
Append Length (64-bits)
K mode 2^64
512- bits
512- bits
512- bits
512- bits
Y0
Y1
Yq
Yl-1
512
512
512
512
128
128
128
128
IV
HxMD5
HxMD5
HxMD5
HxMD5
CV1
CVq
CVl-1
Initialize MD5 Buffer
• A : 0x67452301
• B: 0xEFCDAB89
• C: 0x98BADCFE
• D: 0x10325476
Digest
128--bits
Message Digest

The MD5 Compression Function

- Each iteration consists of 4 rounds of 16 steps each, each step processing 32-bits of the message
- Each round processes all 512 of the input bits
- 4 round functions, 1 per round: F(X, Y, Z)
- X,Y, Z are predefined
- 4 internal 32-bit registers store the current state of the algorithm
- The internal registers are initialized to fixed constants

An MD5 Step



- A – D are the 32-bit internal registers
- F is 1 of 4 functions

$$F_1 = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$F_2 = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$F_3 = X \oplus Y \oplus Z$$
$$F_4 = Y \oplus (X \vee \neg Z)$$

- M_i is the i^{th} message block
- K_i is the i^{th} round constant
- $<<<_s$ is an s-bit rotation to the left

- Now take input as initialize MD buffer i.e. A, B, C, D. Output of B will be fed in C, C will be fed into D, and D will be fed into J. After doing this now we perform some operations to find the output for A.
- In the first step, Outputs of B, C, and D are taken and then the function F is applied to them. We will add modulo 2^{32} bits for the output of this with A.
- In the second step, we add the M[i] bit message with the output of the first step.
- Then add 32 bits constant i.e. K[i] to the output of the second step.
- At last, we do left shift operation by n (can be any value of n) and addition modulo by 2^{32}.
- After all steps, the result of A will be fed into B. Now same steps will be used for all functions G, H, and I. After performing all 64 operations we will get our message digest.

MD 5 security

- MD5 has known vulnerabilities
- is susceptible to collision attacks, where two different inputs can produce the same hash.
- weaknesses make MD5 unsuitable for security-critical applications.

MD 5 strengths

- **Non-Cryptographic Use Cases**:
  - MD5 can still be suitable for non-security-sensitive applications where cryptographic strength is not a primary concern.
  - E.g. checksums and data integrity verification where there is no malicious intent to manipulate the data.
- **Speed**:
  - MD5 is relatively fast
  - advantageous in situations where performance is a significant consideration.
  - It can still be used for quick data deduplication or indexing.
- **Legacy Systems**:
  - Some older systems may still use MD5 for compatibility reasons.
  - While this is not ideal from a security perspective, it may be necessary to work with these systems.

Secure Hash Algorithm-SHA

Secure Hash Algorithm- SHA

- Developed by the National Institute of Standards and Technology (NIST)
- published as a federal information processing standard (FIPS 180) in 1993;
- a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1.
- SHA-1 produces a hash value of 160 bits.
- The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest

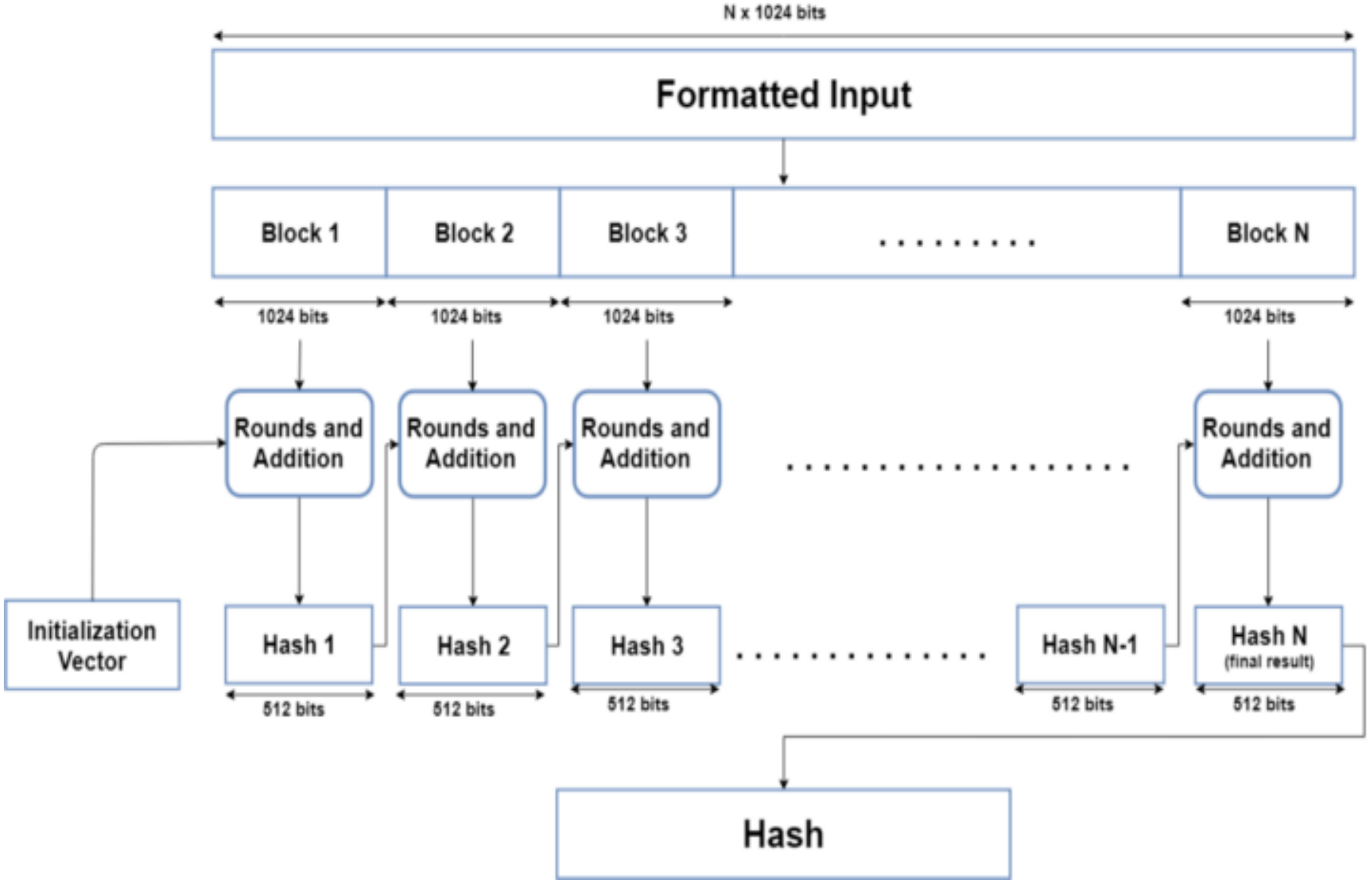
History-Secure Hash Algorithm- SHA

- In 2002, NIST produced a revision of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA512.
- Collectively, these hash algorithms are known as SHA-2. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1.
- In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010.

Features of SHA-512

- **A hashing algorithm used to convert text of any length into a fixed-size string**.
- Each output produces a SHA-512 length of 512 bits (64 bytes).
- This algorithm is commonly used for
  - email addresses hashing
  - password hashing
  - digital record verification.
- With a message digest of 512 bits, SHA-512 expected to be resistant to all attacks, including collision attacks.

SHA 512 working



SHA-512 stages

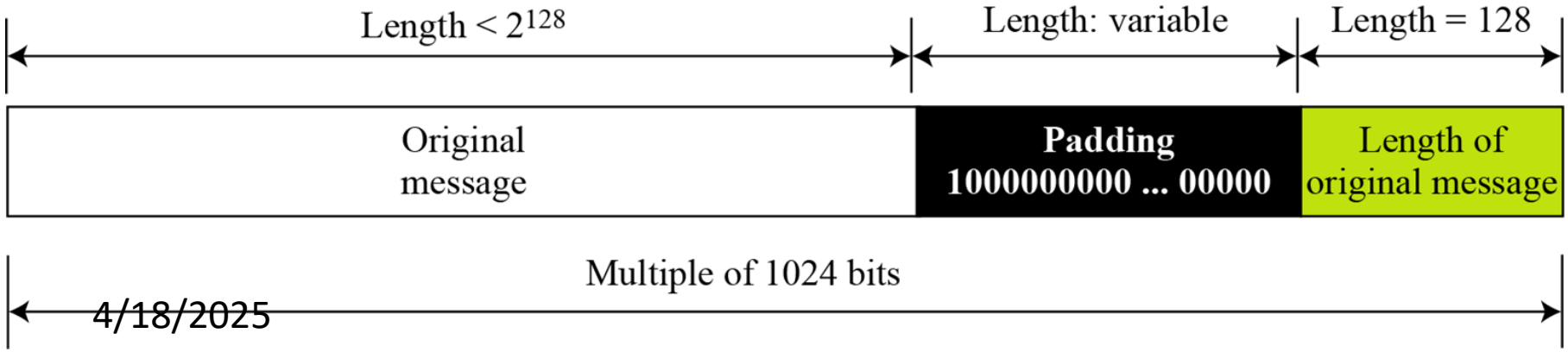
- Input formatting
- Hash buffer initialization
- Message Processing
- Output

Input formatting

- SHA-512 can’t actually hash a message input of any size, i.e. it has an input size limit.
- Message has basically three parts:
  - the original message,
  - padding bits,
  - size of original message.
- Combined size of the whole message must be multiple of 1024 bits.

Input formatting:Padding

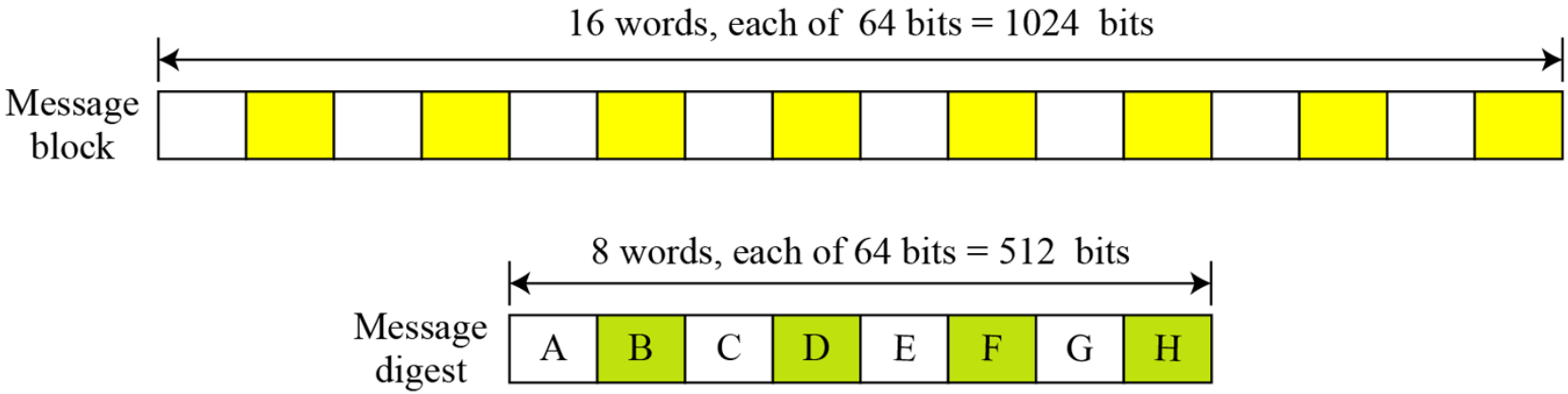
- **Padding bits** to get message of the desired length.
- Padding: ‘0’ bits with a leading ‘1’ (100000…000).
- **Padding size**
- Original message size : represented in 128 bits
  - The only reason that the SHA-512 has a limitation for its input message.



Input formatting

Words

Input message block and output digest as words



Step 2: **Hash buffer initialization**

- Processes each block of 1024 bits from the message using the result from the previous block.
- first block?
  - Uses a Initialization Vector
- Store intermediate result for processing the next block in *hash buffer*
- Also holds the final hash digest of the entire processing phase of SHA-512 as the last of these ‘intermediate’ results.

Initialization vector

IV is initialized with first 64 bits of the fractional parts of the square roots of the first 8 prime numbers (2,3,5,7,11,13,17,19).

Initialization Vector

a = 0x6A09E667F3BCC908	b = 0xBB67AE8584CAA73B
c = 0x3C6EF372FE94F82B	d = 0xA54FF53A5F1D36F1
e = 0x510E527FADE682D1	f = 0x9B05688C2B3E6C1F
g = 0x1F83D9ABFB41BD6B	h = 0x5BE0CD19137E2179

Step 3: **Message Processing**

- Message processing is done upon the formatted input by taking one block of 1024 bits at a time.
- The actual processing uses:
  - The 1024 bit block, and
  - the result from the previous processing.
- This part of the SHA-512 algorithm consists of several ‘Rounds’ and an addition operation.
- So, the Message block (1024 bit) is expanded out into ‘Words’ using a ‘message sequencer’.
- 16 Words to be precise, each of them having a size of 64 bits.

Message processing Rounds

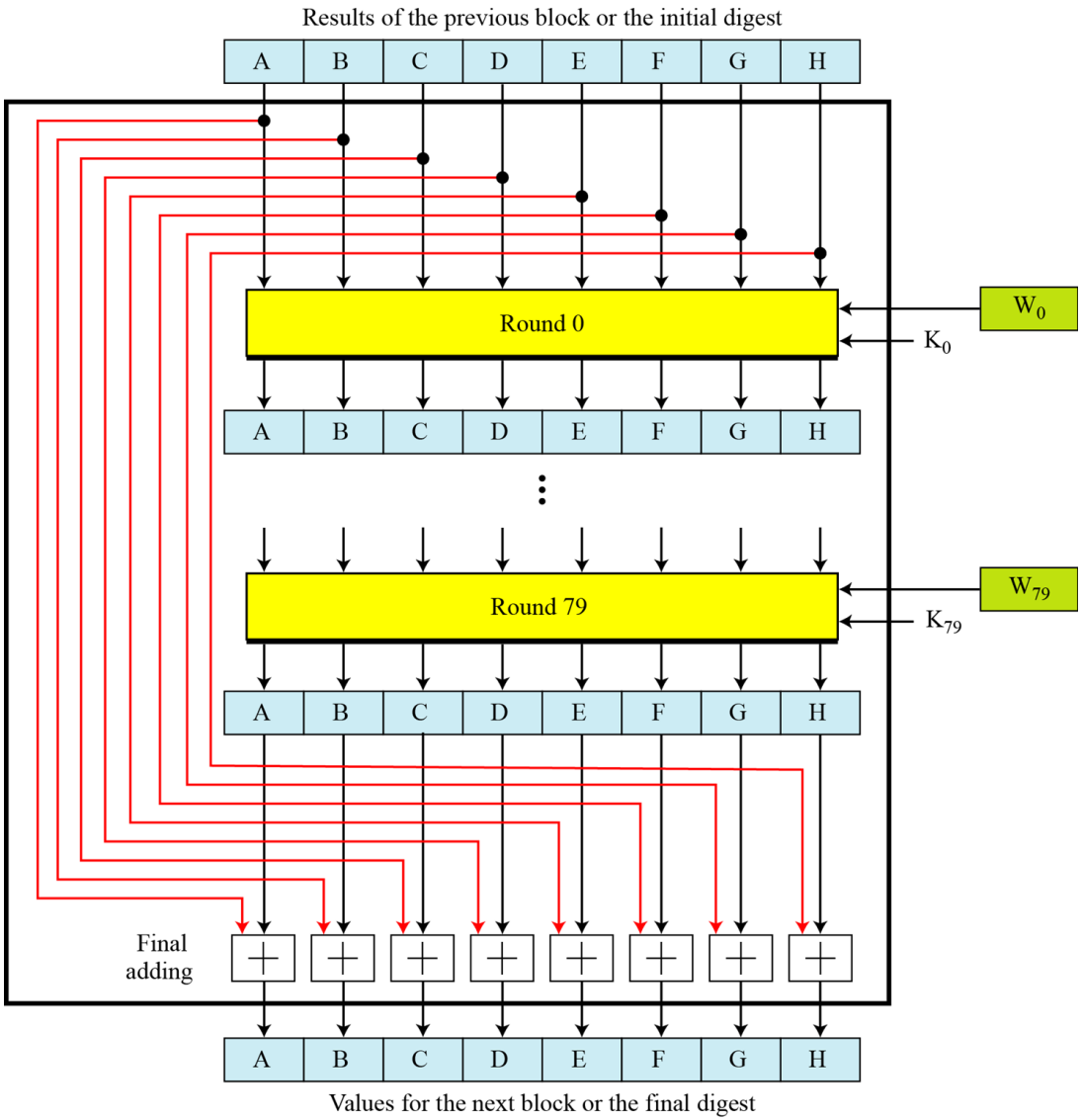
- Each round takes 3 things:
  - one Word,
  - the output of the previous Round,
  - and a SHA-512 constant.
- The first Round uses the Initial Vector (IV)
- SHA-512 constants are predetermined values
  - First 64 bits from the fractional part of the cube roots of the first 80 prime numbers, one for each round
- Every round gives an output of 512 bits.
- Total rounds = 80 Rounds.
- After the 80th Round, its output is simply added to the result of the previous message processing phase to get the final result for this iteration of message processing.

Reading slide

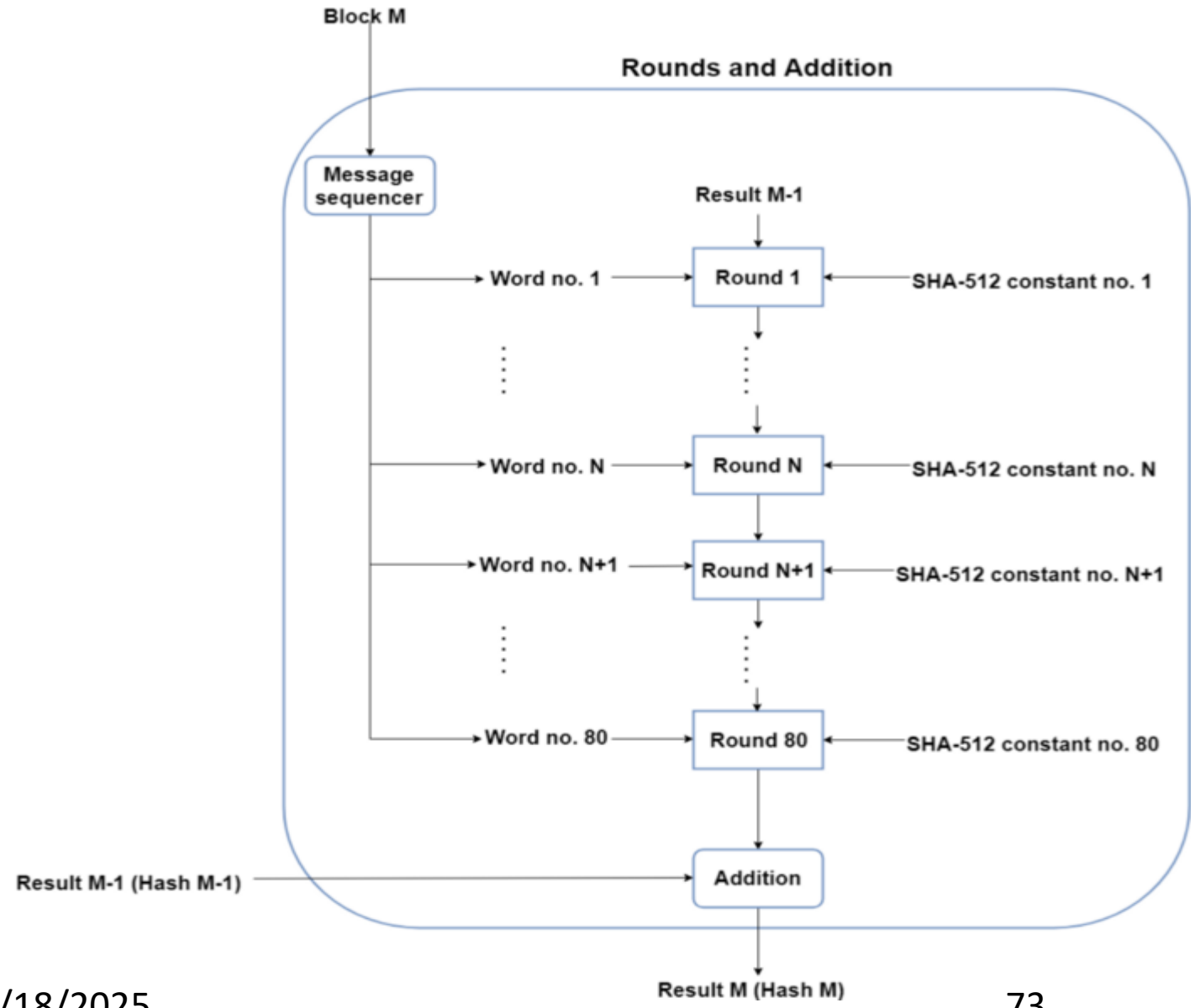
Table 12.3 *Eighty constants used for eighty rounds in SHA-512*

