SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCS136-NETWORK SECURITY

UNIT-3
Security Functions and Data Security

SYLLABUS

Public Key Crypto system - Diffie-Hellman Key Exchange - Key management Techniques - Hash Functions- Requirements - Hash Algorithm - MD5, SHA_1 - Message Authentication Code (MAC) – HMAC - Digital Signature - User Authentication-Kerberos - X.509 Certificates - X.509 Formats, Public Key Infrastructure - PKIX Architecture – Model - Management Functions

Public-Key Cryptosystems

- **Evolved from an attempt to solve** two of the most difficult problems associated with **symmetric encryption.**
  - **Key distribution**
  - **Digital signature**
- **Diffie and Hellman** in **1976** came up with a method
- Each user **generates a pair of keys** to be used for the encryption and decryption
  - Each user places one of the two keys in a public register or other accessible file. This is the **public key**
  - The other key is kept private, which is the **private key**
  - Either of the two related keys can be used for encryption, with the other used for decryption.

Public-Key Cryptosystems

A public-key encryption scheme has six ingredients

- **Plaintext**
- **Encryption algorithm**
- **Public key**
- **Private key**
- **Cipher text**
- **Decryption algorithm**

Public-Key Cryptosystems

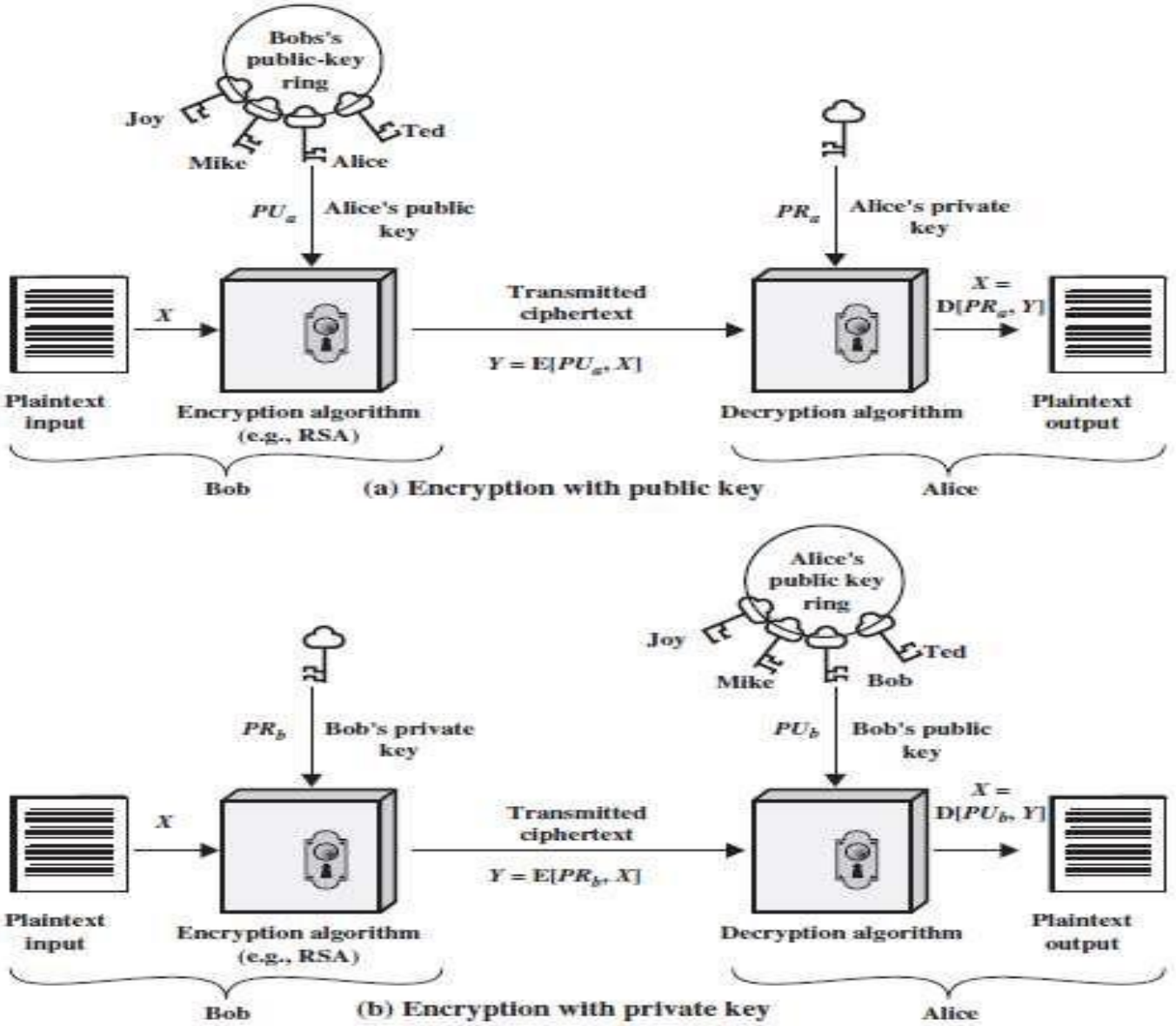


Figure 9.1 Public-Key Cryptography

Encryption using Public key

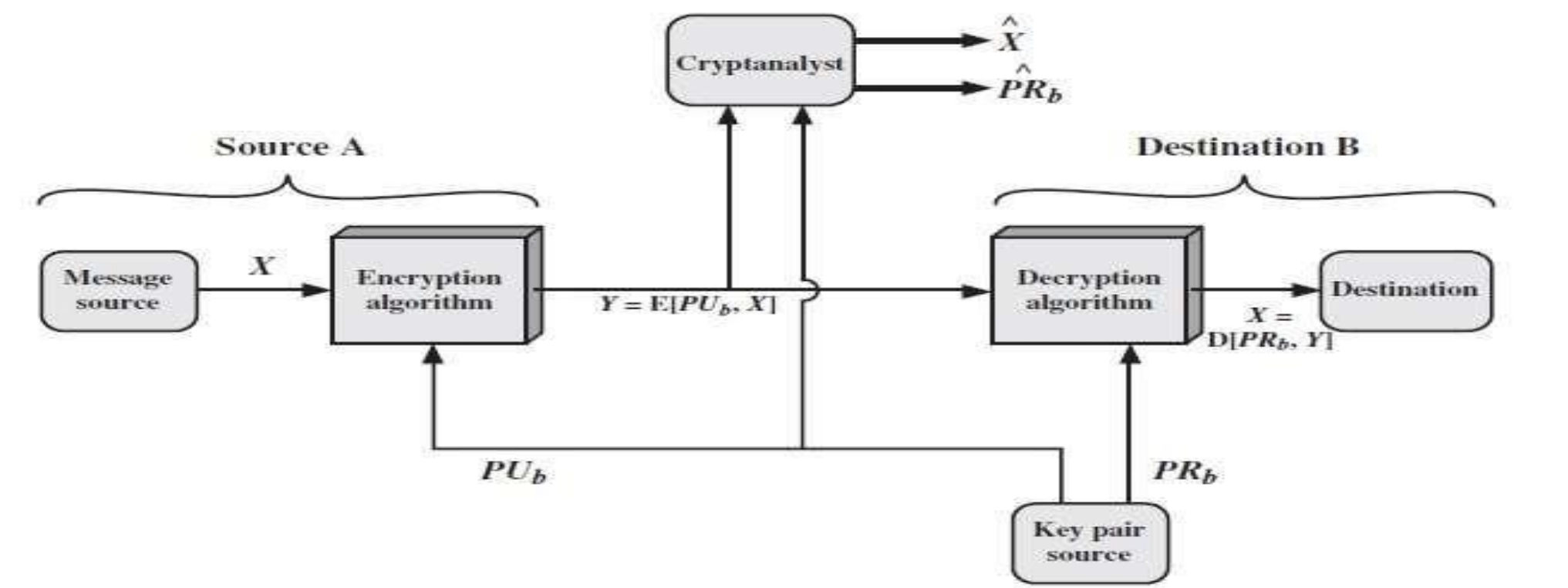


Figure 9.2 Public-Key Cryptosystem: Secrecy

- A forms the **ciphertext Y** , given by

  $Y = E(PUb, X)$

- Receiver in possession of the matching private key ***PRb***, is able to invert the transformation

  $X = D(PRb, Y)$

Encryption using Public key

- An **adversary**, observing *Y* and having access to *PUb*, but nothaving access to *PRb* or *X*, must attempt to recover *X* and/or *PRb*.
- It is assumed that the adversary does **have knowledge** of the
- encryption (E) and decryption (D) **algorithms**.
- If the adversary is interested only in this **particular message**, then the focus of effort is to recover *X* by generating a plaintext estimate *X*n .
- Often, however, the adversary is interested in being able to read **future messages as well**, in which case an attempt is made to recover *PRb* by generating an **estimate *PRnb***.