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

Compression function in SHA-512

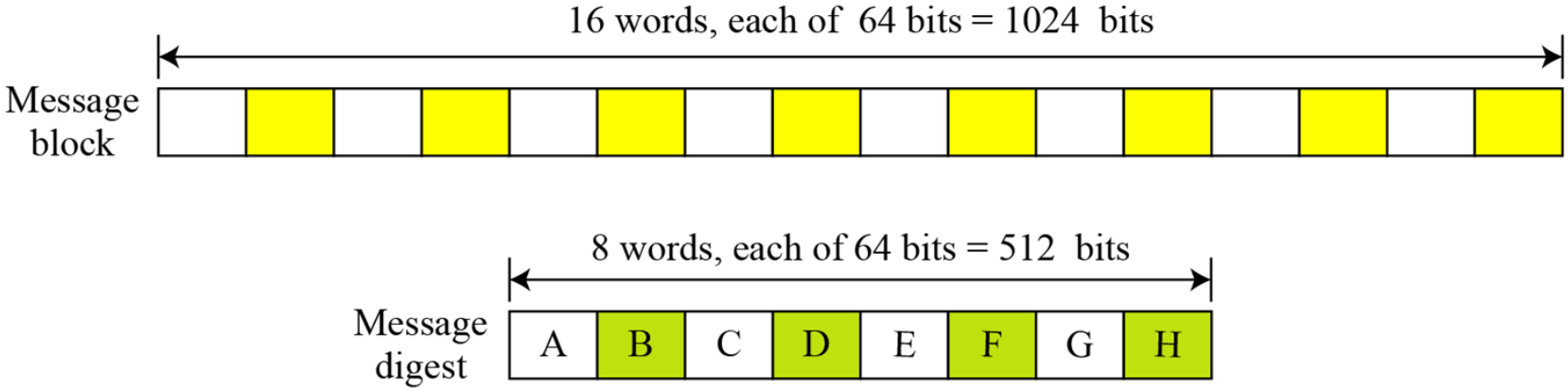


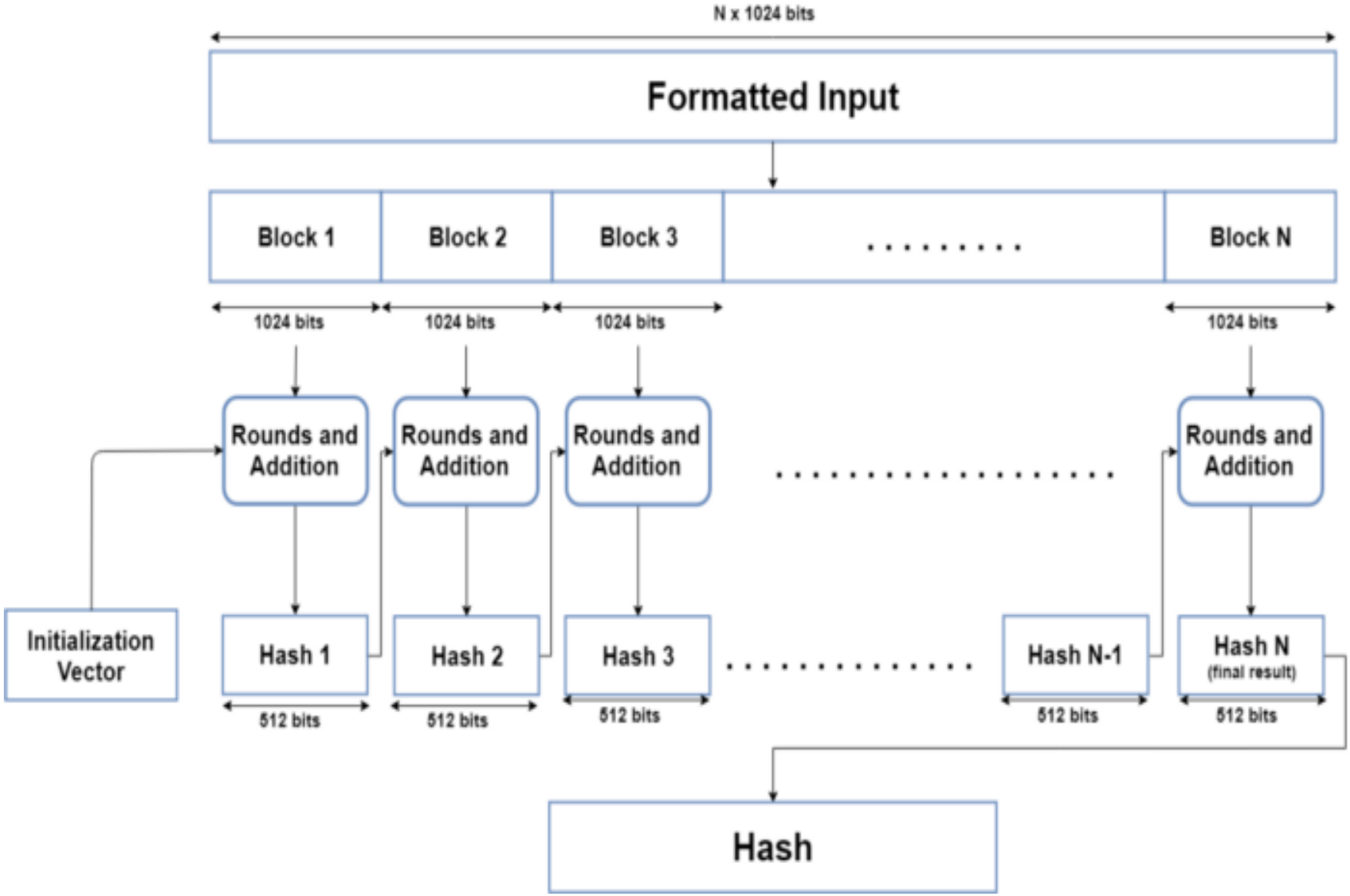
Simplified compression function



Step 4 : Output

- After every block of 1024 bits goes through the message processing phase, the final 512 bit Hash value is generated
- So, the intermediate results are all used from each block for processing the next block.




N x 1024 bits
Formatted Input
Block 1
Block 2
Block 3
. . . . . .
Block N
1024 bits
1024 bits
1024 bits
1024 bits
Rounds and Addition
Rounds and Addition
Rounds and Addition
Rounds and Addition
. . . . . .
Initialization Vector
Hash 1
Hash 2
Hash 3
. . . . . .
Hash N-1
Hash N
(final result)
512 bits
512 bits
512 bits
512 bits
512 bits
Hash

Blockchain Projects That Use The SHA 512

- AlgorithmBitShares (BTS)
- LBRY Credits (LBC)
- Kcash (KCASH)

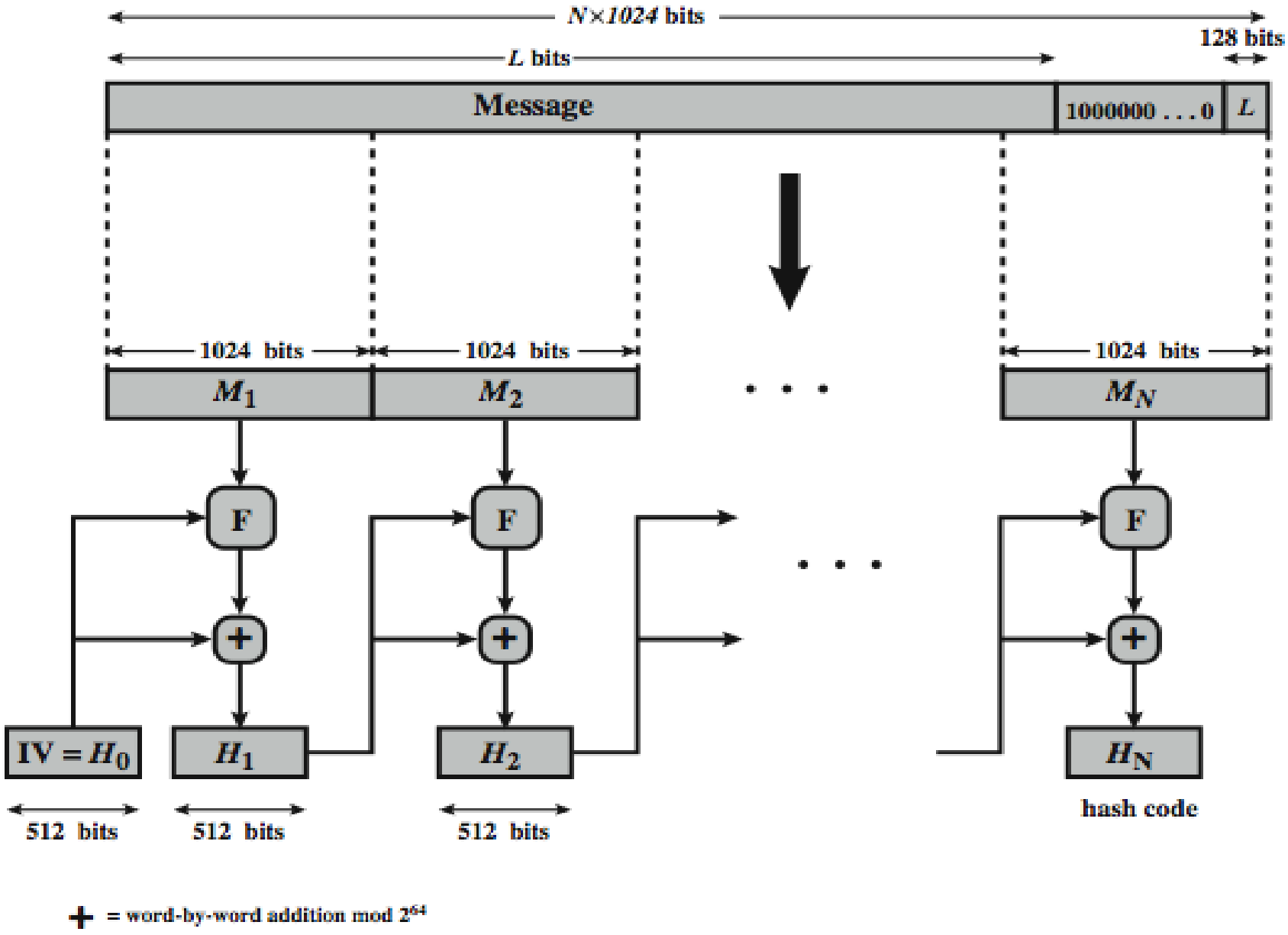
Non-Blockchain Applications For SHA-512

- SHA-512 was used to authenticate archival video from the International Criminal Tribunal of the Rwandan genocide.
- Unix and Linux vendors use both SHA-256 and SHA-512 for secure password hashing.
- An email suppression list solution called OPTIZMO provides the storage and distribution of SHA-512 hashed email addresses for major clients such as Salesforce, LendingTree, Hotwire, and eharmony.

SHA 512

- There are no known attacks against SHA-512, not against **collision resistance** and certainly not against first or second preimage resistance
- **Doesn't have any known vulnerabilities that make it insecure and it has not been "broken" unlike some other popular hashing algorithms**.
- SHA-512 is faster than SHA-256.
- Technically speaking **SHA512 password hashes are not cracked or decrypted** .

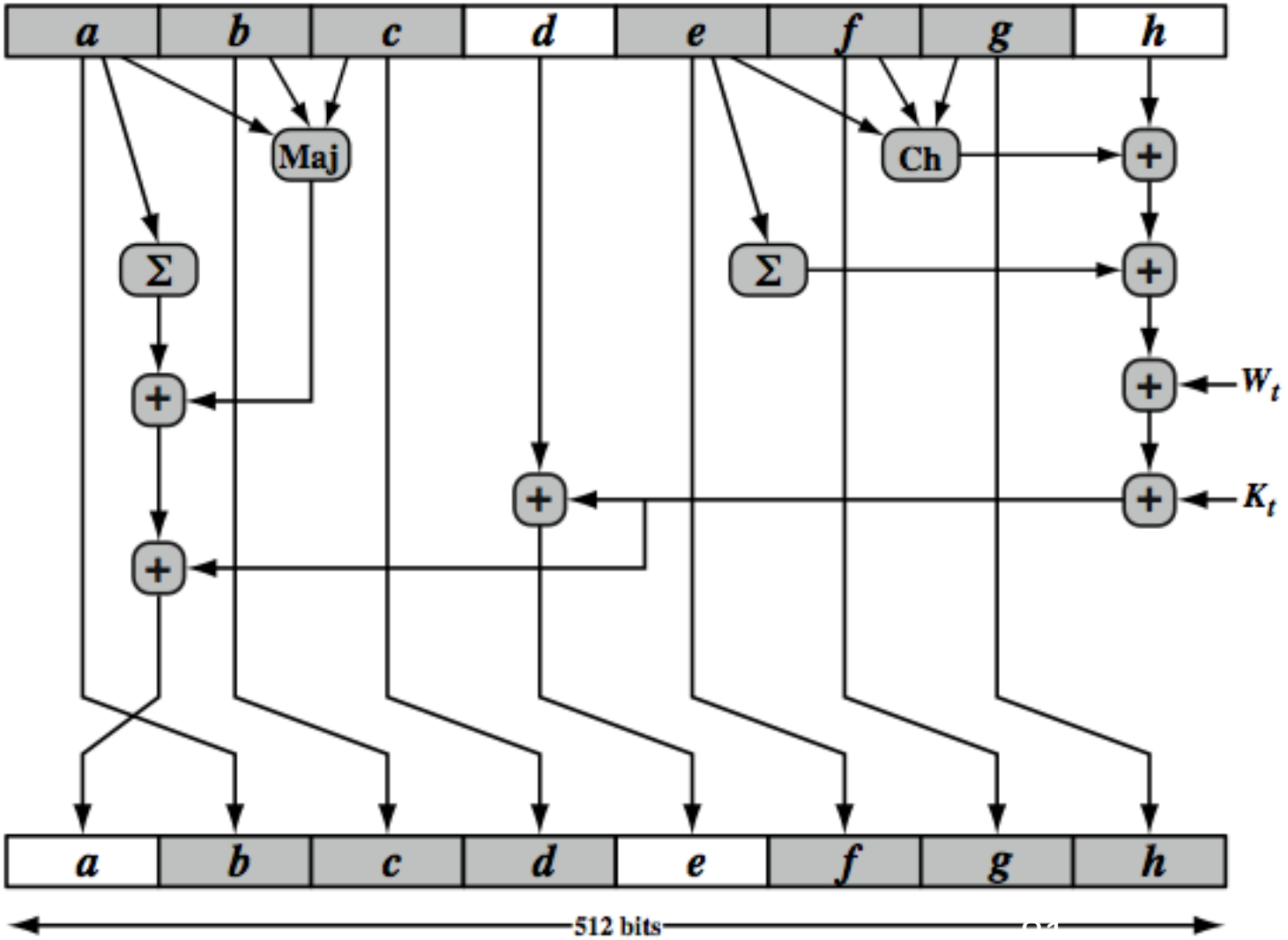
SHA-512 Overview



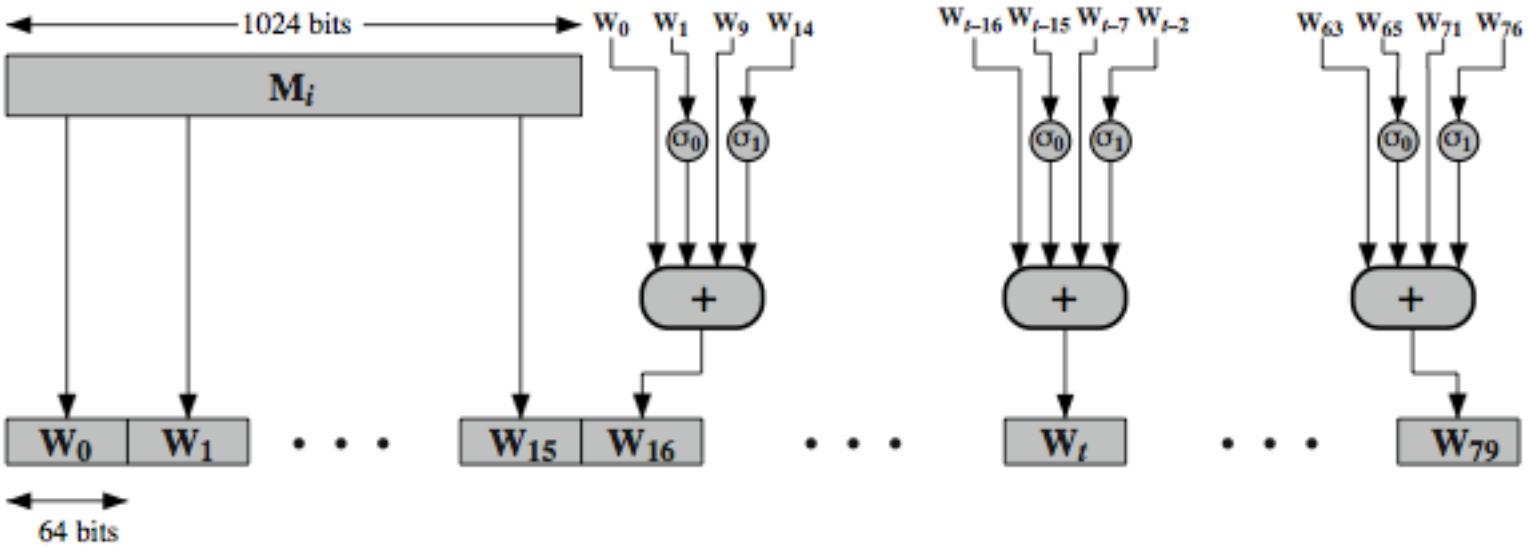
SHA-512 Compression Function

- heart of the algorithm
- processing message in 1024-bit blocks
- consists of 80 rounds
  - updating a 512-bit buffer
  - using a 64-bit value Wt derived from the current message block
  - and a round constant based on cube root of first 80 prime numbers

SHA-512 Round Function



SHA-512 Round Function



SHA-3

- SHA-1 not yet "broken"
  - but similar to broken MD5 & SHA-0
  - so considered insecure
- SHA-2 (esp. SHA-512) seems secure
  - shares same structure and mathematical operations as predecessors so have concern
- NIST announced in 2007 a competition for the SHA-3 next gen NIST hash function
- Keccak winner Oct 2012 – std in Q2,2014

SHA-3 Requirements

- replace SHA-2 with SHA-3 in any use
  - so use same hash sizes
- preserve the online nature of SHA-2
  - so must process small blocks (512 / 1024 bits)
- evaluation criteria
  - security close to theoretical max for hash sizes
  - cost in time & memory
  - characteristics: such as flexibility & simplicity

HMAC- Hash based message authentication code

- uses a hash function in combination with a secret key.
- Used to verify the integrity and authenticity of a message.
- •HMACs are almost similar to digital signatures.
  - They both enforce integrity and authenticity.
  - They both use cryptography keys
  - And they both employ hash functions.
- •The main difference is that digital signatures use asymmetric keys, while HMACs use symmetric keys (no public key)

HMAC working

- A message M: N blocks of length *b* bits.
- Padding: An input signature is padded to the left of the message
- Padded message is passed as input to a hash function which gives a temporary message digest: MD'
- MD' again is appended to an output signature
- MD'+ o/p signature is processed with a hash function again to get final message digest MD.

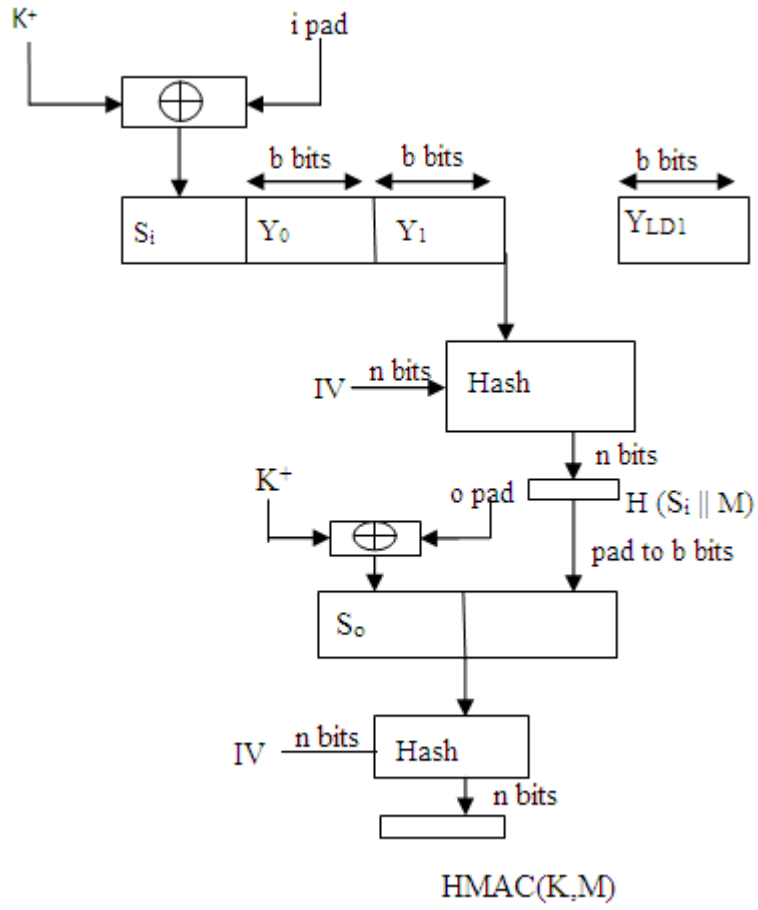
HMAC working

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

HMAC working

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

HMAC structure



H : Hashing function,
M : original message
Si and So : input and output
Yi : the ith block in original message M, where i ranges from [1, L)
L : number of blocks in M
K : secret key used for hashing
IV : initial vector (some constant)

$S_i = K^+ \oplus ipad$

where K^+ is nothing but K padded with zeros on the left so that the result is b bits in length

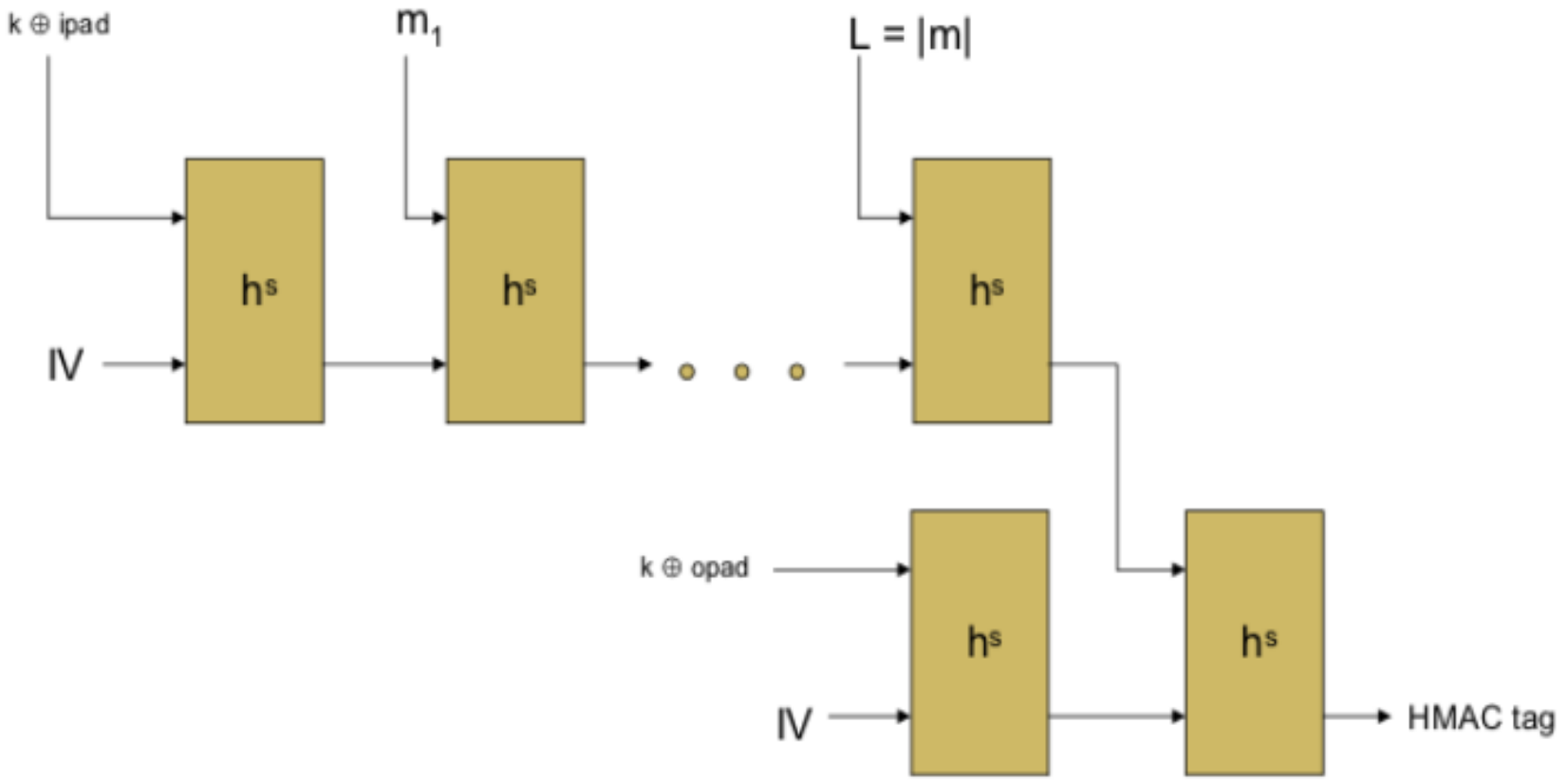
$S_o = K^+ \oplus opad$

where ipad and opad are 00110110 and 01011100 respectively taken b/8 times repeatedly.