Encryption using Private key

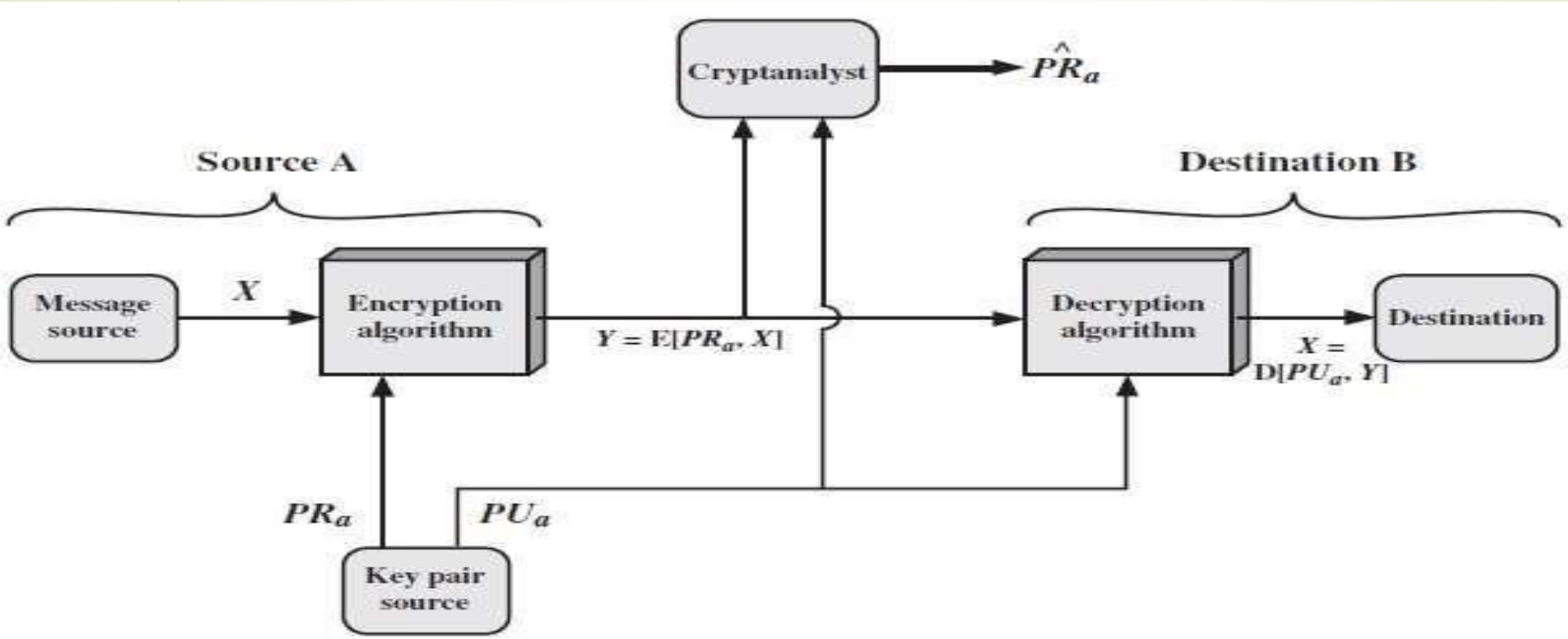


Figure 9.3 Public-Key Cryptosystem: Authentication

Encryption using Private key

- A encrypts it using A's private key before transmitting it.
- B can **decrypt** the message **using A's public key**.

$$Y = E(PRa, X)$$

$$X = D(PUa, Y)$$

- Only A could have prepared the message, Therefore, the entire encrypted message serves as a **digital signature**.

- In addition, it **is impossible to alter** the message without access to A's private key
- The message being sent is **safe from alteration**.
- But **not confidentiality** because any observer can decrypt the message by using the sender's public key.