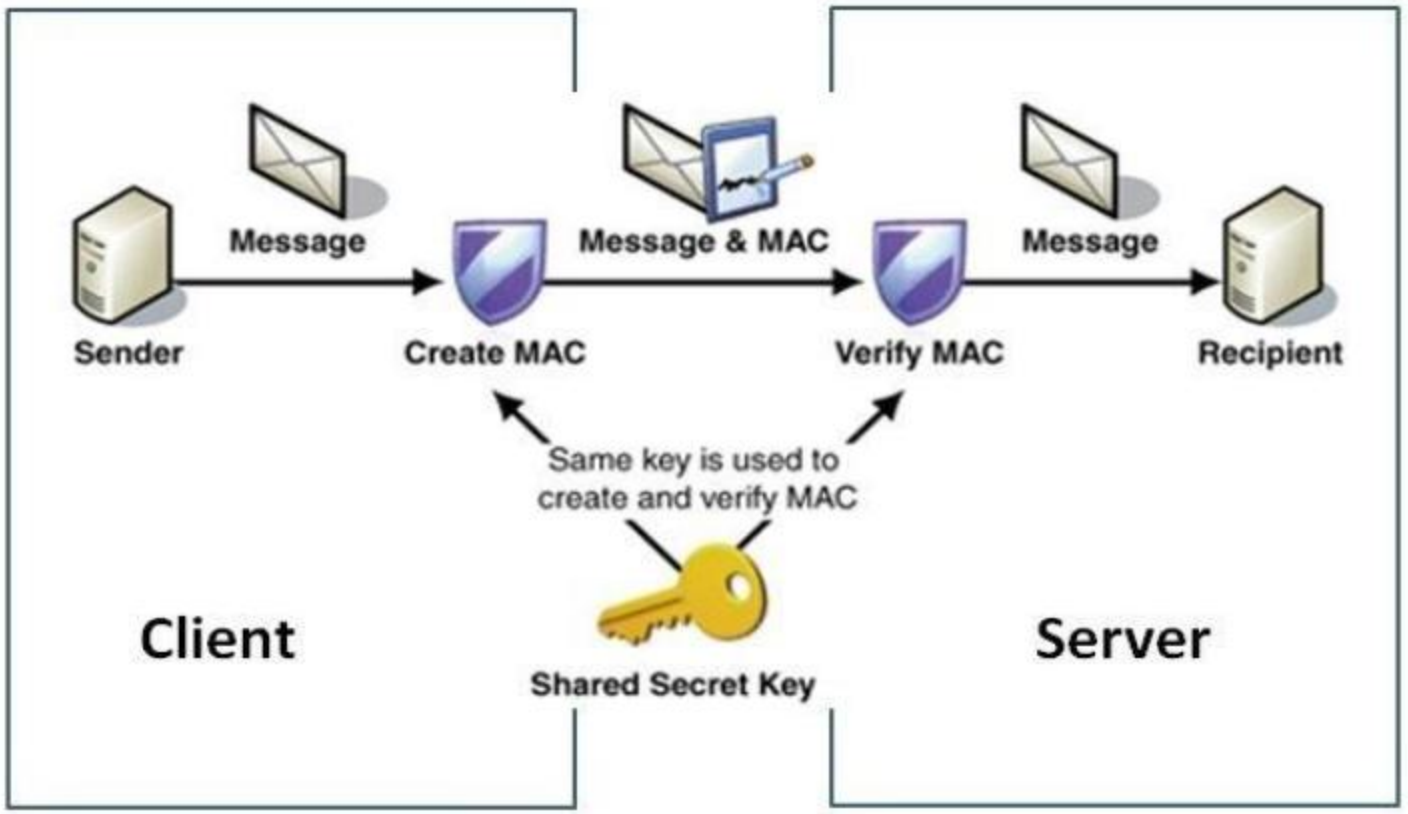
$MD' = H(S_i \,||\, M)$

$MD = H(S_o \,||\, MD')$ or $MD = H(S_o \,||\, H(S_i \,||\, M))$

HMAC Construction



HMAC Authentication



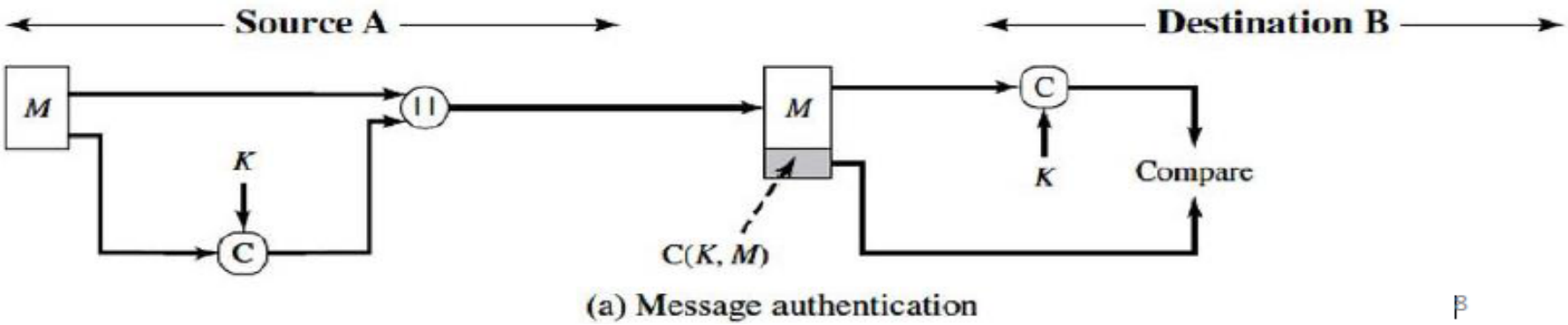
HMAC Security

- Security depends on the cryptographic strength of the underlying hash function
- It is much harder to launch successful collision attacks on HMAC because of secret key

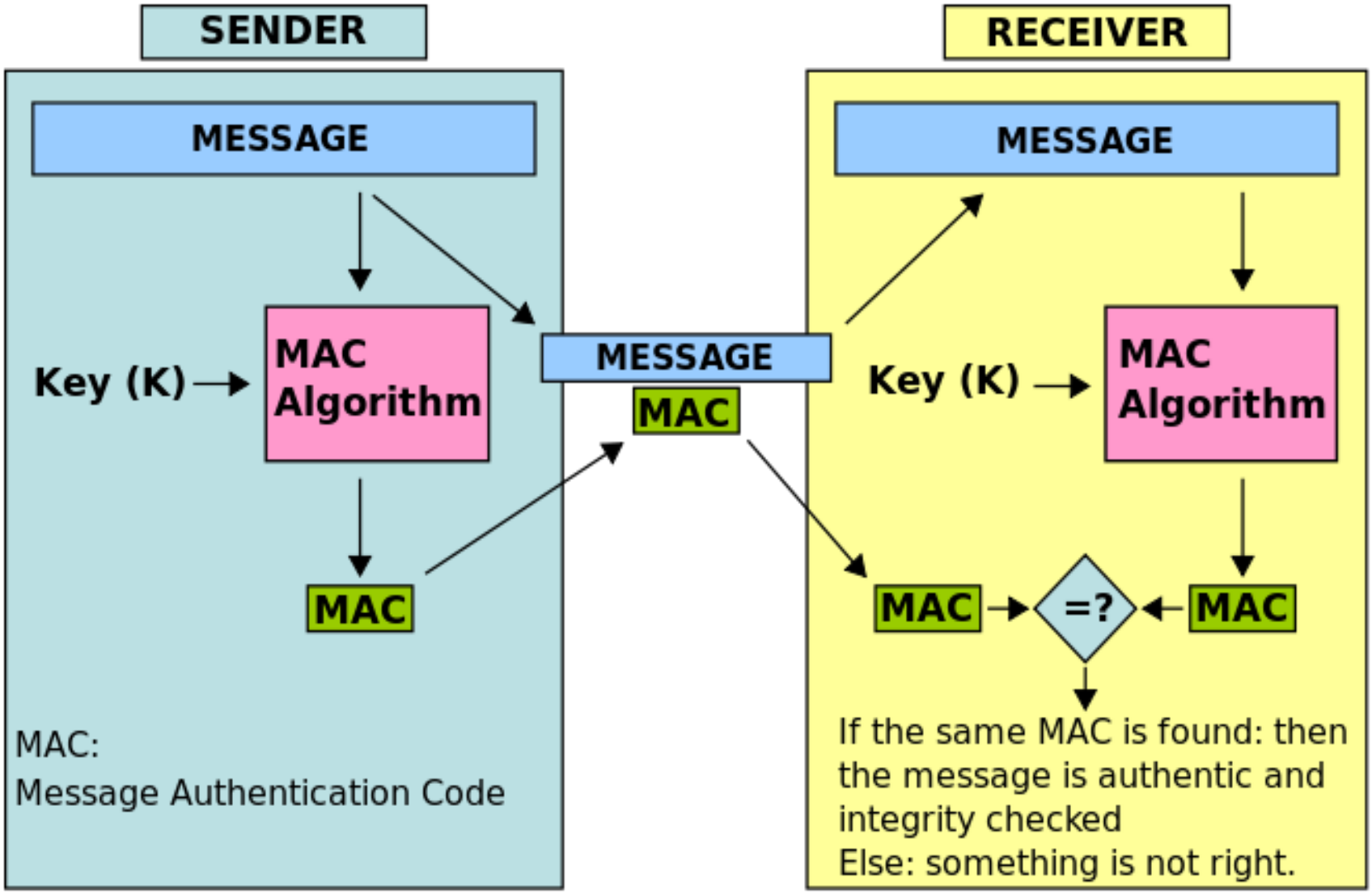
MAC

Message Authentication Code(MAC)

- MAC algorithm is a symmetric key cryptographic technique to provide message authentication.
- For establishing MAC process, the sender and receiver share a Symmetric key K.
- Essentially, a MAC is an encrypted checksum generated on the underlying message that is sent along with a message to ensure message authentication.



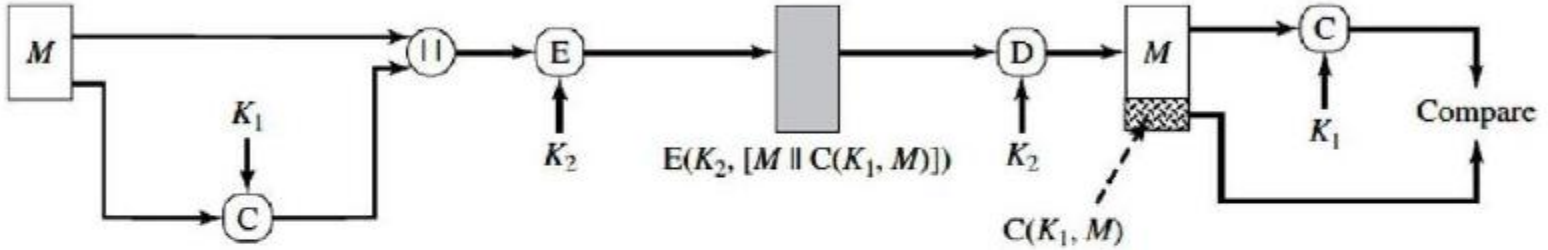
(a) Message authentication


SENDER
RECEIVER
MESSAGE
MESSAGE
Key (K)
MAC Algorithm
MESSAGE
MAC
Key (K)
MAC Algorithm
MAC
MAC
=?
MAC
MAC:
Message Authentication Code
If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.

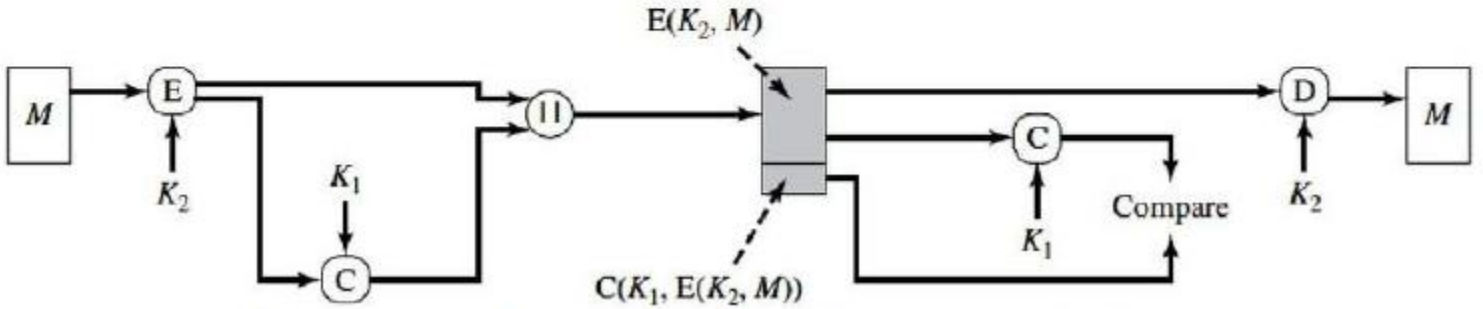
Message Authentication Code(MAC) Process:

- The sender uses MAC algorithm, inputs the message and the secret key K and produces a MAC value.
- Similar to hash, MAC function also compresses an arbitrary long input into a fixed length output.
- The sender forwards the message along with the MAC.
- On receipt of the message and the MAC, the receiver feeds the received message and the shared secret key K into the MAC algorithm and re-computes the MAC value.
- The receiver now checks equality of freshly computed MAC with the MAC received from the sender. If they match, then the receiver accepts the message and assures himself that the message has been sent by the intended sender.
- If the computed MAC does not match the MAC sent by the sender, the receiver cannot determine whether it is the message that has been altered or it is the origin that has been falsified.
- As a bottom-line, a receiver safely assumes that the message is not the genuine.
- No confidentiality but assures message origin authentication.

MAC with Confidentiality and Authentication



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



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

Message Authentication Code

- To realize and construct MAC algorithms, two different cryptographic primitives are used.
- MACs can be implemented using cryptographic hash functions or using symmetric block ciphers.
- Cryptographic hash functions-HMAC
- Symmetric block ciphers–
  - Data Authentication Algorithm,
  - Cipher based Message Authentication Code (CMAC)
    - –CBC–MAC(it uses AES)and
    - –variant of CBC –MAC is CMAC (AES+tripleDES)

MAC Based on block ciphers

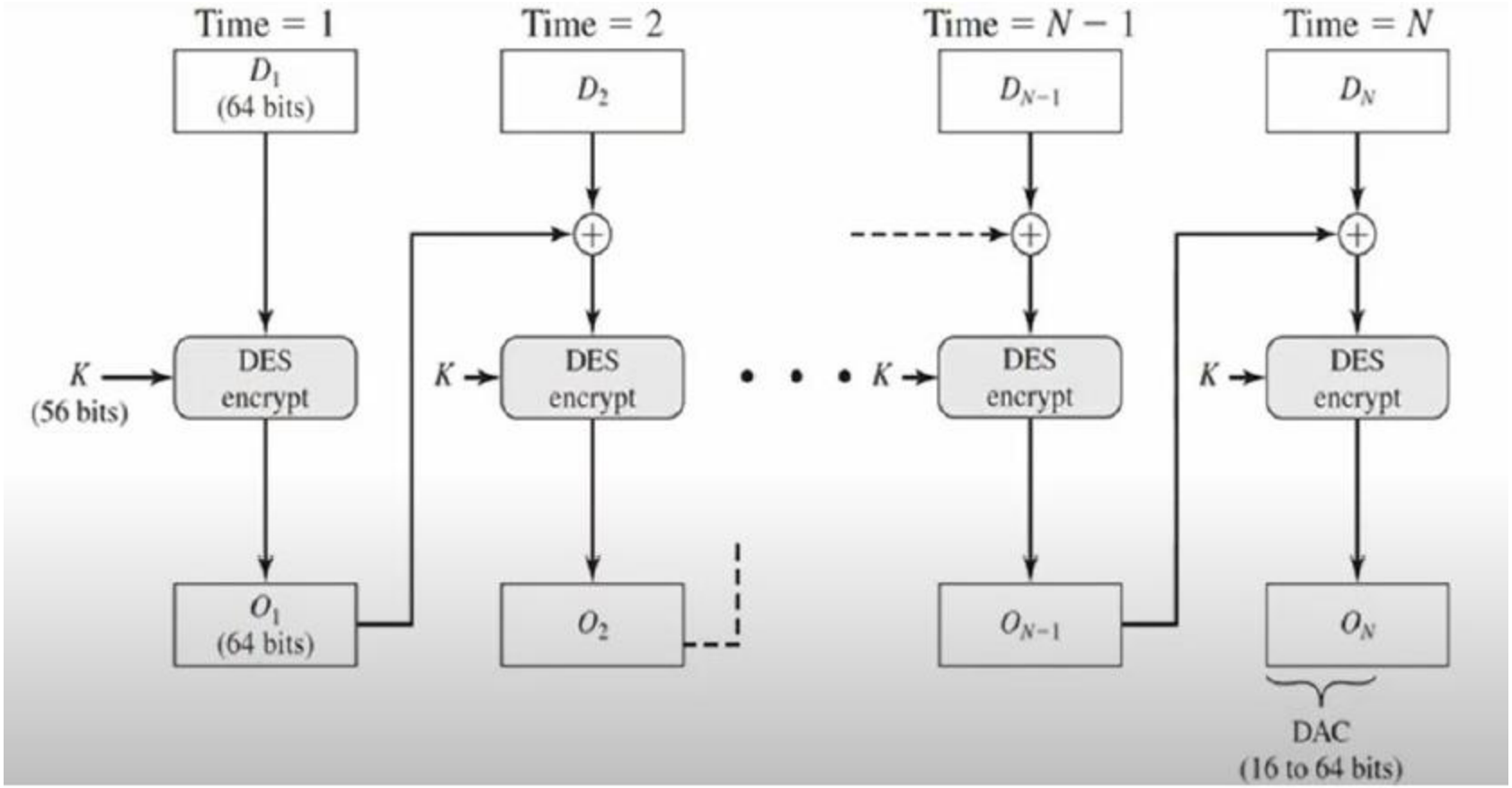
- Data Authentication Algorithm (DAA)
- Cipher based Message Authentication Code (CMAC)

DAA

- The Data Authentication Algorithm (DAA) is a former U.S. government standard for producing cryptographic MACs.
- DAA was withdrawn in September 2008.
- The algorithm is not considered secure by today's standards.
- According to the standard, a code produced by the DAA is called a Data Authentication Code (DAC).
- The algorithm chain encrypts the data, with the last cipher block truncated and used as the DAC.

- Data Authentication Algorithm (DAA) widely used MAC based on DES-CBC (Cipher block chaining)
- The message to be authenticated is grouped into contiguous 64-bit blocks: D1,D2,...,DN.
- The final block is padded on the right with zeros to form a full 64-bit block
- Using DES encryption algorithm E and a secret key K, Data Authentication Code (DAC) is calculated.

Digital Authentication Algo (DAA) working



DAA

$$
\begin{aligned}
O_1 &= \mathrm{E}(K, D) \\
O_2 &= \mathrm{E}(K, [D_2 \oplus O_1]) \\
O_3 &= \mathrm{E}(K, [D_3 \oplus O_2]) \\
&\cdot \\
&\cdot \\
&\cdot \\
O_N &= \mathrm{E}(K, [D_N \oplus O_{N-1}])
\end{aligned}
$$

Reading Assignment

- MAC Vs DAA

Applications for Hash Functions

- Data Integrity
  - Downloading of large files often include the MD5 digest to verify the integrity of the file
- Proof of Ownership
  - Publish the digest of a document describing a patent idea in the New York Times (“All the news that is fit to print”)
- Digital Signatures
  - Digitally sign the digest of a message instead of the message to save computational time

How Secure are Hash Functions?

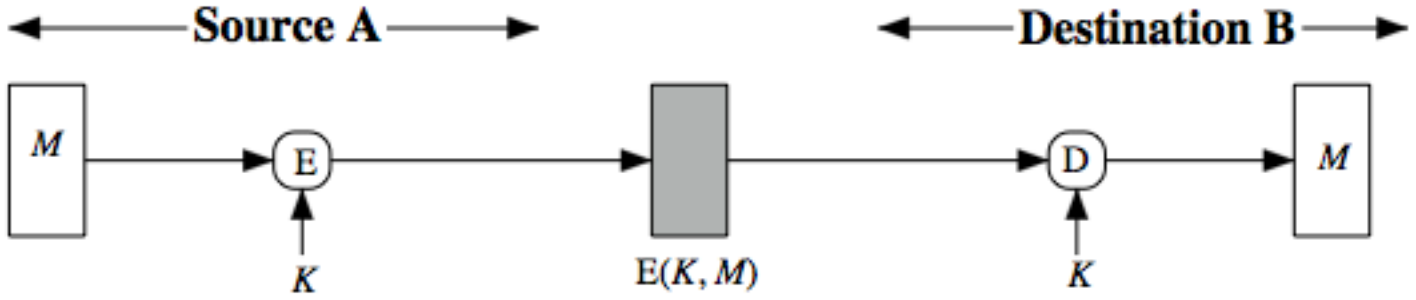
- No hash function is collision free, but it usually takes extremely long to find a collision.
- Even if a hash function has never been broken, a successful attack against a weakened variant may undermine the experts' confidence and lead to its abandonment.
- In the past, weaknesses had been found in several then-popular hash functions, including SHA-0, RIPEMD, and MD5.
- These weaknesses called into question the security of stronger algorithms derived from the weak hash functions such as the SHA-1, RIPEMD-128, and RIPEMD-160.
- Also, there are applications of cryptographic hash functions that do not rely on collision resistance.
- This means that collision attacks do not affect their security.
- For example, HMACs are not vulnerable.
- For the hash attack to be successful, the attacker must be in control of the input to the hash function.