Authentication and confidentiality

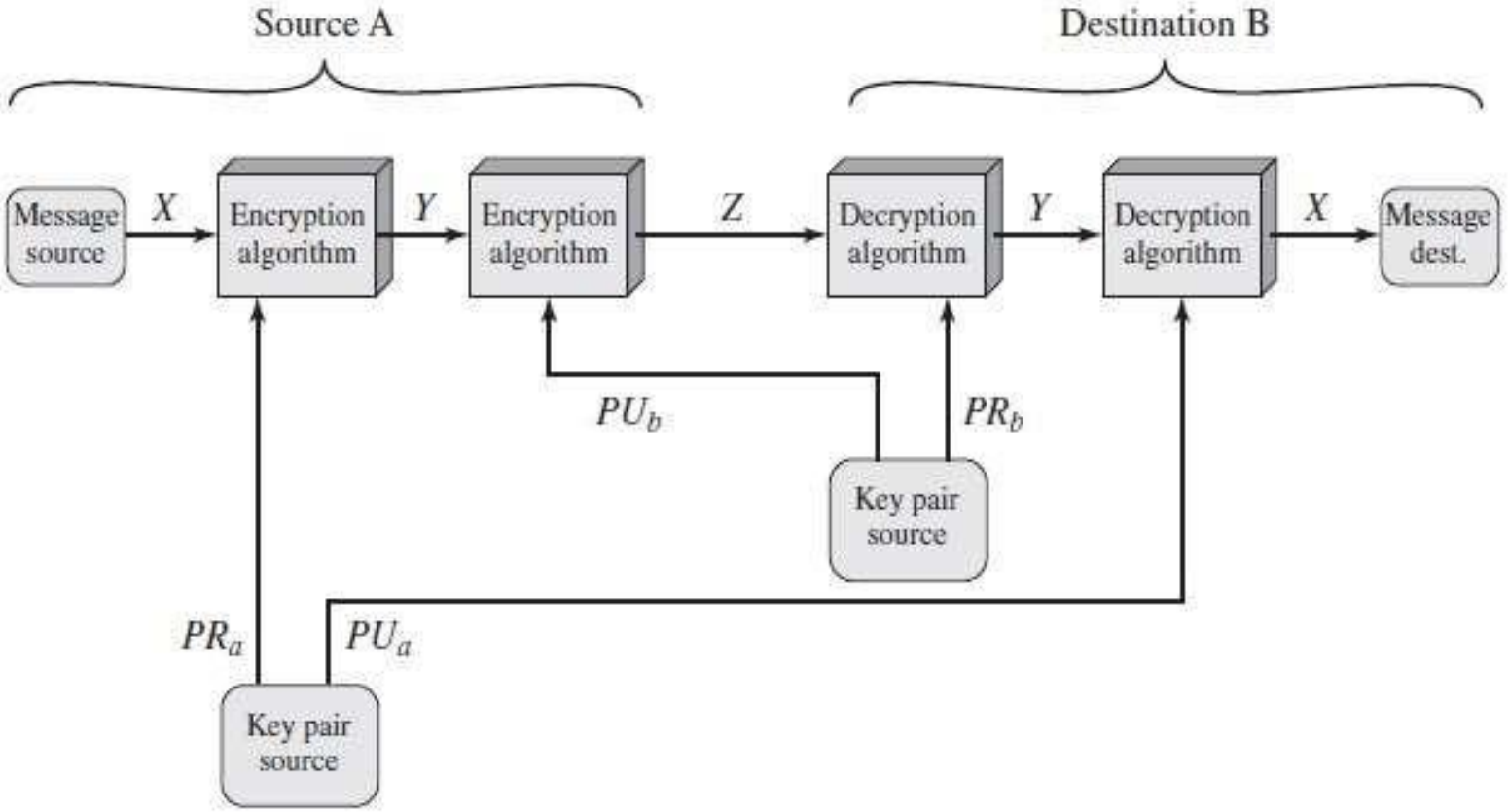


Figure 9.4 Public-Key Cryptosystem: Authentication and Secrecy

Authentication and confidentiality

Both the authentication function and confidentiality by a double use of the public-key scheme

$$Z = \mathrm{E}(PUb, \mathrm{E}(PRa, X))$$

$$X = \mathrm{D}(PUa, \mathrm{D}(PRb, Z))$$

- We begin by encrypting a message, using the sender's private key.
- This provides the **digital signature**.
- Next, we encrypt again, using the receiver's public key.
- The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key.
- Thus, **confidentiality** is provided.

Applications for Public-Key Cryptosystems

- **Encryption/decryption**
- **Digital signature**
- **Key exchange**
- Some algorithms are suitable for all three applications,
- Some used only for one or two of these applications.

Requirements for Public-Key Cryptography

- **1.** It is computationally easy for a party B to generate a pair of key (public key *PUb*, private key *PRb*).

- **2.** It is computationally easy for a sender A, knowing the public key and the message to be encrypted, *M*, to generate the corresponding ciphertext:

$$C = \mathrm{E}(PUb, M)$$

- **3.** It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

$$M = \mathrm{D}(PRb, C) = \mathrm{D}[PRb, \mathrm{E}(PUb, M)]$$

- Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange

- first public-key type scheme proposed
- by Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
- is a practical method for public exchange of a secret key
- used in a number of commercial products

Diffie-Hellman Key Exchange

- a public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

Diffie-Hellman Setup

- all users agree on global parameters:
  - large prime integer or polynomial q
  - a being a primitive root mod q
- each user (eg. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$
- each user makes public that key y_A

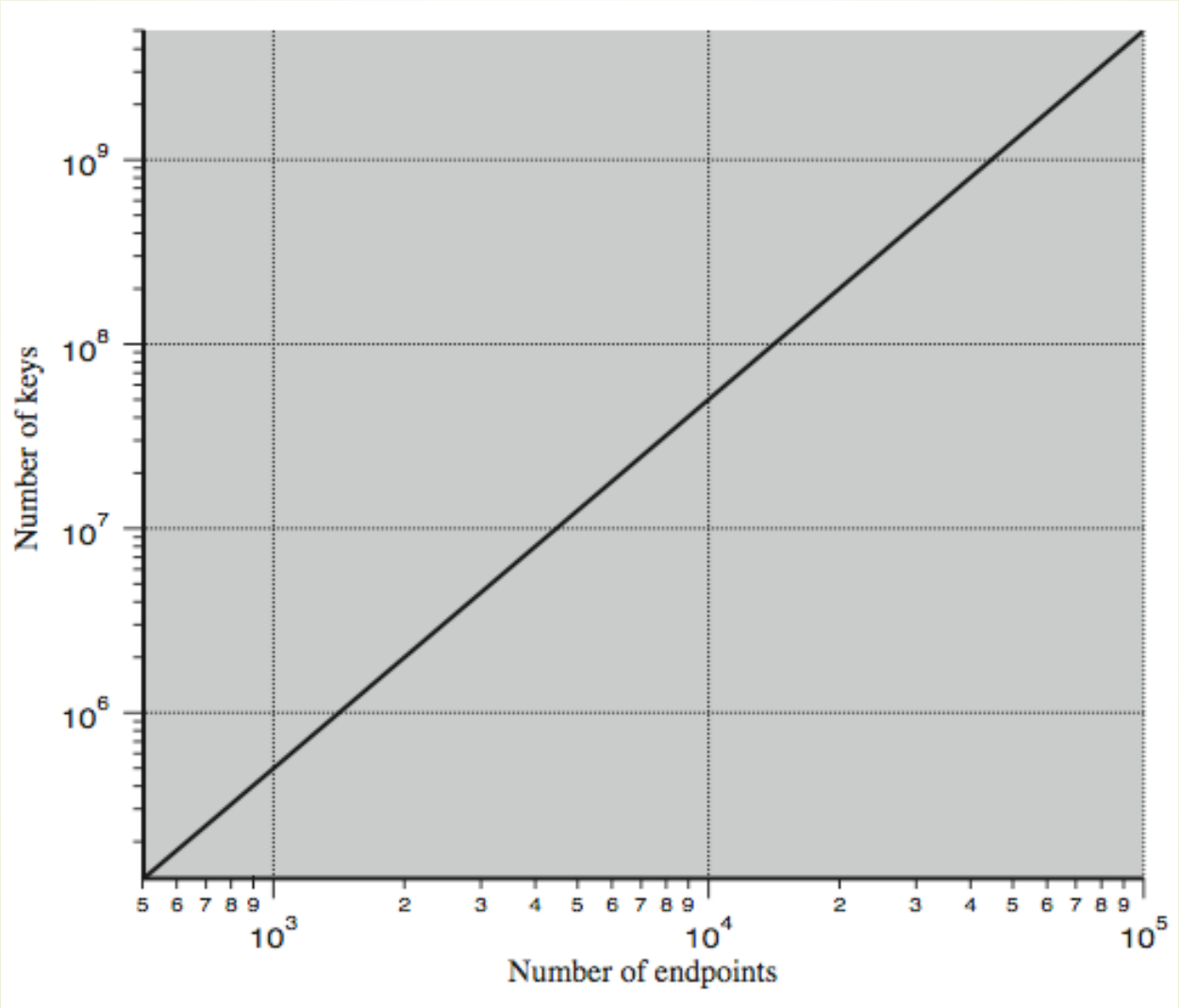
Diffie-Hellman Key Exchange

- shared session key for users A & B is K_{AB}:
  - $K_{AB} = a^{x_A . x_B} \bmod q$
  - $= y_A^{x_B} \bmod q$ (which **B** can compute)
  - $= y_B^{x_A} \bmod q$ (which **A** can compute)
- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- attacker needs an x, must solve discrete log

Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime $q=353$ and $a=3$
- select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- compute respective public keys:
  - $y_A=3^{97} \mod 353 = 40$ (Alice)
  - $y_B=3^{233} \mod 353 = 248$ (Bob)
- compute shared session key as:
  - $K_{AB}= y_B^{x_A} \mod 353 = 248^{97} = 160$ (Alice)
  - $K_{AB}= y_A^{x_B} \mod 353 = 40^{233} = 160$ (Bob)