Current State of Hash Functions

- Researchers try to break hash functions in 1 of 2 ways
  - Finding two messages (usually single message blocks) hash to the same digest
  - Find the message that creates the digest of all zeros or all ones (essentially finding a preimage)
- Collisions and preimages can be found for MD4
- Collisions can be constructed for MD5 and SHA-0
- Theoretic collisions discovered for SHA-1
- SHA-256 & SHA-512 have no known attacks

Symmetric Message Encryption

- encryption can also provides authentication
- if symmetric encryption is used then:
  - receiver know sender must have created it
  - since only sender and receiver know key used
  - know content cannot have been altered...
  - ... if message has suitable structure, redundancy or a suitable checksum to detect any changes



(a) Symmetric encryption: confidentiality and authentication

Public-Key Message Encryption

- if public-key encryption is used:
  - encryption provides no confidence of sender
    - since anyone potentially knows public-key
  - however if
    - sender **signs** message using their private-key
    - then encrypts with recipients public key
    - have both secrecy and authentication
  - again need to recognize corrupted messages
  - but at cost of two public-key uses on message

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

Public-Key Message Encryption

- Dirty little detail on PKCS
  - Every time you encrypt, size expands
  - Due to protections in PKCS#1
- So signing (by encryption) then encrypting, the size is more than doubled!

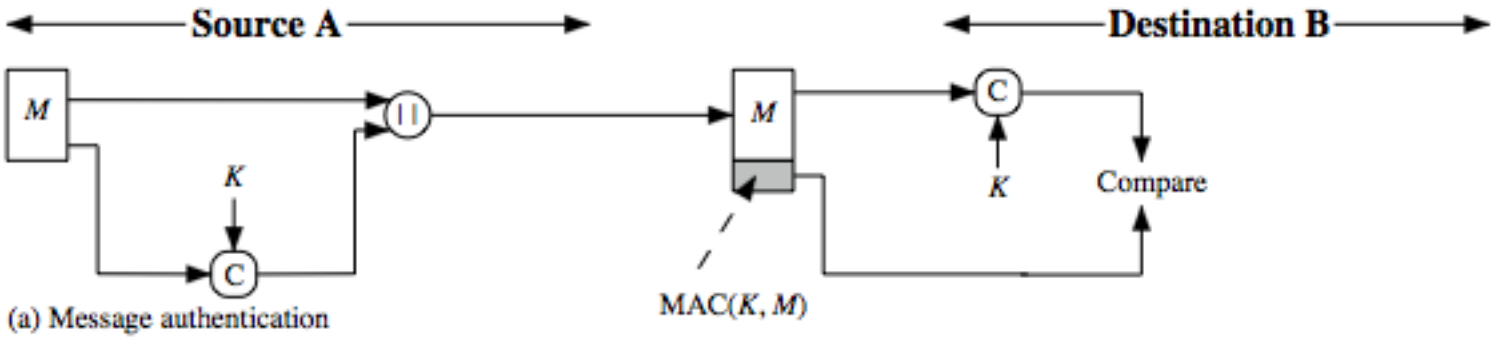
(d) Public-key encryption: confidentiality, authentication, and signature

Message Authentication Code (MAC)

- generated by an algorithm that creates a small fixed-sized block
  - depending on both message and secret key
  - like encryption though need not be reversible
- appended to message as a "**signature**"
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender

Message Authentication Code

- a small fixed-sized block of data
  - generated from message + secret key
  - MAC = C(K,M)
  - appended to message when sent



Message Authentication Codes

- as shown the MAC provides authentication
- can also use encryption for secrecy
  - generally use separate keys for each
  - can compute MAC either before or after encryption
  - is generally regarded as better done before, but see Generic Composition

Message Authentication Codes

- why use a MAC?
  - sometimes only authentication is needed
  - sometimes need authentication to persist longer than the encryption (e.g. archival use)
- note that a MAC is not a digital signature
  - Does NOT provide non-repudiation

MAC Properties

- a MAC is a cryptographic checksum

  $MAC = C_K(M)$

  - condenses a variable-length message M
  - using a secret key K
  - to a fixed-sized authenticator
- is a many-to-one function
  - potentially many messages have same MAC
  - but finding these needs to be very difficult

Requirements for MACs

- taking into account the types of attacks
- need the MAC to satisfy the following:
  1. knowing a message and MAC, 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

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

Security of MACs

- cryptanalytic attacks exploit structure
  - like block ciphers want brute-force attacks to be the best alternative
- more variety of MACs so harder to generalize about cryptanalysis

Keyed Hash Functions as MACs

- want a MAC based on a hash function
  - because hash functions are generally faster
  - crypto hash function code is widely available
- hash includes a key along with message
- original proposal:

```
KeyedHash = Hash(Key|Message)
```

  - some weaknesses were found with this
- eventually led to development of HMAC

Problem with Keyed Hash

- **`KeyedHash = Hash(Key|Message)`**
- Recall hash function works on blocks
- Let M = Key | Message | Padding and M

  $M=M_1\ M_2\ \ldots\ M_L$, where $|M_i|$ = Blocksize

  $Hash=H(H(\ldots H(H(IV,M_1),M_2),\ldots,M_L)$
- But can add extra block(s) M_{L+1} by

  $Hash'=H(Hash,M_{L+1})$
- Unless formatting prevents it…

  … but still best to use HMAC!

HMAC Design Objectives

- use, without modifications, hash functions
- allow for easy replacement of embedded hash function
- preserve original performance of hash function without significant degradation
- use and handle keys in a simple way.
- have well understood cryptographic analysis of authentication mechanism strength

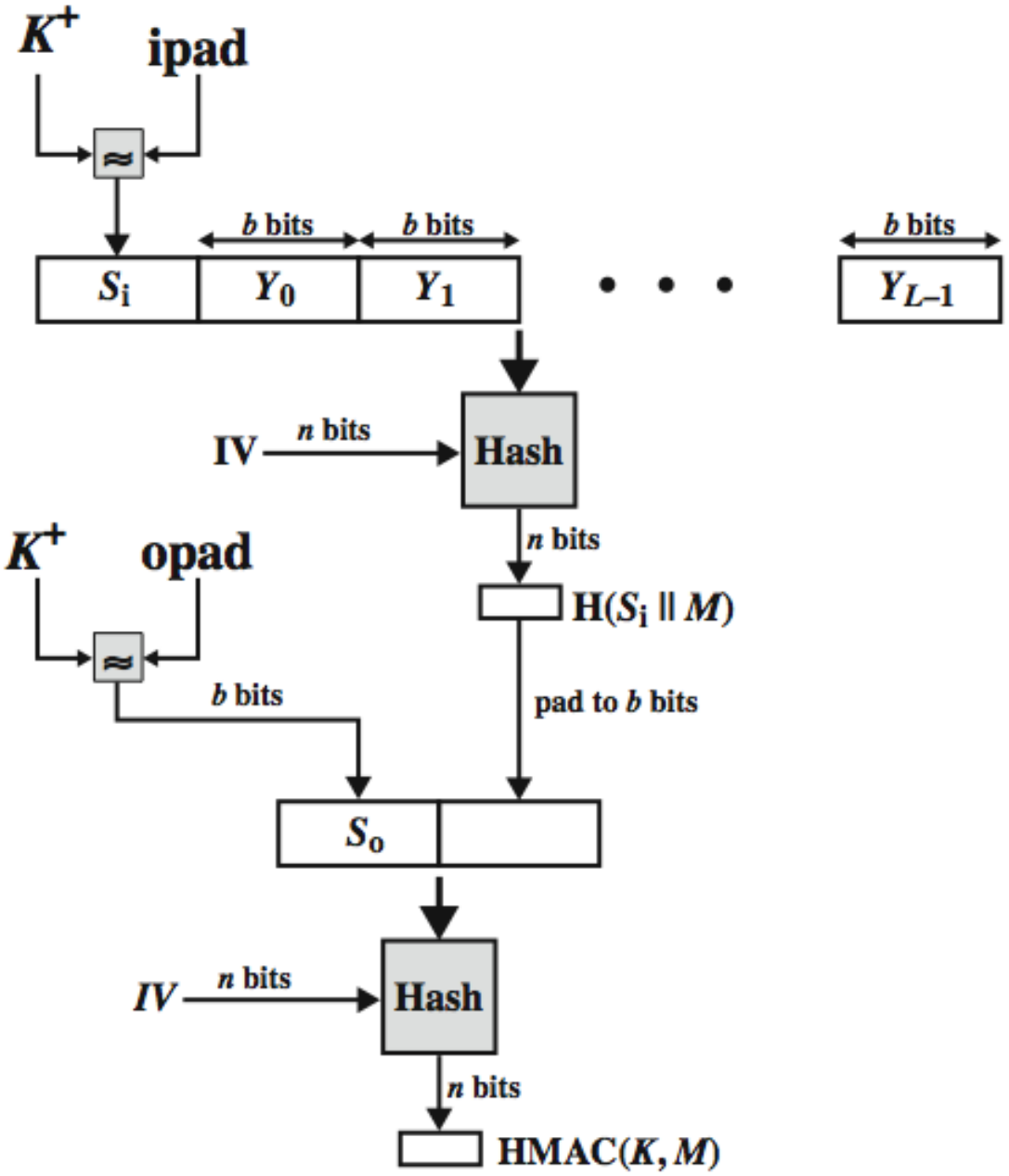
HMAC

- specified as Internet standard RFC2104
- uses hash function on the message:

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

  - where K^+ is the key padded out to block size
  - $opad$, $ipad$ are specified padding constants
- overhead is just 3 more hash block calculations than the message needs alone
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC Overview



HMAC Security

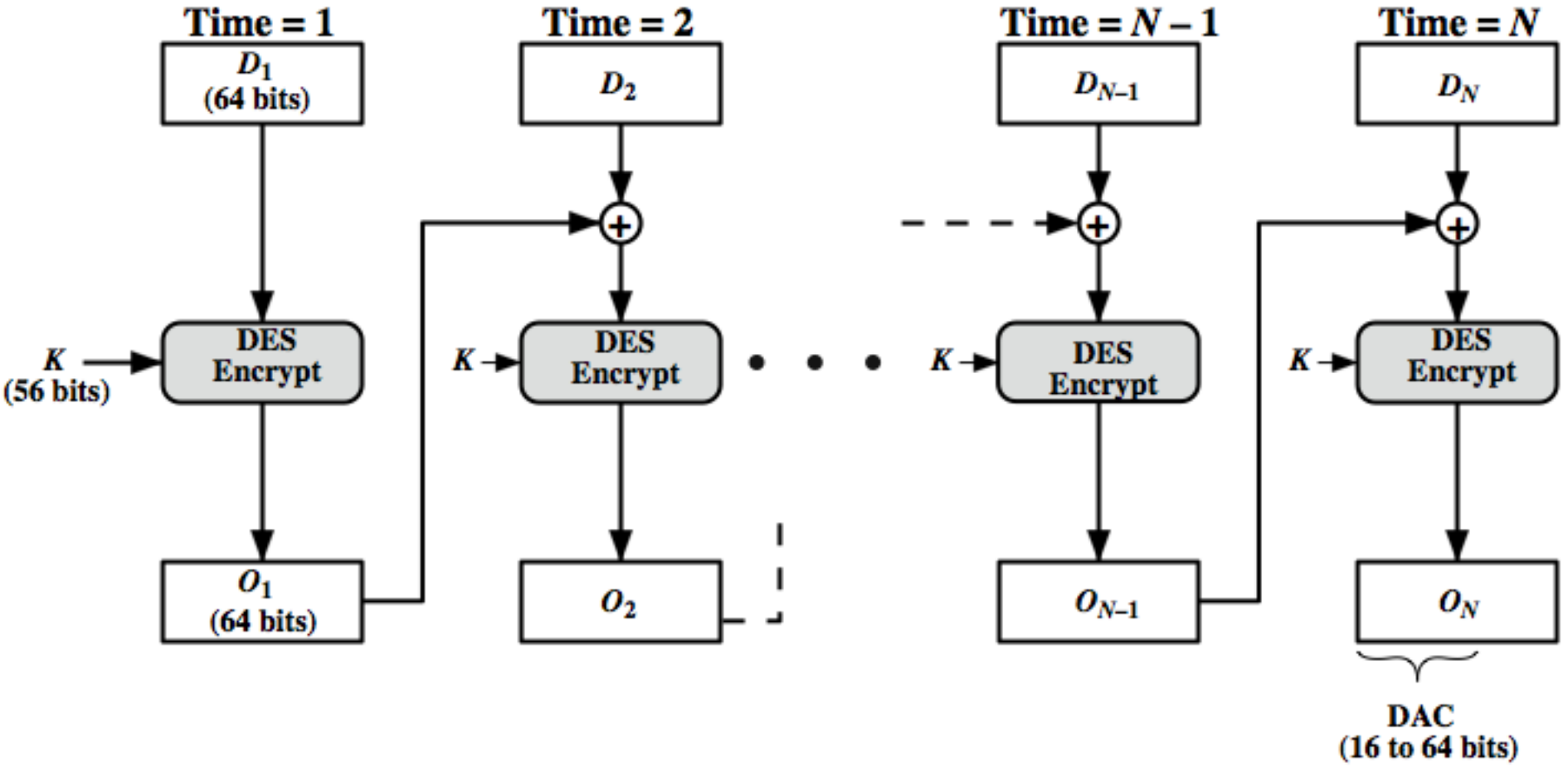
- proved security of HMAC relates to that of the underlying hash algorithm
- attacking HMAC requires either:
  - brute force attack on key used
  - birthday attack (but since keyed would need to observe a very large number of messages)
- choose hash function used based on speed verses security constraints

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…

… can use message blocks in reverse order…

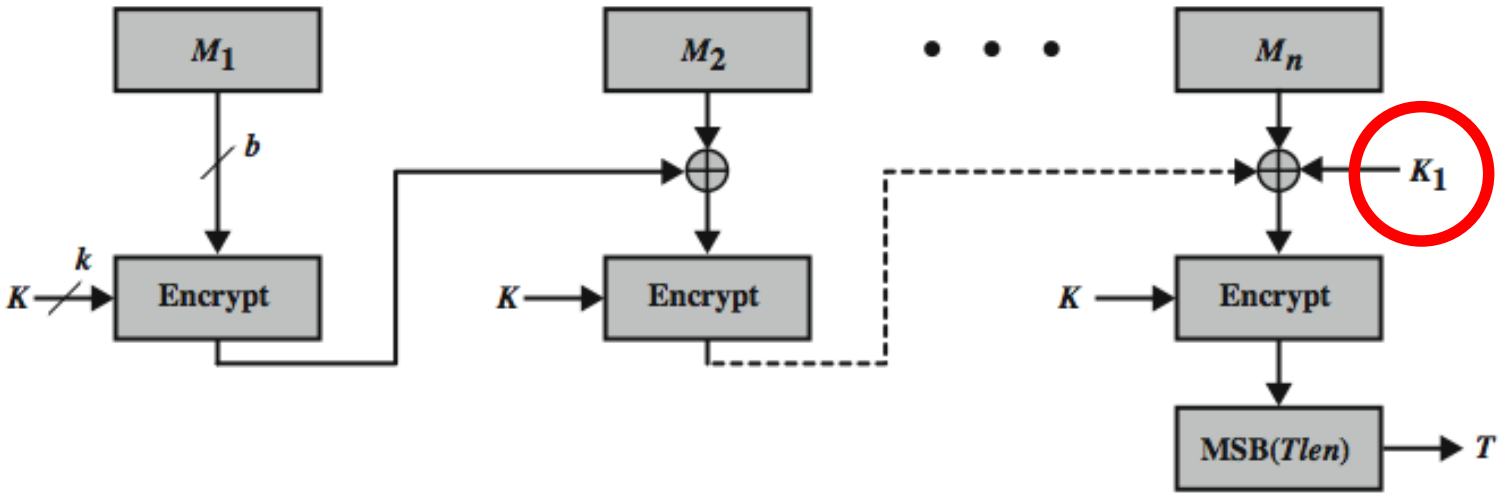
Data Authentication Algorithm



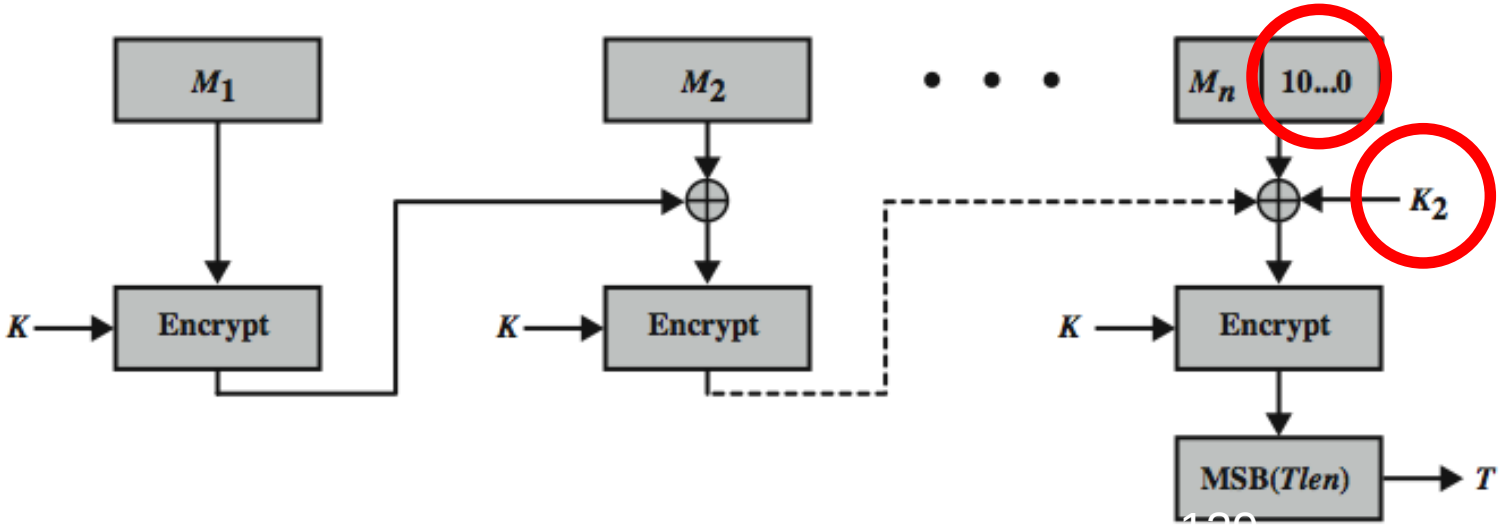
CMAC

- previously saw the DAA (CBC-MAC)
- widely used in govt & industry
- but has message size limitation
- can overcome using 2 keys & padding
- thus forming the Cipher-based Message Authentication Code (CMAC)
- adopted by NIST SP800-38B

CMAC Overview



(a) Message length is integer multiple of block size



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

Authenticated Encryption

- simultaneously protect confidentiality and authenticity of communications
  - often required but usually separate
- approaches
  - Hash-then-encrypt: E(K, (M || H(M))
  - MAC-then-encrypt: E(K2, (M || MAC(K1, M))
  - Encrypt-then-MAC: (C=E(K2, M), T=MAC(K1, C)
  - Encrypt-and-MAC: (C=E(K2, M), T=MAC(K1, M)
- decryption /verification straightforward
- but security vulnerabilities with all these

DIGITAL SIGNATURE

- Traditional signature
  - Sign
  - Verify
  - Methods of forgery
  - Chances of forgery
- Digital signature
  - Sign
  - Verify
  - Methods of forgery
  - Chances of forgery

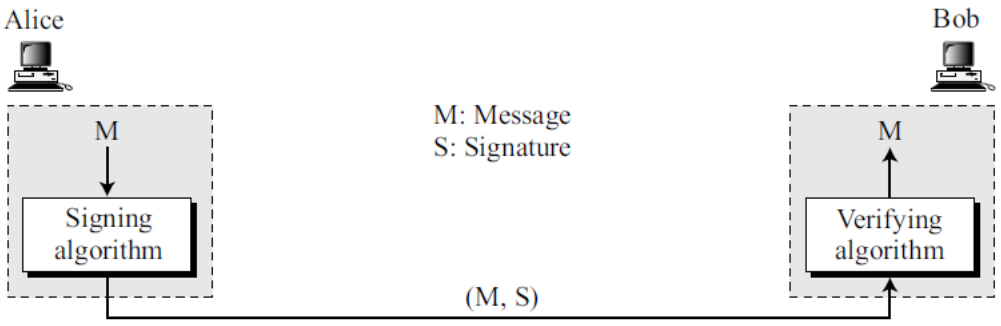
Digital signatures

- a cryptographic technique used to verify the authenticity and integrity of a digital message or document.
- Provide a way for a sender to prove that a document has not been altered in transit and that it was indeed signed by the purported sender

Digital signature process

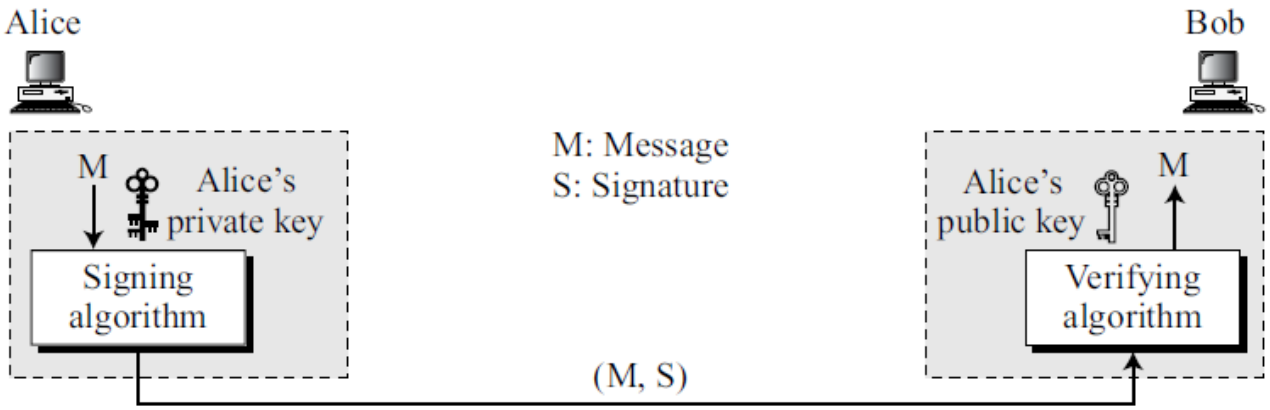
- The sender uses a signing algorithm to sign the message.
- The message and the signature are sent to the receiver.
- Receiver receives the message and signature
- Receiver applies the verifying algorithm to the combination.
- If the result is true, the message is accepted; otherwise, it is rejected.

Figure 13.1 *Digital signature process*



Sign and verify

Figure 13.2 *Adding key to the digital signature process*



Which key should be used for signing- public or private?

Keys and digital signatures

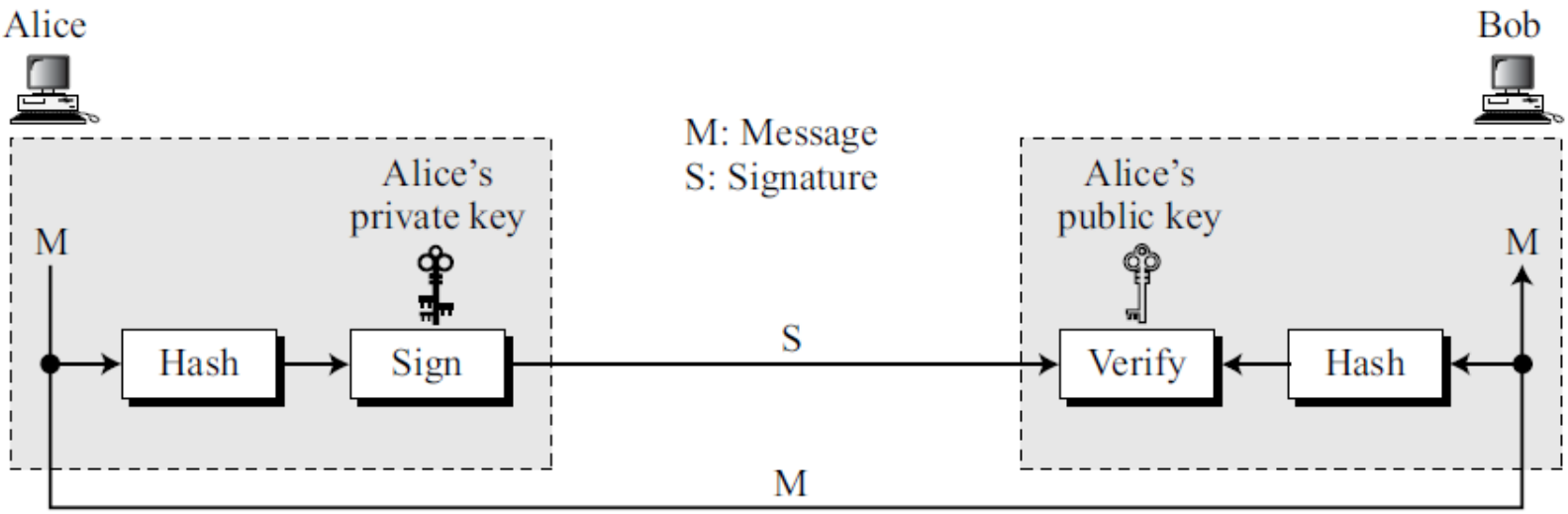
- The signer uses her private key, applied to a signing algorithm, to sign the document.
- The verifier, on the other hand, uses the public key of the signer, applied to the verifying algorithm, to verify the document.
- When a document is signed, anyone can verify it because everyone has access to sender's public key.
- Sender must not use her public key to sign the document because then anyone could forge her signature.

What if we use symmetric key cryptography?

1. The secret key is known by only two entities. So while sending document to third person, sender must use a diff key.
   - No of keys to be maintained is an issue
2. creating a secret key for a session involves authentication, which uses a digital signature, forming a vicious circle
3. No strong repudiation as key is known to more than one person

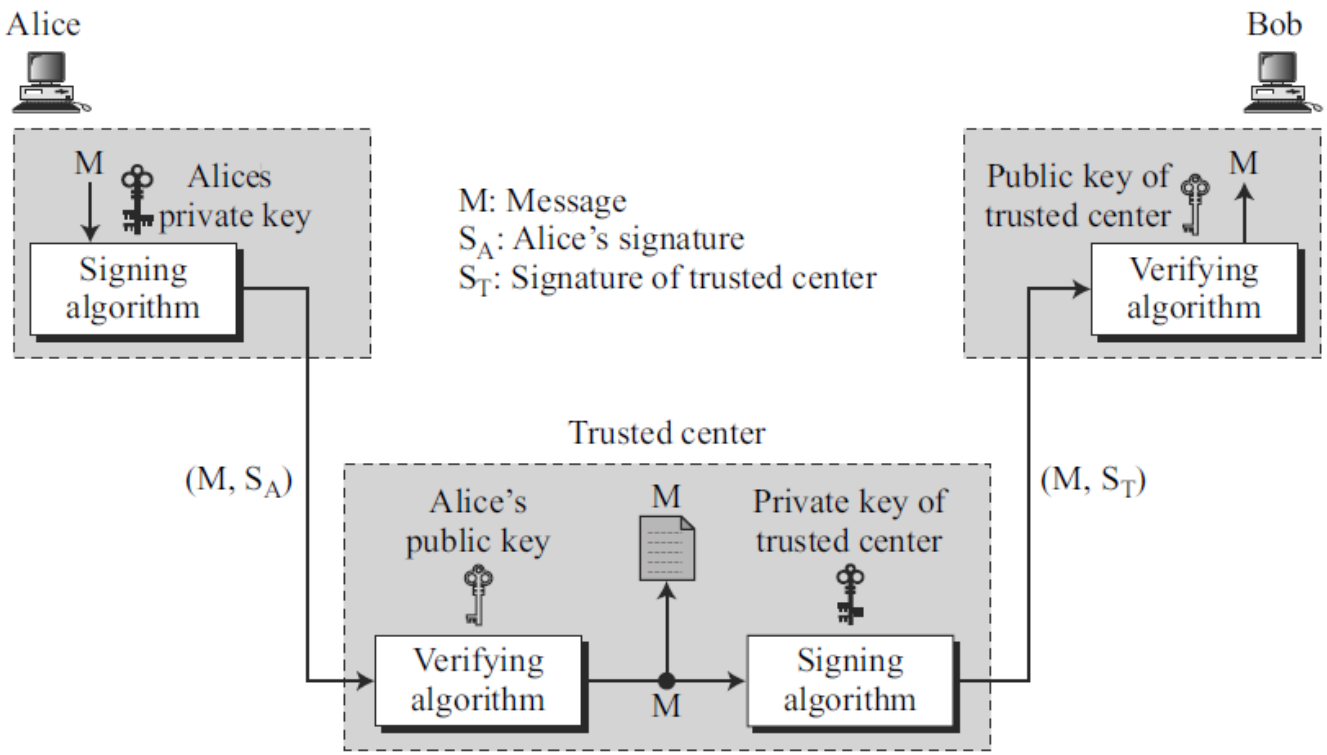
A digital signature needs a public-key system. The signer signs with her private key; the verifier verifies with the signer's public key.

Figure 13.3 *Signing the digest*



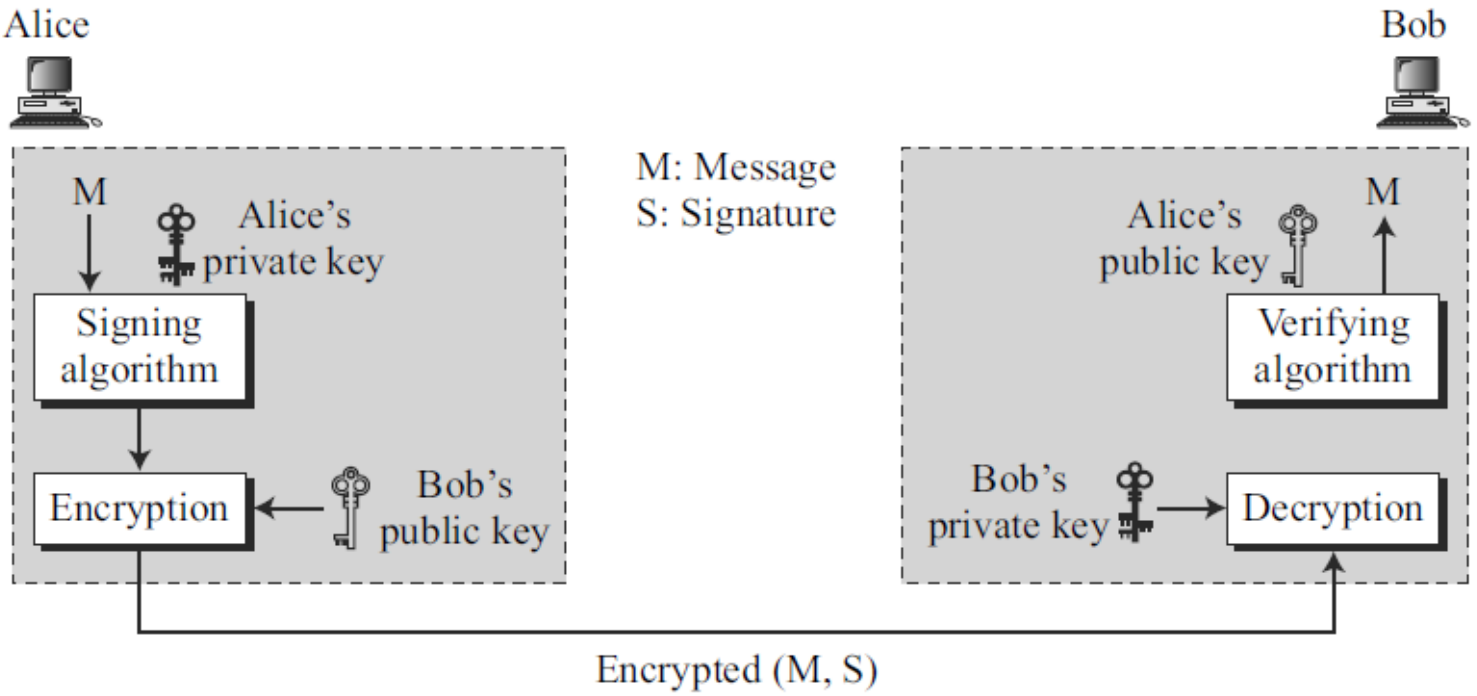
Trusted centre for nonrepudiation

Figure 13.4 *Using a trusted center for nonrepudiation*



Confidentiality and Digital Signatures

Figure 13.5 *Adding confidentiality to a digital signature scheme*



Key benefits

- **Authentication**:
  - confirm the identity of the sender.
  - They are typically generated using the sender's private key, and can only be verified using the corresponding public key. This ensures that the sender is who they claim to be.
- **Data Integrity**:
  - Even a minor alteration to the signed document will result in an invalid signature.
- **Non-Repudiation**:
  - Once a digital signature is applied, the sender cannot deny their involvement in sending the document.
  - Especially important in legal and business contexts.

Comparison

	Conventional Signature	Digital Signature
(1) Inclusion	Included in the document as part of the document.	Send the signature as a separate document.
(2) Verification Method	Recipient compares the signature on the document with the signature on file.	The recipient receives the message and the signature. The recipient needs to apply a verification technique to the combination of the message and the signature to verify the authenticity.
(3) Relationship	Normally a one-to-many relationship between a signature and documents.	•One-to-one relationship between a signature and a message.
Duplicity	A copy of the signed document can be distinguished from the original one on file.	No such distinction unless there is a factor of time on the document

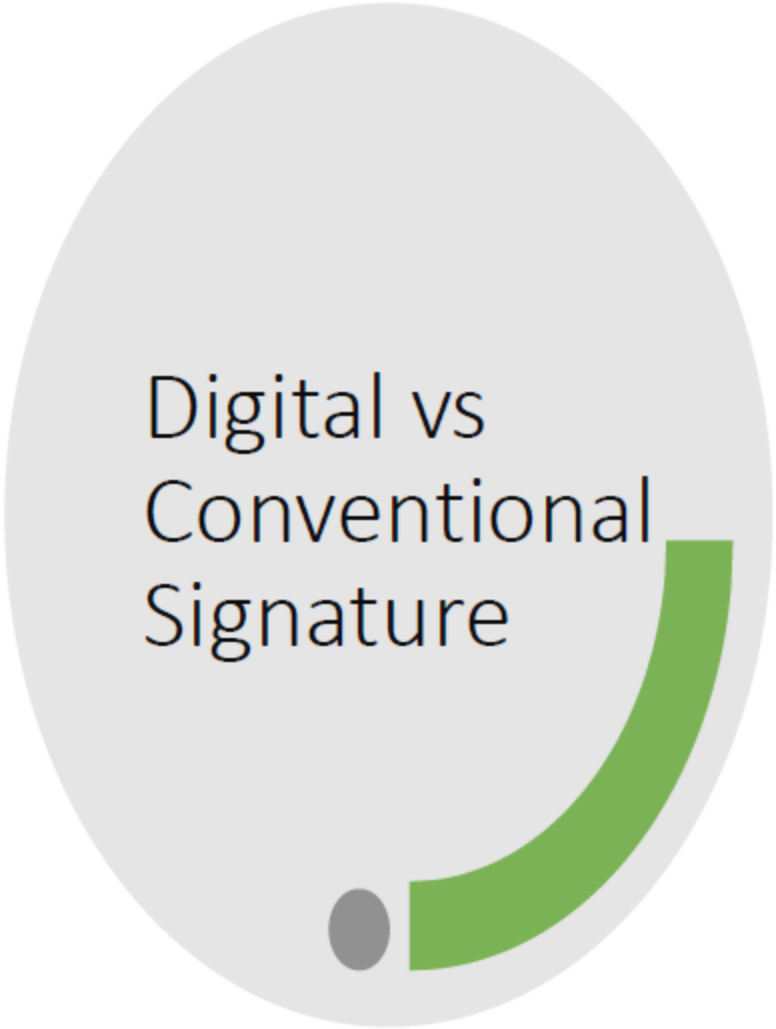
Image courtesy: https://slideplayer.com/slide/17591771/

Conventional –

- Traditional method of document signing (Handwritten, seal etc.)
- Physical part of document
- Verified by comparing it to authentic signatures
- Same sign on various docs – (one to many)

Digital –

- E-signature based on public key cryptography
- Authenticating digital documents or message
- Issued by Certificate Authority (CA)
- Schemes – RSA, Elgamal
- Verified by verification algorithm
- Different sign for different documents(one to one)



Cryptographic services and digital certificates

- Confidentiality,
- Message authentication,
- Message integrity
- Nonrepudiation.

What Digital signatures cant achieve?

- A digital signature does not provide privacy.
- If there is a need for privacy, another layer of encryption/decryption must be applied.

Applications

- Certificates and certification authorities have been developed
- Following Protocols use services of CA
  - IPSec
  - SSL/TLS
  - S/MIME
  - PGP

Digital Signature vs Digital Certificate

- Digital signatures and digital certificates are related but serve different purposes in the context of digital security and authentication.

	Digital signature	Digital certificate
Purpose	A cryptographic technique that provides a way to verify the authenticity and integrity of a digital message or document.	A data file that associates a user's identity with their public key, enabling others to verify their identity. It is issued by a trusted Certificate Authority (CA) and serves as a digital ID card
Components	composed of two parts: the document or message itself and the attached digital signature.	contains information about the certificate holder, their public key, and the digital signature of the CA. It also includes the validity period of the certificate.
Process	created by the sender using their private key to encrypt a hash of the message or document. The recipient can verify the signature using the sender's public key.	Involves registration authority, certification authority, purpose of usage, validity of period, keys, hash algorithms used etc. Complicated process compared to Digital signatures
Usage	used to authenticate the sender, provide data integrity, and ensure non-repudiation of digital messages and documents.	primarily used for public key infrastructure (PKI) applications, such as secure websites (HTTPS), email encryption and authentication, and digital signatures. They are a crucial part of authenticating users and securing online communications.
Issuer	Created by the sender of a specific message or document.	Issued by a trusted Certificate Authority (CA)
Application	Applied to individual messages or documents.	Used to secure communication channels, websites, and validate digital signatures.
Example	person signs an email with a digital signature to prove that the message is indeed from them and has not been altered in transit.	when one visits a secure website (e.g., an online banking website), the web browser checks the website's digital certificate to ensure that it is legitimate and not a malicious imposter.

Digital Signature Scheme

Digital Signature Standard (DSS)

- NIST has published Federal Information Processing Standard FIPS186, known as DSS.
- Makes use of the Secure Hash Algorithm(SHA)
- Originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme.

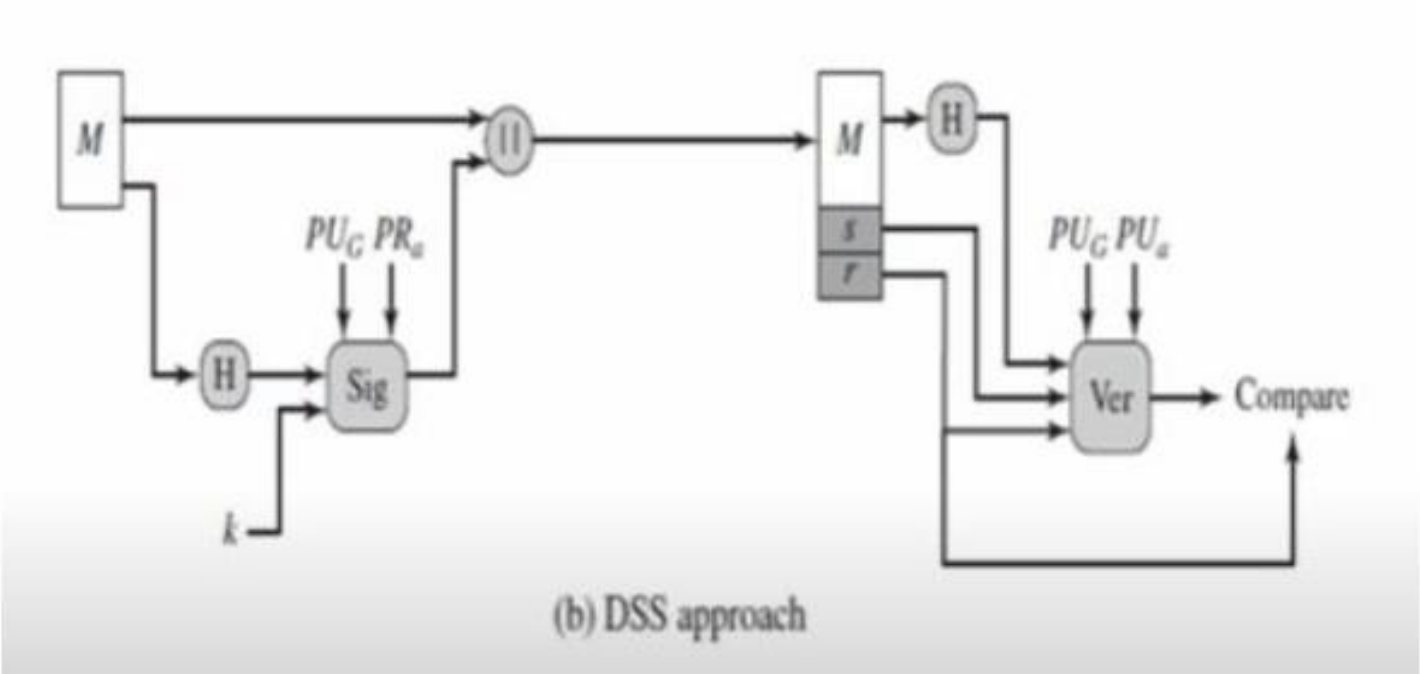
DSS

- A digital signature scheme typically consists of three algorithms:
  - A key generation algorithm that selects a private key uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding public key.
  - A signing algorithm that, given a message and a private key, produces a signature.
  - A signature verifying algorithm that, given the message, public key and signature, either accepts or rejects the message's claim to authenticity.

- Formally, a **digital signature scheme** is a triple of probabilistic polynomial time algorithms, (G, S, V), satisfying:
  - G (key-generator) generates a public key (pk), and a corresponding private key (sk), on input 1^n, where n is the security parameter.
  - S (signing) returns a tag, t, on the inputs: the private key (sk), and a string (x).
  - V (verifying) outputs *accepted* or *rejected* on the inputs: the public key (pk), a string (x), and a tag (t).
- 1^n refers to a unary number.

DSS steps

- Generation of Public and Private key for User A
- Creation of Digital Signature by User A for message M
- User B verifies the Digital Signature

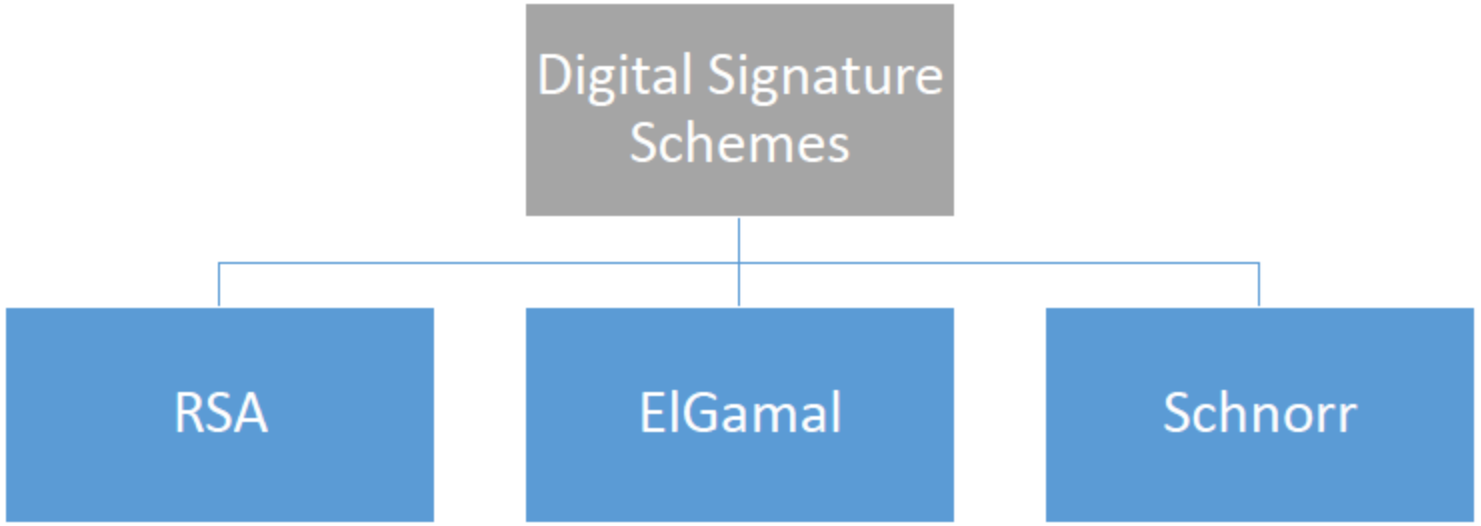


(b) DSS approach

Properties of DSS

1. Authenticity of a signature generated from a fixed message and fixed private key can be verified by using the corresponding public key.
2. Should be computationally infeasible to generate a valid signature without knowing that party's private key.

A digital signature is an authentication mechanism that enables the creator of the message to attach a code that acts as a signature.