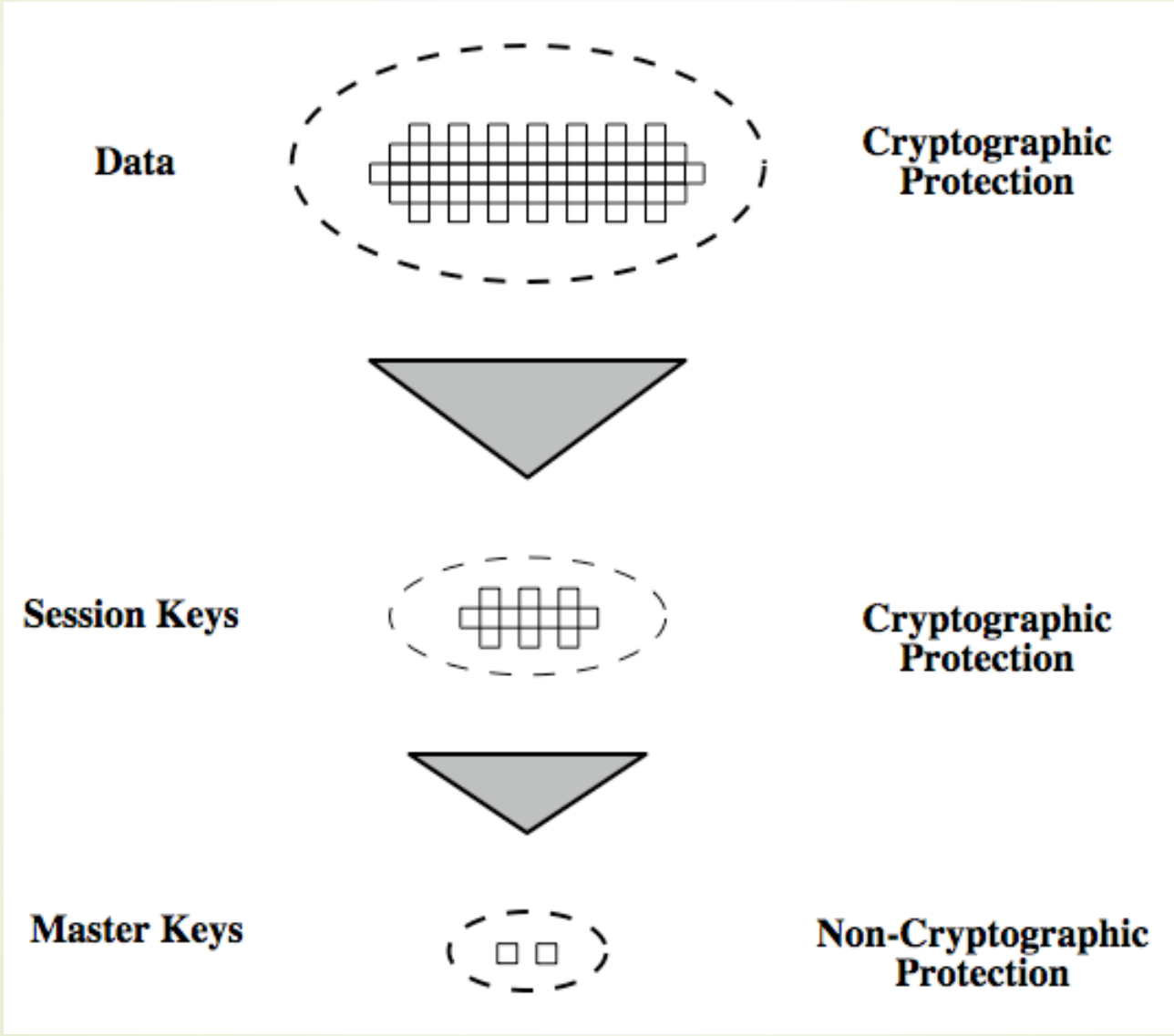
Key Distribution Task



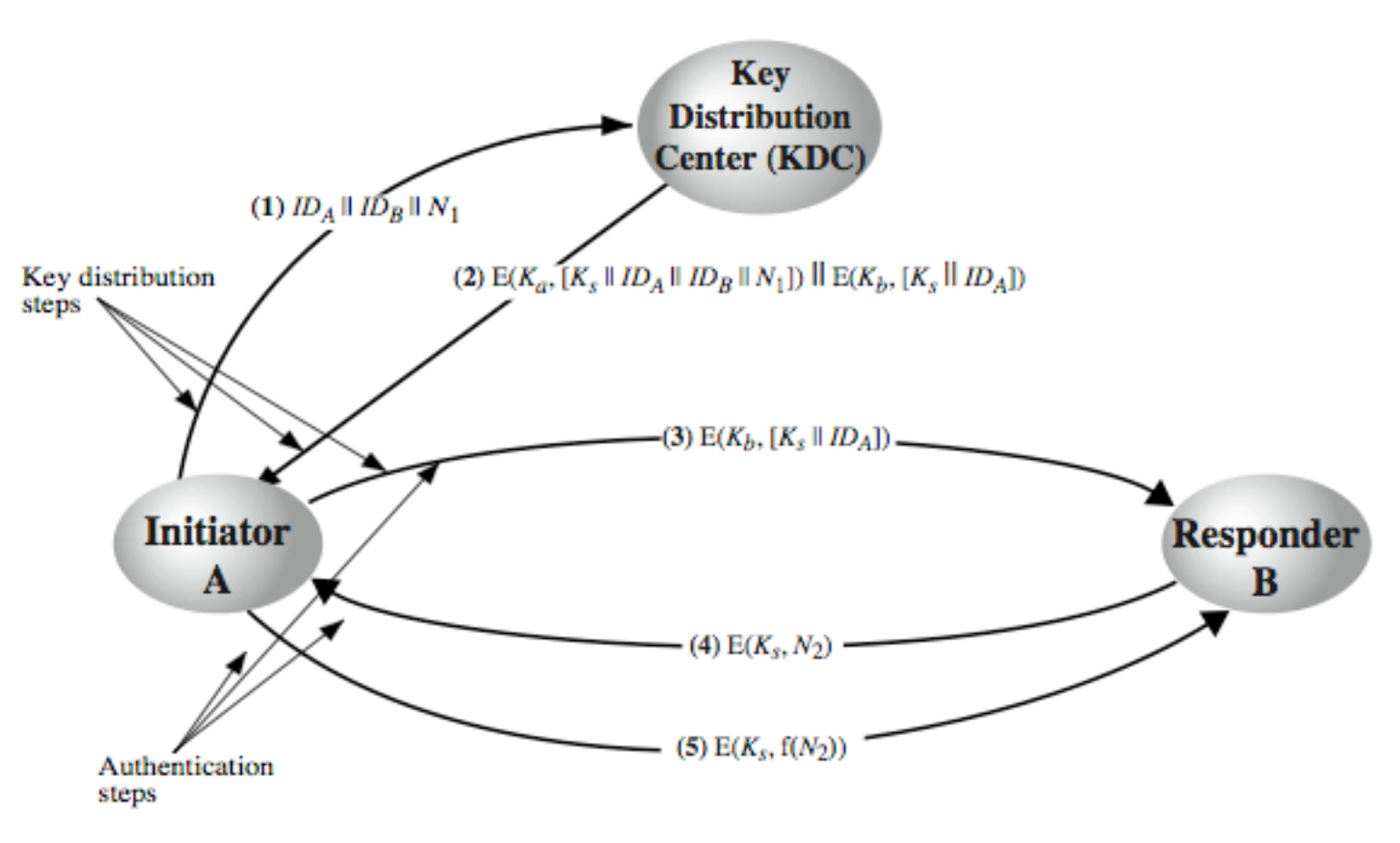
Key Hierarchy

- session key
  - temporary key
  - used for encryption of data between users
  - for one logical session then discarded
- master key
  - used to encrypt session keys
  - shared by user & key distribution center

Key Hierarchy



Key Distribution Scenario



Key Distribution Issues

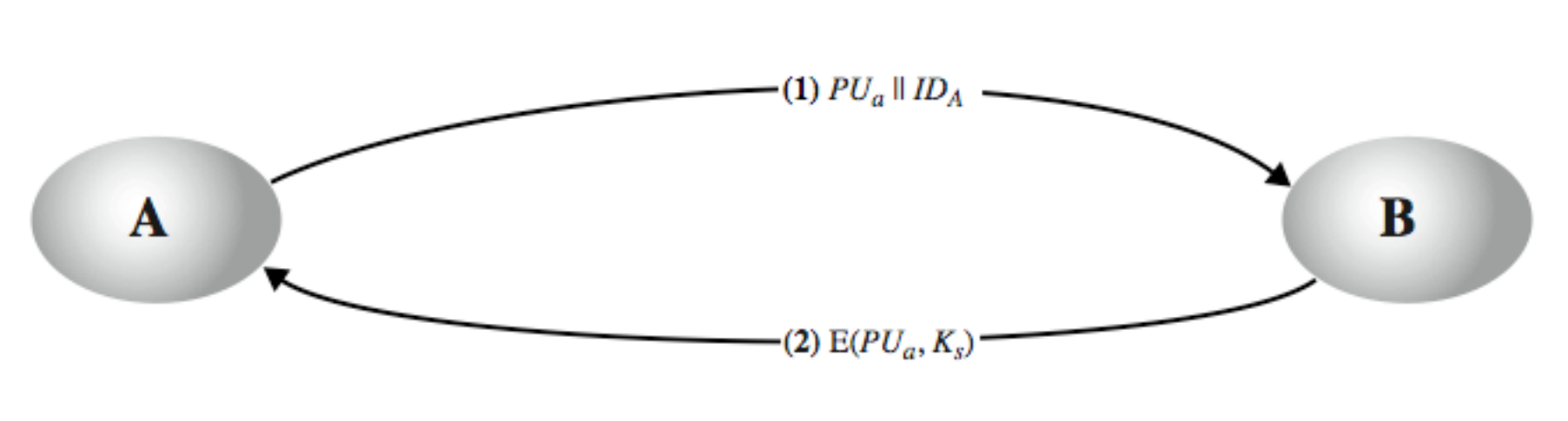
- hierarchies of KDC's required for large networks, but must trust each other
- session key lifetimes should be limited for greater security
- use of automatic key distribution on behalf of users, but must trust system
- use of decentralized key distribution
- controlling key usage