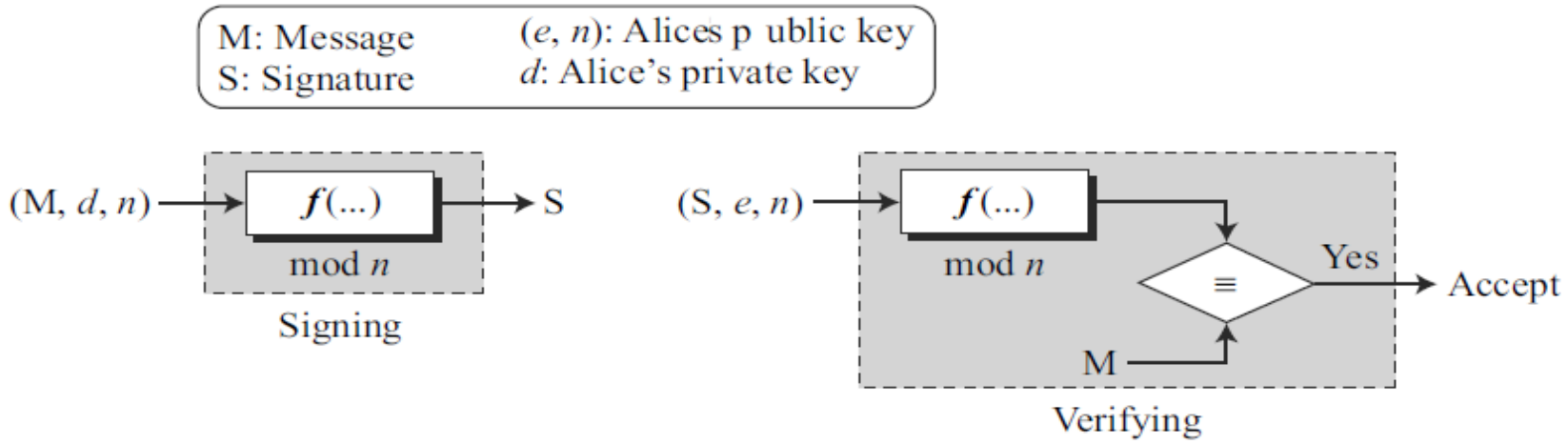

Digital Signature Schemes
RSA
ElGamal
Schnorr

RSA Digital Signature Scheme

- The digital signature scheme changes the roles of the private and public keys.
- Sender uses her own private key to sign the document; the receiver uses the sender's public key to verify it.
- i.e. private key plays the role of the sender's own signature,
- Sender's public key plays the role of the copy of the signature that is available to the public.

RSA digital signature scheme

Figure 13.6 *General idea behind the RSA digital signature scheme*



- The signing and verifying sites use the same function, but with different parameters.
- The verifier compares the message and the output of the function for congruence.
- If the result is true, the message is accepted.

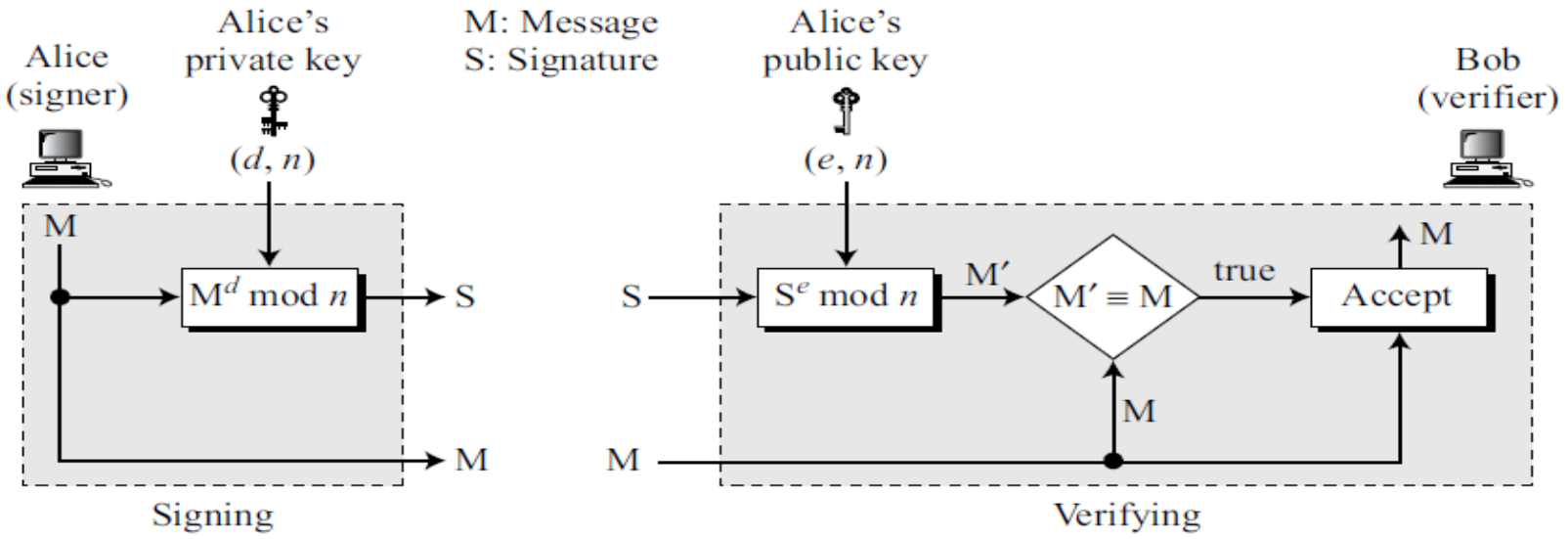
RSA digital signature scheme

Steps:

1. Key generation
2. Sign
3. Verify

1. Key generation
   i. Sender chooses p & q,
   ii. computes n=p*p,
   iii. Computes $\varphi(n) = (p - 1) * (q - 1)$.
   iv. Chooses e, and computes d such that $e*d = 1 \bmod \varphi(n)$
   v. Keeps d, publicly announces {e,n} pair

Figure 13.7 *RSA digital signature scheme*

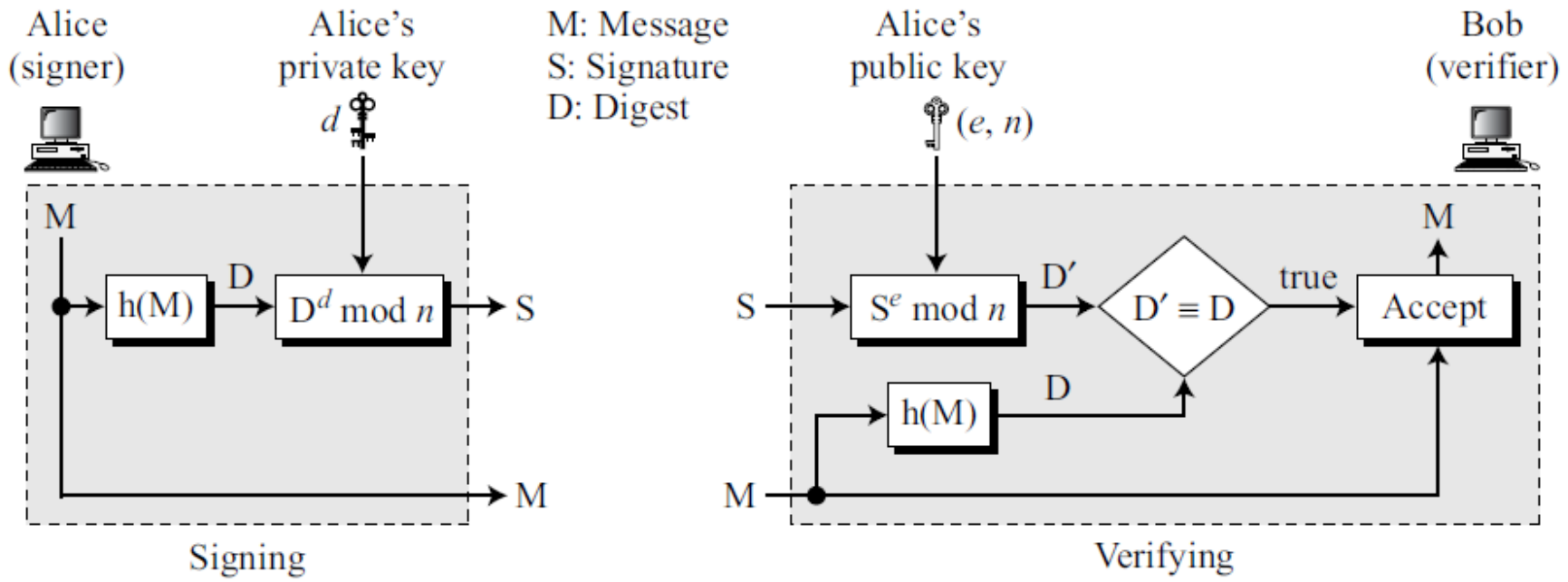


2. Sign and verify

- Sign $S = M^d \bmod n$ and sends the message and the signature to Bob.
- Verify to create a copy of the message $M' = S^e \bmod n$.
- $M' \equiv M \pmod n \rightarrow S^e \equiv M \pmod n \rightarrow M^{d \times e} \equiv M \pmod n$

RSA signature on message digest

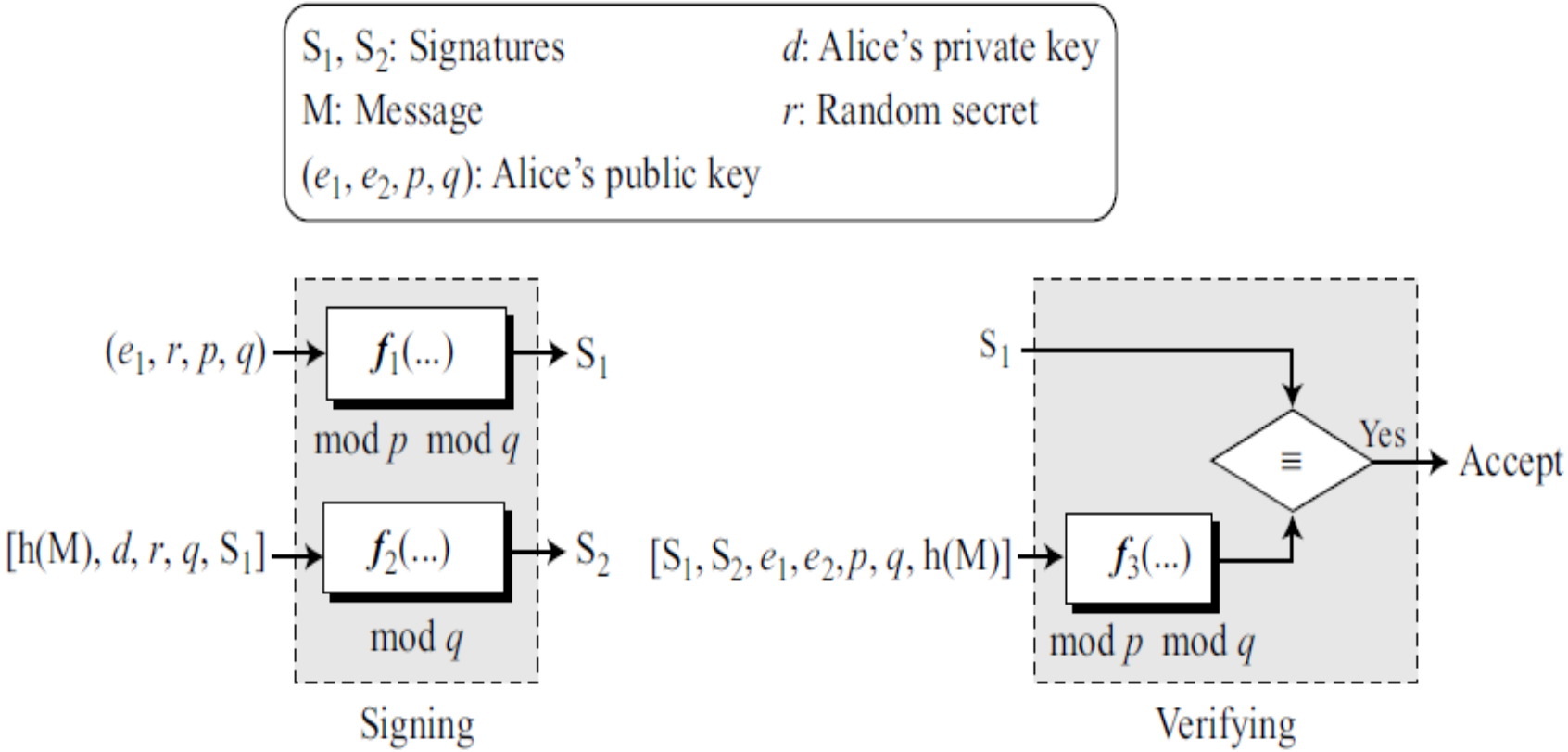
Figure 13.8 *The RSA signature on the message digest*



Digital Signature Standard (DSS)

- DSS was adopted by the National Institute of Standards and Technology (NIST) in 1994.
- NIST published DSS as FIPS 186.
- DSS uses a digital signature algorithm (DSA) based on the ElGamal scheme with some ideas from the Schnorr scheme.
- DSS has been criticized from the time it was published.
- The main complaint regards the secrecy of DSS design.
- The second complaint regards the size of the prime, 512 bits.
- Later NIST made the size variable to respond to this complaint.

DSS scheme general idea



- In the signing process, two functions create two signatures;
- in the verifying process, the output of one function is compared to the first signature for verification.
- This is similar to Schnorr, but the inputs are different.
- Another difference is that this scheme uses the message digest (not the message) as part of inputs to functions 1 and 3.
- The interesting point is that the scheme uses two public moduli: p and q.
- Functions 1 and 3 use both p and q; function 2 uses only q.

Key Generation in DSS

1. Alice chooses a prime p, between 512 and 1024 bits in length.

→No of bits in p must be a multiple of 64.

2. q is 160-bit prime in such a way that q divides (p – 1).
3. Alice uses two multiplication groups $<Z_p^*, \times>$ and $<Z_q^*, \times>$; the second is a subgroup of the first.
4. Alice creates e_1 to be the q^{th} root of 1 modulo p. i.e.

$$e_1^q = 1 \bmod p.$$

5. To do so, Alice chooses e_0 which is a primitive element in Z_p, and calculates $e_1 = e_0^{(p-1)/q} \bmod p$.
5. Alice chooses d as the private key and calculates

$$e_2 = e_1^d \bmod p.$$

7. Alice's public key is (e_1, e_2, p, q); her private key is (d).

Signing in DSS

1. Alice chooses a random number r ($1 \le r \le q$).

Note: public and private keys can be chosen once and used to sign many messages, Alice needs to select a new r each time she needs to sign a new message.

2. Alice calculates the first signature $S_1 = (e_1^r \bmod p) \bmod q$.

Note that the value of the first signature S_1 does not depend on M, the message.

3. Alice creates a digest of message h(M).
4. Alice calculates the second signature

$$S_2 = [(h(M) + d\ S1)*r^{-1}] \bmod q.$$

Note that the calculation of S_2 is done in modulo q arithmetic.

5. Alice sends M, S_1, and S_2 to Bob.

Verifying in DSS

1. Bob checks to see if $0 < S_1 < q$.
2. Bob checks to see if $0 < S_2 < q$.
3. Bob calculates a digest of M using the same hash algorithm used by Alice.
4. Bob calculates

$$; V = [(e_1^{h(M)S_2^{-1}} e_2^{S_1 S_2^{-1}}) \bmod p] \bmod q.$$

5. If S_1 is congruent to V, the message is accepted; otherwise, it is rejected.

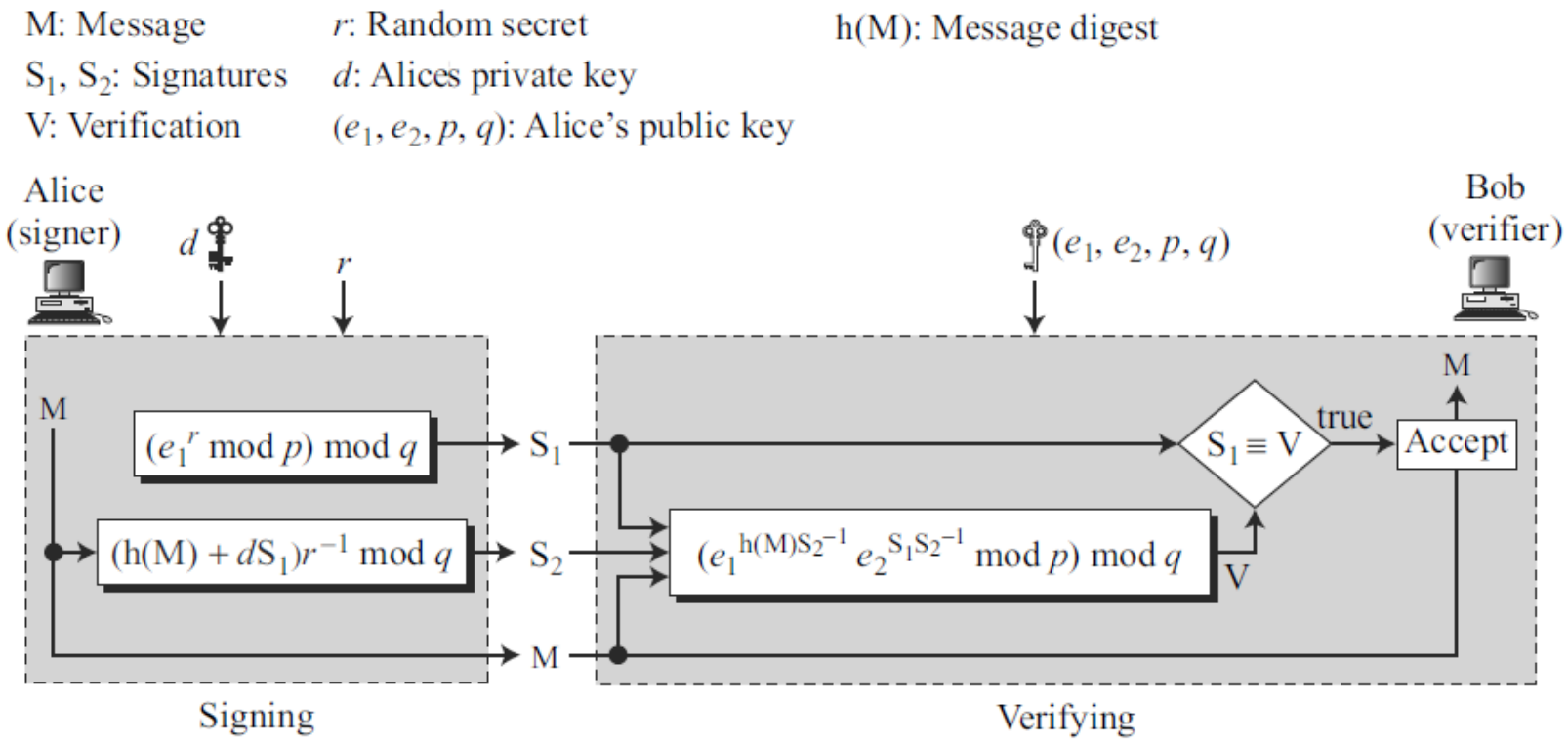
- DSS Versus RSA

Computation of DSS signatures is faster than computation of RSA signatures when using the same p.

- DSS Versus ElGamal

DSS signatures are smaller than ElGamal signatures because q is smaller than p.

Sign and Verify in DSS



13.6.1 Variations and applications of DSS

Time Stamped Signatures

Sometimes a signed document needs to be time stamped to prevent it from being replayed by an adversary. This is called time-stamped digital signature scheme.

Blind Signatures

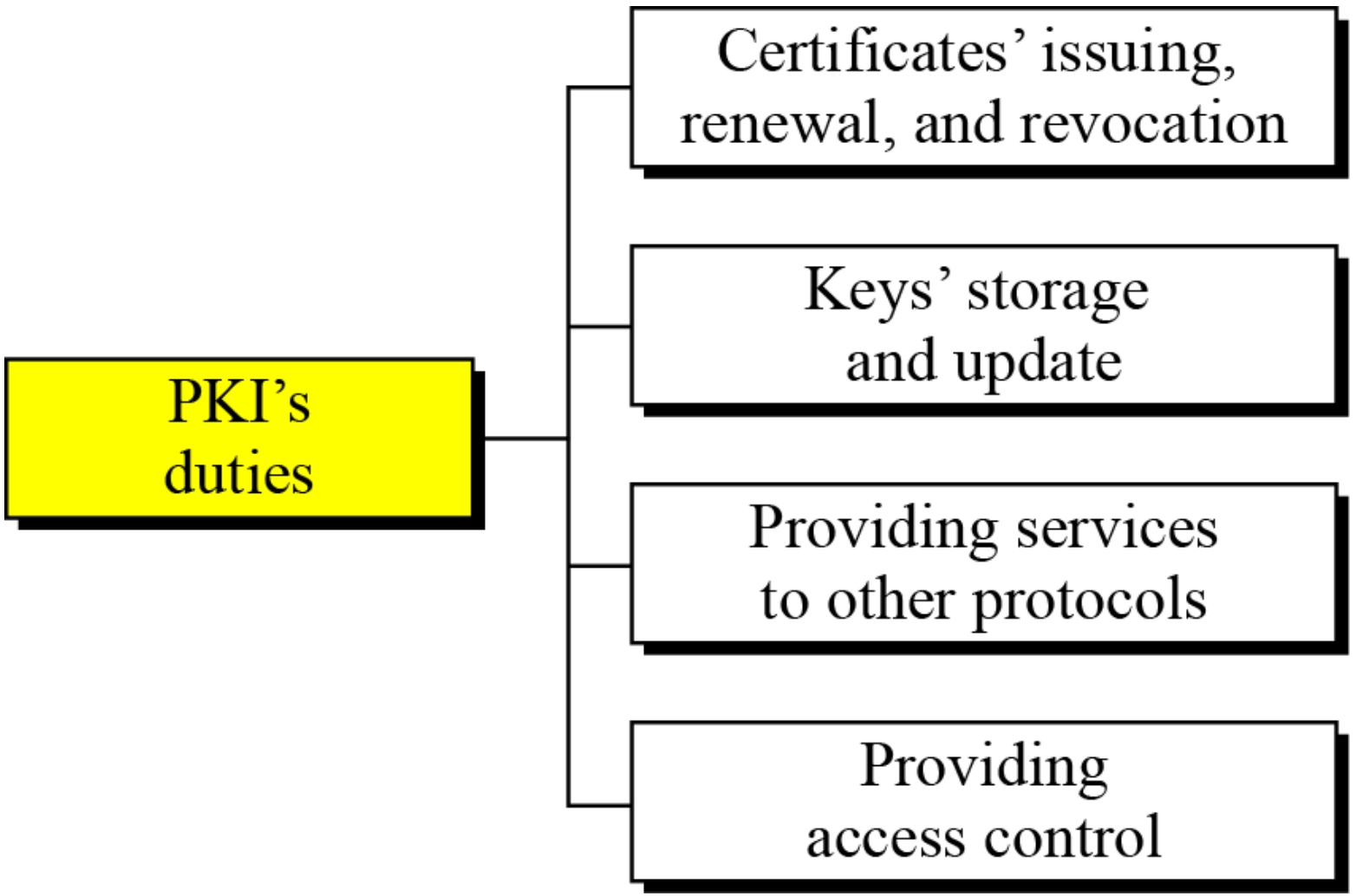
Sometimes we have a document that we want to get signed without revealing the contents of the document to the signer.

Undeniable Digital Signatures

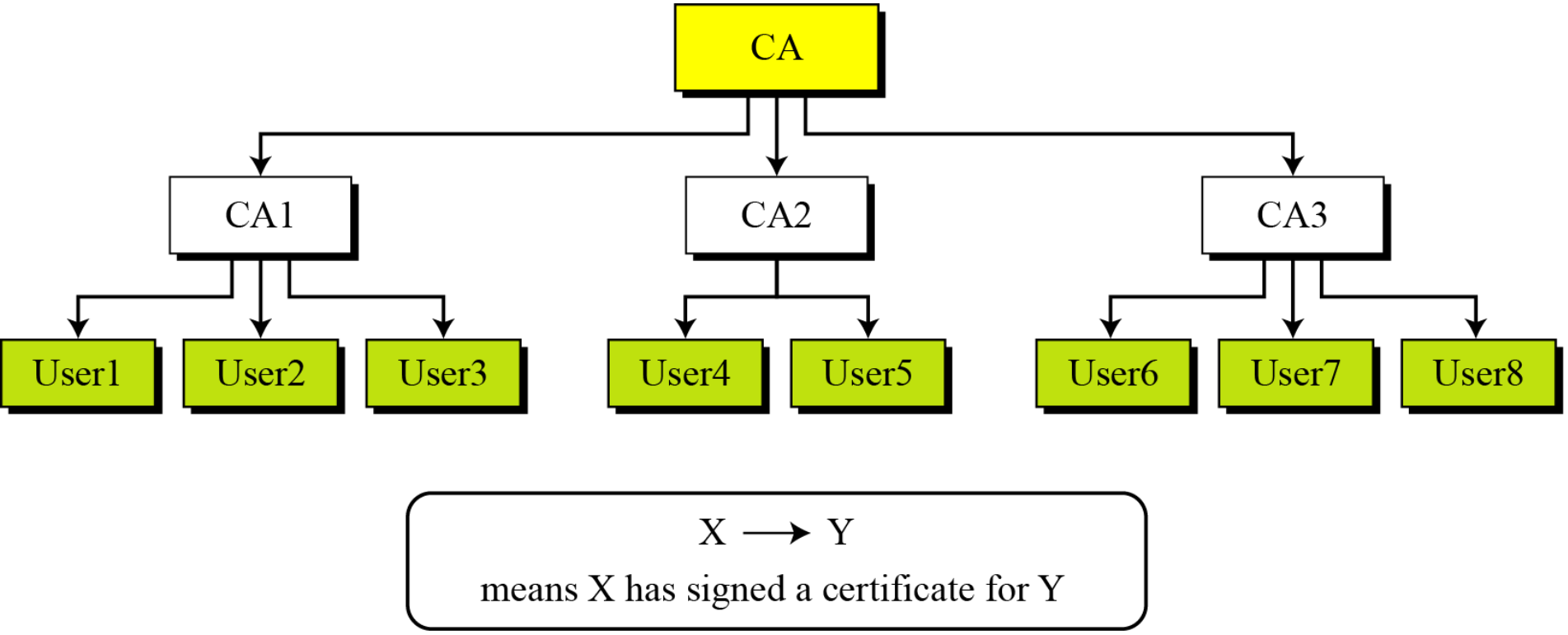
- Invented by Chaum and Van antwerpen
- Has three components –
  - A signing algorithm
  - A verification protocol
  - A disavowal(non-redudiation) protocol
- Signing algo allows Alice to sign a message
- Verification protocol uses challenge response mechanism & involves Alice in verification; prevents message duplication and distribution without Alice's approval
- Disavowal helps Alice to deny a forged signature

Reading slides: Public Key Infrastructures

PKI duties


PKI's duties
Certificates' issuing, renewal, and revocation
Keys' storage and update
Providing services to other protocols
Providing access control

PKI hierarchical trust model



- Show how User1, knowing only the public key of the CA (the root), can obtain a verified copy of User3's public key.

Solution

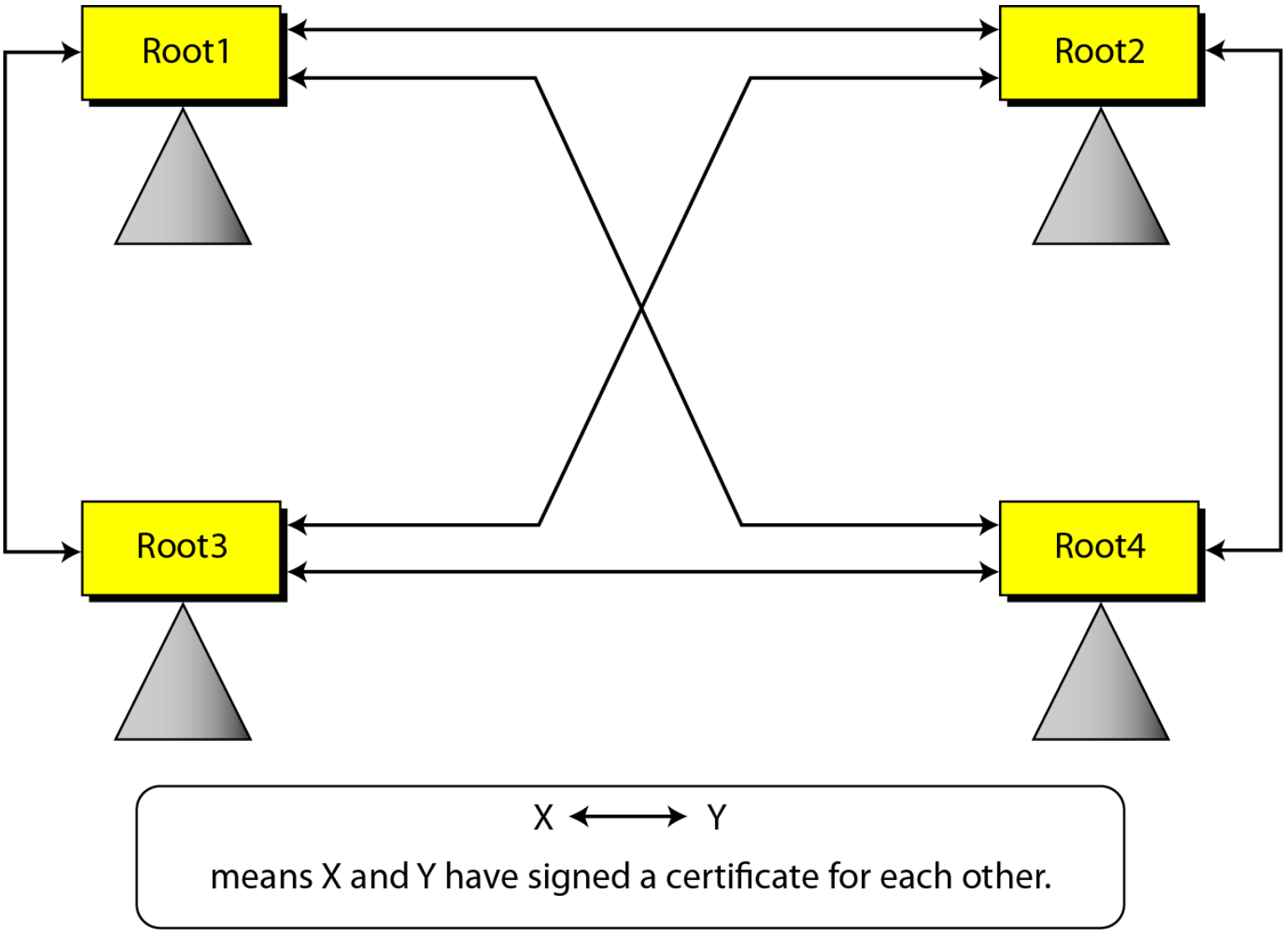
User3 sends a chain of certificates, CA<<CA1>> and CA1<<User3>>, to User1.

a. User1 validates CA<<CA1>> using the public key of CA.
b. User1 extracts the public key of CA1 from CA<<CA1>>.
c. User1 validates CA1<<User3>> using the public key of CA1.
d. User1 extracts the public key of User 3 from CA1<<User3>>.

Example.. contd

- Some Web browsers, such as Netscape and Internet Explorer, include a set of certificates from independent roots without a single, high-level, authority to certify each root. One can find the
- list of these roots in the Internet Explorer at Tools/Internet Options/Contents/Certificate/Trusted roots (using pull-down menu). The user then can choose any of this root and view the certificate.

PKI Mesh Model



Example 2

- Alice is under the authority Root1; Bob is under the authority Root4. Show how Alice can obtain Bob's verified public key.

Solution

- Bob sends a chain of certificates from Root4 to Bob. Alice looks at the directory of Root1 to find Root1<<Root1>> and
- Root1<< Root4>> certificates. Using the process shown in Figure 15.21, Alice can verify Bob's public key.

x.509 authentication service

X.509

- Definition: X.509 is a standard defining the format of public key certificates.
- X. 509 certificates are used in many Internet protocols, including TLS/SSL, which is the basis for HTTPS, the secure protocol for browsing the web.
- Importance: Essential for secure communication and authentication in various applications.
- History: Developed in the 1980s as part of the ITU-T X series.

Digital Certificates

- Digital certificate is issued by a trusted third party which proves sender's identity to the receiver and receiver's identity to the sender.

- A digital certificate is a certificate issued by a Certificate Authority (CA) to verify the identity of the certificate holder.
- The CA issues an encrypted digital certificate containing the applicant's public key and a variety of other identification information.
- Digital certificate is used to attach public key with a particular individual or an entity.

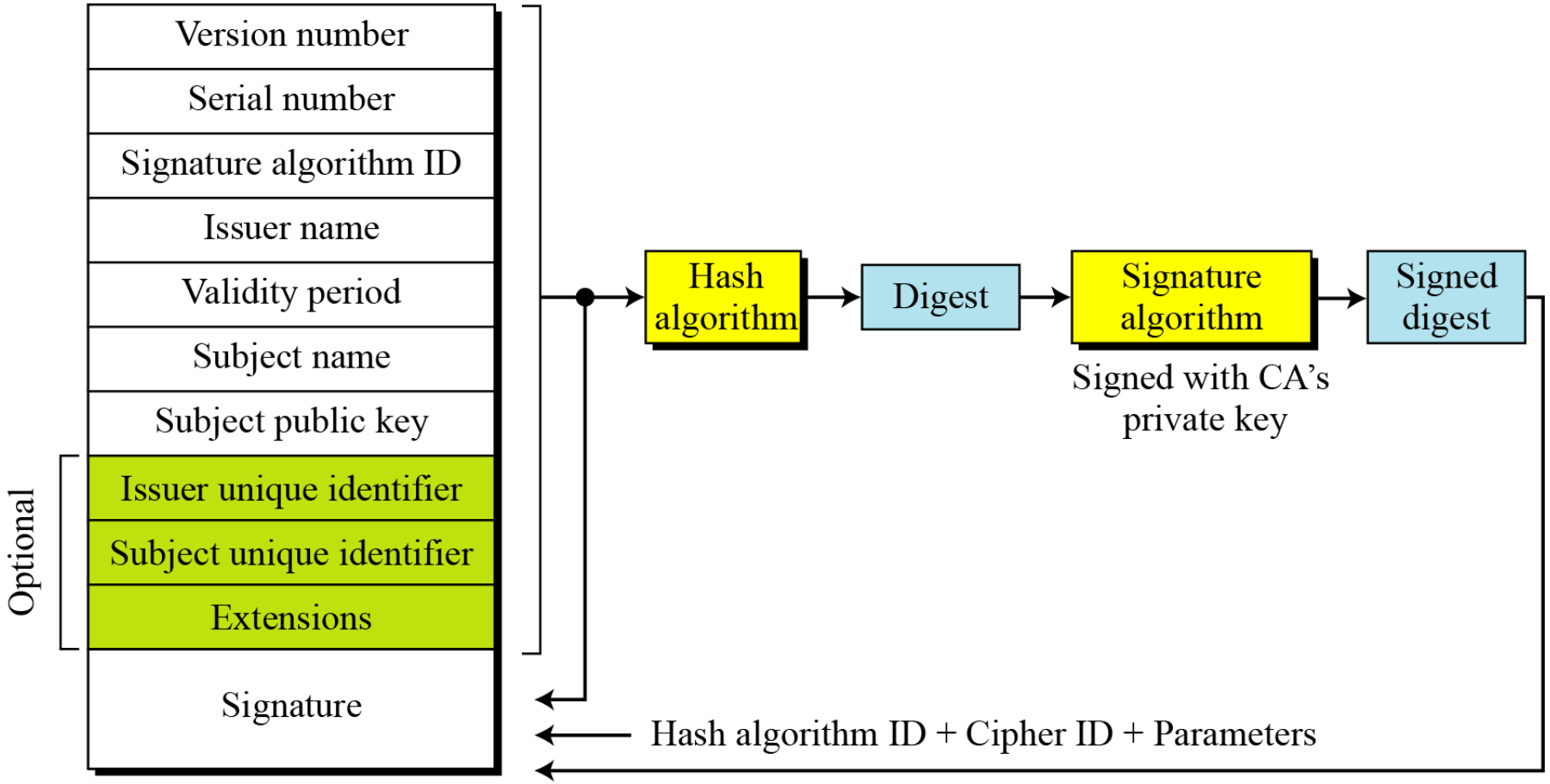
X.509 Certificates

- Definition: Digital documents used to verify the authenticity of an entity.
- Components: Public key, identity information, digital signature.
- Types: End-entity certificates and CA certificates.

Key Components of X.509

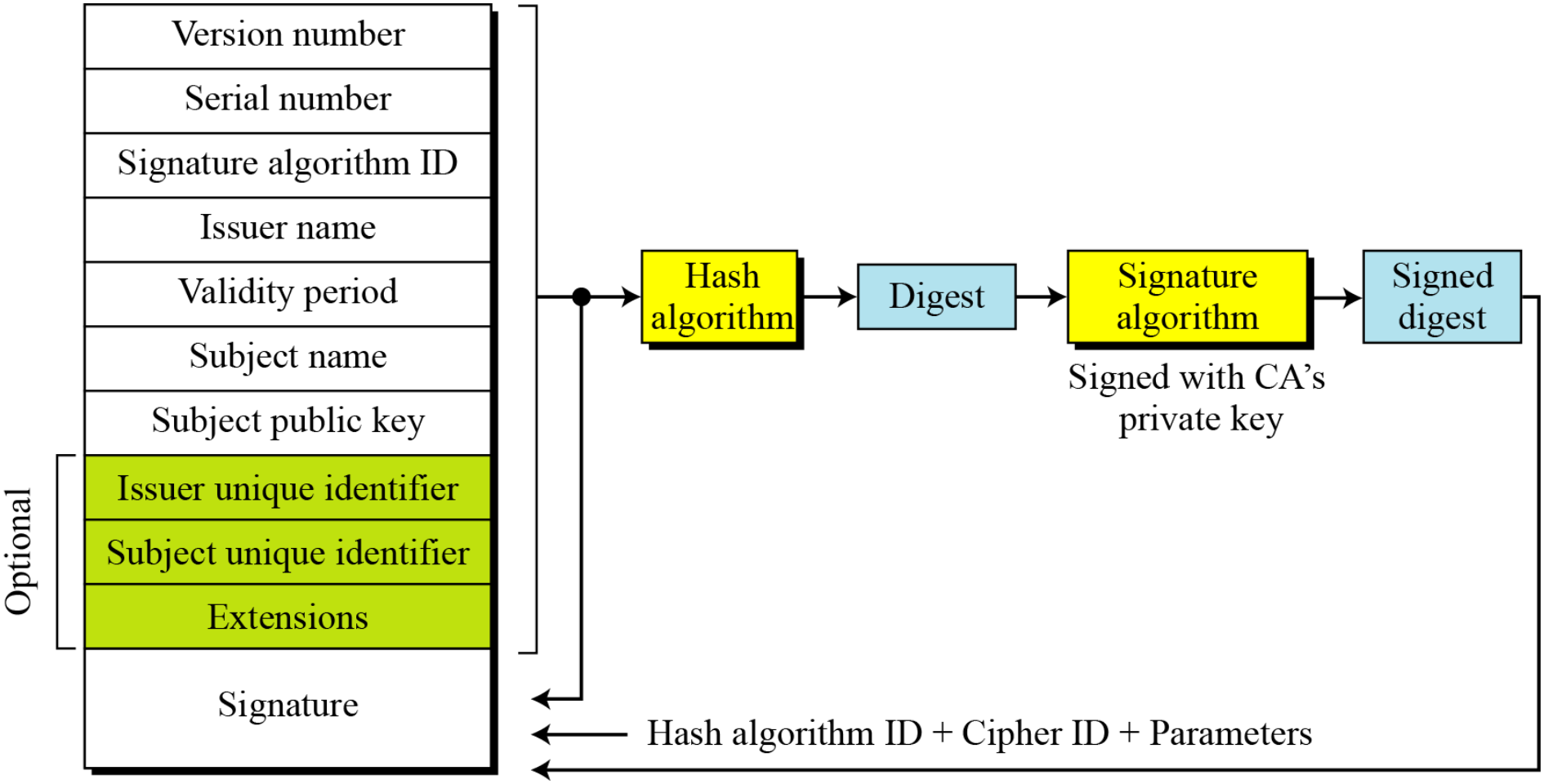
- X.509 Infrastructure
  - A fundamental framework for establishing trust in digital communication.
  - Key Role: Ensures the authenticity, integrity, and confidentiality of data.
- Public Key Infrastructure (PKI)
  - Definition: A subset of X.509, a PKI is a set of hardware, software, policies, and standards that work together to provide a framework for secure communications.
  - Components: Certificates, CAs, registration authorities, and directories.
- Digital Certificates
  - Role: Digital documents that bind a public key to an entity's identity.Structure: Includes public key, subject, issuer, validity period, and digital signature
  - .Examples: SSL/TLS certificates, email certificates.
    Certification Authorities
- Certification Authorities (CAs)
  - Role: Trusted entities that issue and verify certificates.
  - Types: Root CAs, Intermediate CAs, and End-entity CAs.
  - Hierarchy: Root CA signs the certificates of Intermediate CAs, forming a chain of trust.
- Certificate Revocation Lists
  - Definition: Lists of revoked digital certificates issued by a Certificate Authority (CA).
  - Purpose: Ensure the security of digital communication by identifying compromised or no longer valid certificates.

X.509 certificate format



- Certificate Renewal
  - Each certificate has a period of validity. If there is no problem with the certificate, the CA issues a new certificate before the old one expires.
- Certificate Renewal
  - In some cases a certificate must be revoked before its expiration.
- Delta Revocation
  - To make revocation more efficient, the delta certificate revocation list (delta CRL) has been introduced.

Certificate revocation list format



Reasons of certificate revocation

- User's key is compromised
- CA doesn't wish to certify user for some reason(User's certificate relates to the organization that she no longer works for.)
- CA's private key is compromised

How CRLs Work?

- Issuance: CRLs are periodically issued by the CA.
- Content: They contain the serial numbers of revoked certificates, a time of issue, and a next update time.
- Distribution: CRLs are made publicly available and distributed to relying parties.
-

Why CRLs Are Important?

- Assurance of Certificate Validity: CRLs help users and systems avoid relying on compromised or invalid certificates.
- Trustworthiness: The presence of a CRL adds an additional layer of trust to the X.509 certificate infrastructure.
- Regulatory Compliance: Many industries and standards require the use of CRLs for compliance.

Trust in the X.509 Infrastructure

- Trust Model: Trust is established through the trusted root CA certificate.
- Certificate Chain: Building and verifying certificate chains.
- Revocation: Handling revoked certificates through CRLs and OCSP.

Applications of X.509 Infrastructure

- SSL/TLS: Securing web traffic.
- S/MIME: Secure email communication.
- Code Signing: Authenticating software and updates.
- VPNs: Establishing secure connections.
-

Challenges and Considerations in X.509

- Security: Protecting private keys and mitigating vulnerabilities.
- Trust: Ensuring CAs are reliable and secure.
- Scalability: Handling a growing number of certificates.
- Regulatory Compliance: Meeting legal and industry-specific requirements.

Future of X.509 Infrastructure

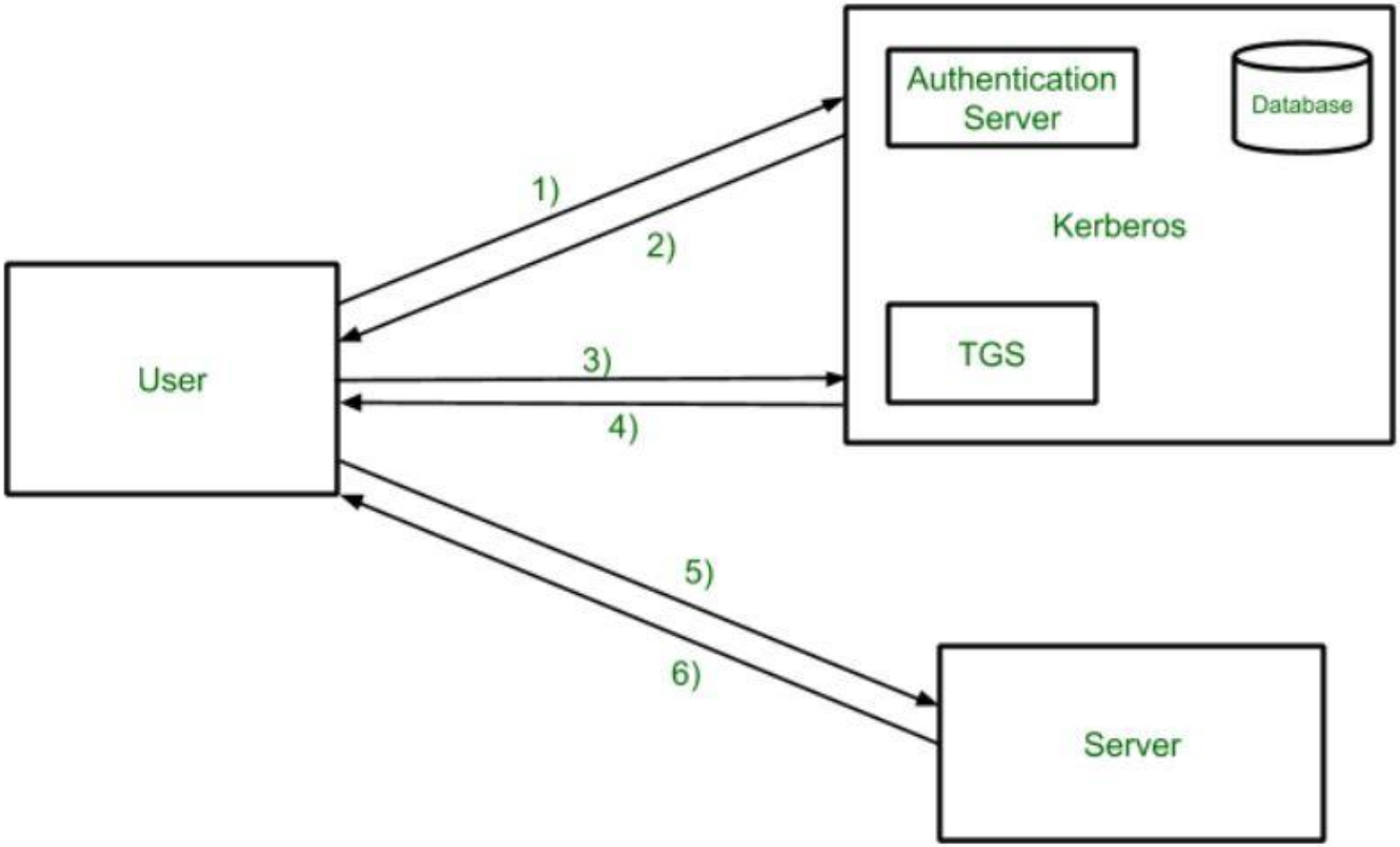
- Quantum Computing: Impact on cryptographic algorithms.
- Post-Quantum Cryptography: Preparing for quantum-resistant certificates.
- Advances in PKI: Improving certificate management and trust models.


KERBEROS

Kerberos

- In mythology, Kerberos(also known as Cerberus) is a large, three-headed dog that guards the gates to the underworld to keep souls from escaping.
- Kerberos is the computer network authentication protocol initially developed in the 1980s by Massachusetts Institute of Technology(MIT) computer scientists.
- The idea behind Kerberos is to authenticate users while preventing passwords from being sent over the internet.
- It uses secret-key cryptography and a trusted third party for authenticating client-server applications and verifying users‘ identities.
- But in the protocol's case, the three heads of Kerberos represent the client, the server, and the Key Distribution Center(KDC).

- Kerberos is an authentication protocol,
- Its a KDC, too.
- Several systems, including Windows 2000, use Kerberos.
- Originally designed at MIT, it has gone through several versions.


Authentication Server
Database
Kerberos
TGS
User
1)
2)
3)
4)
5)
6)
Server

Terms

- Authentication server – AS
- Ticket granting server – TGS
- Key distribution center - KDC

Kerberos working

- Initial Authentication
- Authentication Server Verification
- TGT Request
- TGS Verification
- Service Ticket
- Accessing the Service
- Service Verification
- Secure Communication

Kerberos working

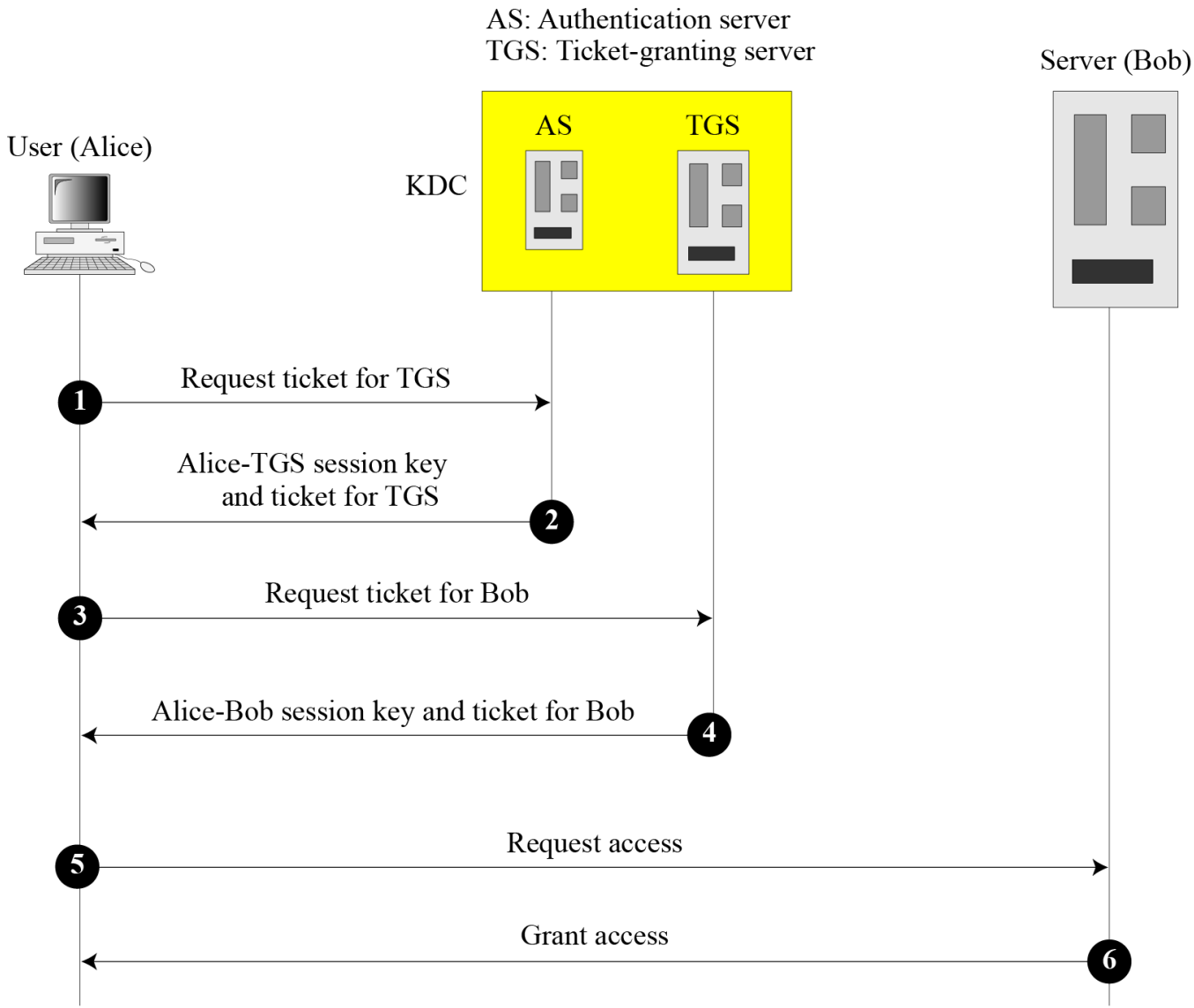
- **Initial Authentication**:
  - A user wants to access a network resource, such as a file server.
  - The user provides their username and password to a component called the "Authentication Server" (AS).
- **AS Verification**:
  - The AS checks the provided credentials and verifies the user's identity.
  - If the credentials are valid, the AS generates a small, secret encryption key called the Ticket Granting Ticket (TGT).
- **TGT Request**:
  - The user receives the TGT and wants to access a specific service, like a file server.
  - To do this, the user requests a service ticket from the Ticket Granting Server (TGS).
- **TGS Verification**:
  - The user's request, along with the TGT, is sent to the TGS.
  - The TGS validates the TGT and ensures that the user is allowed to access the requested service.

Kerberos working

- **Service Ticket**:
  - If the TGS approves the request, it creates a service ticket for the user. This ticket includes a session key.
- **Accessing the Service**:
  - The user takes the service ticket and forwards it to the desired service (e.g., the file server).
- **Service Verification**:
  - The service validates the service ticket using the session key it shares with the TGS.
- **Secure Communication**:
  - Once the service ticket is verified, the user and the service have a shared session key that they use to encrypt and decrypt their communication.

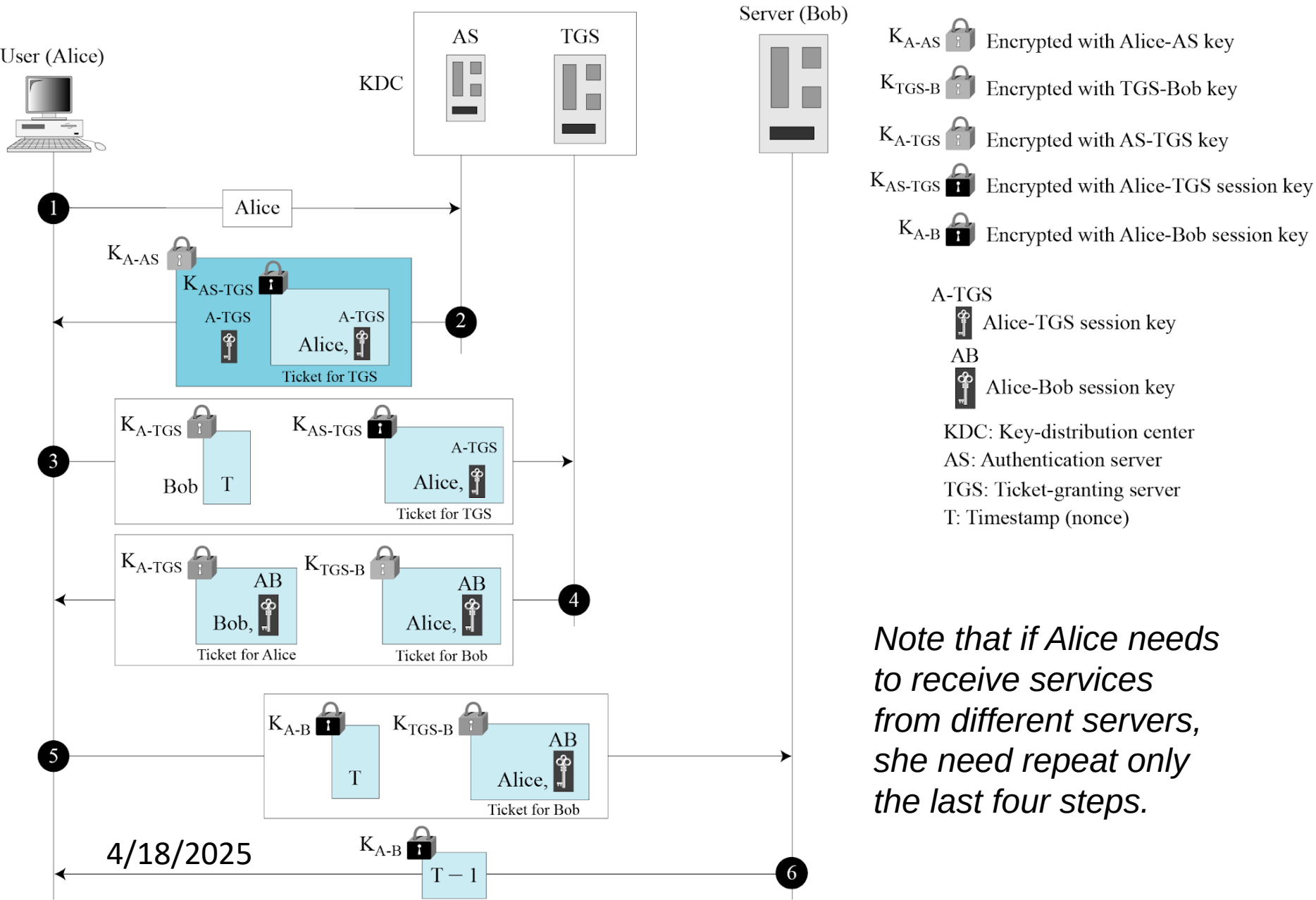
Kerberos working

- In simple terms, Kerberos is like a security guard for your network.
- It first checks your identity (username and password) and gives you a special ticket (TGT).
- Then, when you want to access different parts of the network (services), you use this ticket to get access.
- It's like showing your ID to the security guard (TGS) at each door you want to enter.
- If you're authorized, the guard gives you a key (service ticket) to that door, which you can use for secure communication with the service.
- This way, only authorized users can access specific network resources, and the communication between them is encrypted for added security.


AS: Authentication server
TGS: Ticket-granting server
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

- Authentication Server (AS)
  - The authentication server (AS) is the KDC in the Kerberos protocol.
- Ticket-Granting Server (TGS)
  - The ticket-granting server (TGS) issues a ticket for the real server (Bob).
- Real Server
  - The real server (Bob) provides services for the user (Alice).

Kerberos Example



K_{A-AS} Encrypted with Alice-AS key

K_{TGS-B} Encrypted with TGS-Bob key

K_{A-TGS} Encrypted with AS-TGS key

K_{AS-TGS} Encrypted with Alice-TGS session key

K_{A-B} Encrypted with Alice-Bob session key

A-TGS
Alice-TGS session key

AB
Alice-Bob session key

KDC: Key-distribution center
AS: Authentication server
TGS: Ticket-granting server
T: Timestamp (nonce)

Note that if Alice needs to receive services from different servers, she need repeat only the last four steps.

Kerberos versions

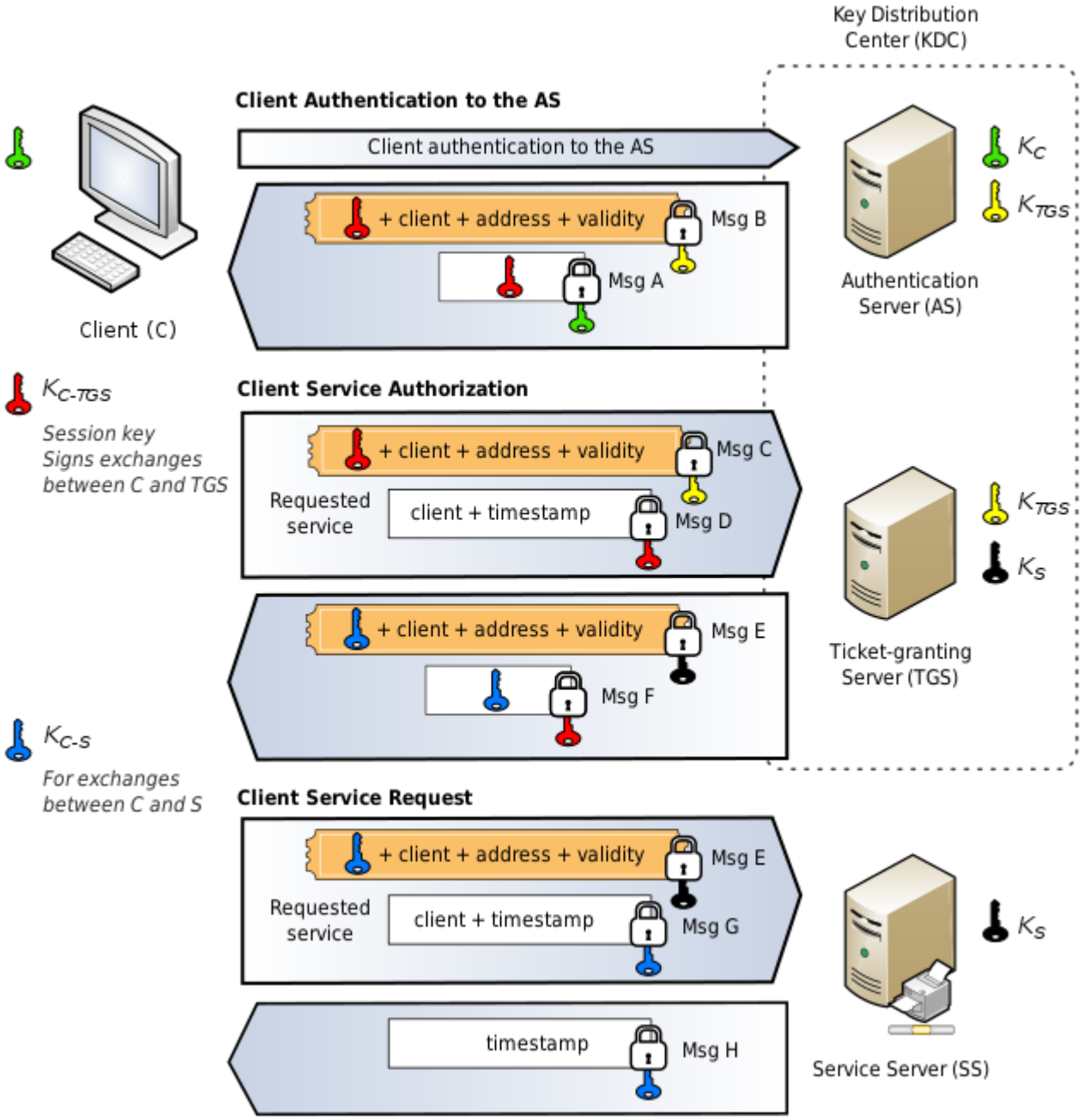
- Version 5 has a longer ticket lifetime.
- Version 5 allows tickets to be renewed.
- Version 5 can accept any symmetric-key algorithm.
- Version 5 uses a different protocol for describing data types.
- Version 5 has more overhead than version 4.

Kerberos

- The client authenticates itself to the **Authentication Server (AS)** which forwards the username to a **key distribution center (KDC)**.
- The KDC issues a time stamped **ticket-granting ticket (TGT)**, and encrypts it using the **ticket-granting service's (TGS)** secret key and returns the encrypted result to the user's workstation.
- This is done infrequently, typically at user logon;
- The TGT expires at some point although it may be transparently renewed by the user's session manager while they are logged in.

Kerberos

- When the client needs to communicate with a service on another node (a "principal", in Kerberos parlance), the client sends the TGT to the TGS,
- The service must have already been registered with the TGS with a **Service Principal Name (SPN)**.
- The client uses the SPN to request access to this service.
- After verifying that the TGT is valid and that the user has access to a the requested service, the TGS issues ticket and session keys to the client.
- The client then sends the ticket to the **service server (SS)** along with its service request.


Key Distribution Center (KDC)
Client Authentication to the AS
Client authentication to the AS
+ client + address + validity
Msg B
Msg A
Client (C)
K_C
K_{TGS}
Authentication Server (AS)
$K_{C\text{-}TGS}$
Session key
Signs exchanges between C and TGS
Client Service Authorization
+ client + address + validity
Msg C
Requested service
client + timestamp
Msg D
+ client + address + validity
Msg E
Msg F
Ticket-granting Server (TGS)
K_{TGS}
K_S
$K_{C\text{-}S}$
For exchanges between C and S
Client Service Request
+ client + address + validity
Msg E
Requested service
client + timestamp
Msg G
timestamp
Msg H
Service Server (SS)
K_S

Client authentication

- The client sends a cleartext message of the user ID to the requesting services on behalf of the user. (Note: Neither the secret key nor the password is sent to the AS.)
- The AS checks if the client is in its database.
- If it is, the AS generates the secret key by hashing the password of the user found at the database and sends back the following two messages to the client:
  - Message A: *Client/TGS Session Key* encrypted using the secret key of the client/user.
  - Message B: *Ticket-Granting-Ticket* (TGT, which includes the client ID, client network address, ticket validity period, and the *Client/TGS Session Key*) encrypted using the secret key of the TGS.
- Once the client receives messages A and B, it attempts to decrypt message A with the secret key generated from the password entered by the user.
- If the user entered password does not match the password in the AS database, the client's secret key will be different and thus unable to decrypt message A.
- With a valid password and secret key the client decrypts message A to obtain the *Client/TGS Session Key*.
- This session key is used for further communications with the TGS. (Note: The client cannot decrypt Message B, as it is encrypted using TGS's secret key.)
- At this point, the client has enough information to authenticate itself to the TGS.

Client Service Authorization

- When requesting services, the client sends the following messages to the TGS:
  - Message C: Composed of the message B (the encrypted TGT using the TGS secret key) and the ID of the requested service.
  - Message D: Authenticator (which is composed of the client ID and the timestamp), encrypted using the *Client/TGS Session Key*.
- Upon receiving messages C and D, the TGS retrieves message B out of message C.
- It decrypts message B using the TGS secret key and gets : *Client/TGS Session Key* and the client ID (both are in the TGT).
- Using this *Client/TGS Session Key*, the TGS decrypts message D (Authenticator) and compares the client IDs from messages B and D; if they match, the server sends the following two messages to the client:
  - Message E: *Client-to-server ticket* (which includes the client ID, client network address, validity period, and *Client/Server Session Key*) encrypted using the service's secret key.
  - Message F: *Client/Server Session Key* encrypted with the *Client/TGS Session Key*.

Client Service Request

- Upon receiving messages E and F from TGS, the client has enough information to authenticate itself to the Service Server (SS).
- The client connects to the SS and sends the following two messages:
  - Message E: From the previous step (the *Client-to-server ticket*, encrypted using service's secret key).
  - Message G: A new Authenticator, which includes the client ID, timestamp and is encrypted using *Client/Server Session Key*.
- The SS decrypts the ticket (message E) using its own secret key to retrieve the *Client/Server Session Key*.
- Using the sessions key, SS decrypts the Authenticator and compares client ID from messages E and G, if they match server sends the following message to the client to confirm its true identity and willingness to serve the client:
  - Message H: The timestamp found in client's Authenticator (plus 1 in version 4, but not necessary in version 5[10][11]), encrypted using the *Client/Server Session Key*.
- The client decrypts the confirmation (message H) using the *Client/Server Session Key* and checks whether the timestamp is correct.
- If so, then the client can trust the server and can start issuing service requests to the server.
- The server provides the requested services to the client.

Advantages of Kerberos

- **Passwords aren't exposed to eavesdropping**
- **Password is only typed to the local workstation**
  - It never travels over the network
  - It is never transmitted to a remote server
- **Password guessing more difficult**
- **Single Sign-on**
  - More convenient: only one password, entered once
  - Users may be less likely to store passwords
- **Stolen tickets hard to reuse**
  - Need authenticator as well, which can't be reused
- **Much easier to effectively secure a small set of limited access machines (the AS's)**
- **Easier to recover from host compromises**
- **Centralized user account administration**

Drawbacks and limitations

- Kerberos has strict time requirements, which means that the clocks of the involved hosts must be synchronized within configured limits.
  - The tickets have a time availability period, and if the host clock is not synchronized with the Kerberos server clock, the authentication will fail.
  - The default configuration per MIT requires that clock times be no more than five minutes apart.
  - In practice, Network Time Protocol daemons are usually used to keep the host clocks synchronized
- The administration protocol is not standardized and differs between server implementations.
- In case of symmetric cryptography adoption (Kerberos can work using symmetric or asymmetric (public-key) cryptography), since all authentications are controlled by a centralized key distribution center (KDC), compromise of this authentication infrastructure will allow an attacker to impersonate any user.
- Each network service that requires a different host name will need its own set of Kerberos keys. This complicates virtual hosting and clusters.
- Kerberos requires user accounts and services to have a trusted relationship to the Kerberos token server.
- The required client trust makes creating staged environments difficult: Either domain trust relationships need to be created that prevent a strict separation of environment domains, or additional user clients need to be provided for each environment.

Can Kerberos Be Hacked?

- Yes. Because it is one of the most widely used authentication protocols, hackers have developed several ways to crack into Kerberos.
- Most of these hacks take advantage of a vulnerability, weak passwords, or malware – sometimes a combination of all three.
- Some of the more successful methods of hacking Kerberos include:
  - Pass-the-ticket: the process of forging a session key and presenting that forgery to the resource as credentials
  - Golden Ticket: A ticket that grants a user domain admin access
  - Silver Ticket: A forged ticket that grants access to a service
  - Credential stuffing/ Brute force: automated continued attempts to guess a password
  - Encryption downgrade with Skeleton Key Malware: A malware that can bypass Kerberos, but the attack must have Admin access
  - DCShadow attack: a new attack where attackers gain enough access inside a network to set up their own DC to use in further infiltration

Questions?