Symmetric Key Distribution Using Public Keys

- public key cryptosystems are inefficient
  - so almost never use for direct data encryption
  - rather use to encrypt secret keys for distribution

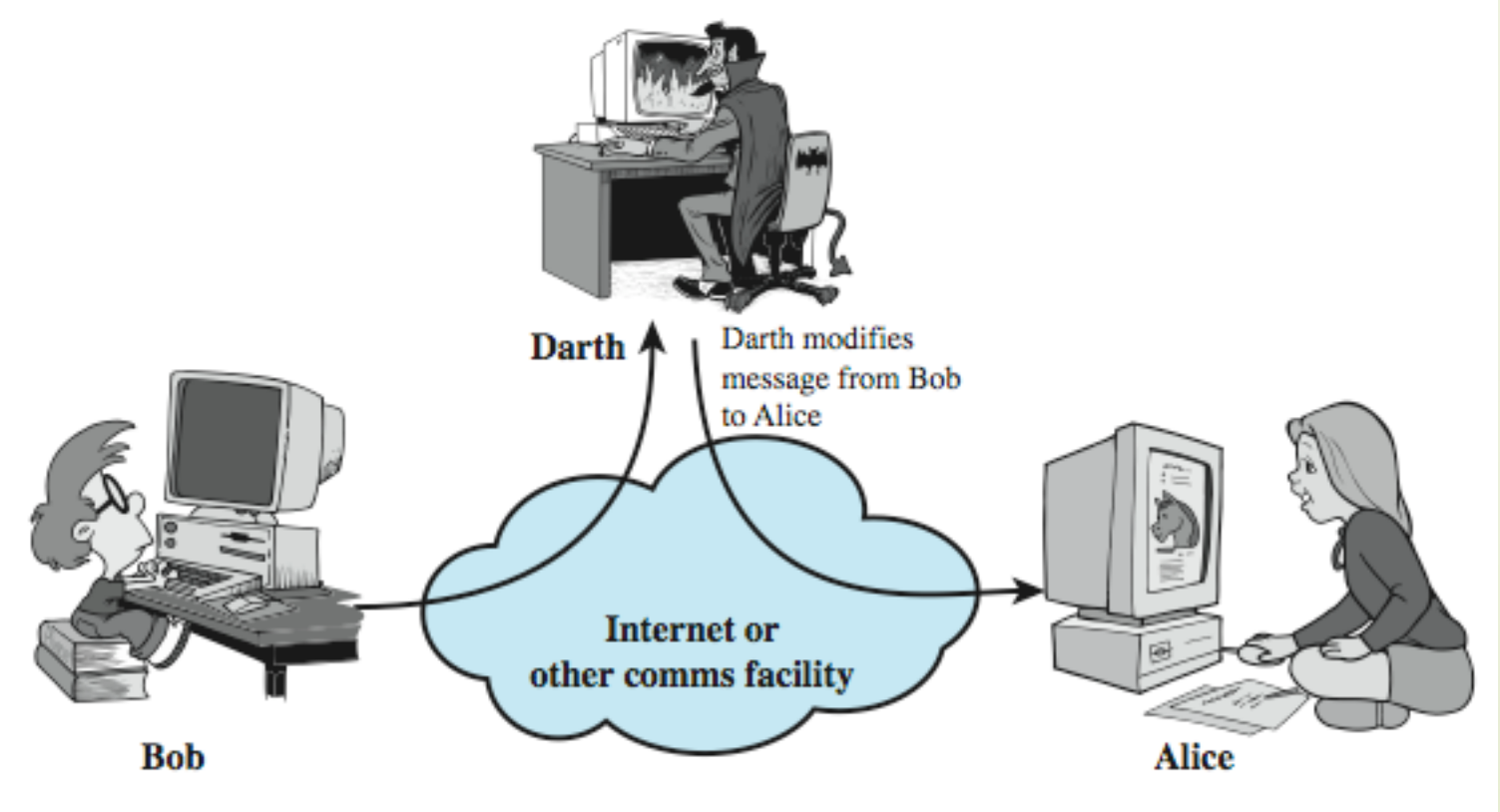
Simple Secret Key Distribution

- Merkle proposed this very simple scheme
  - allows secure communications
  - no keys before/after exist

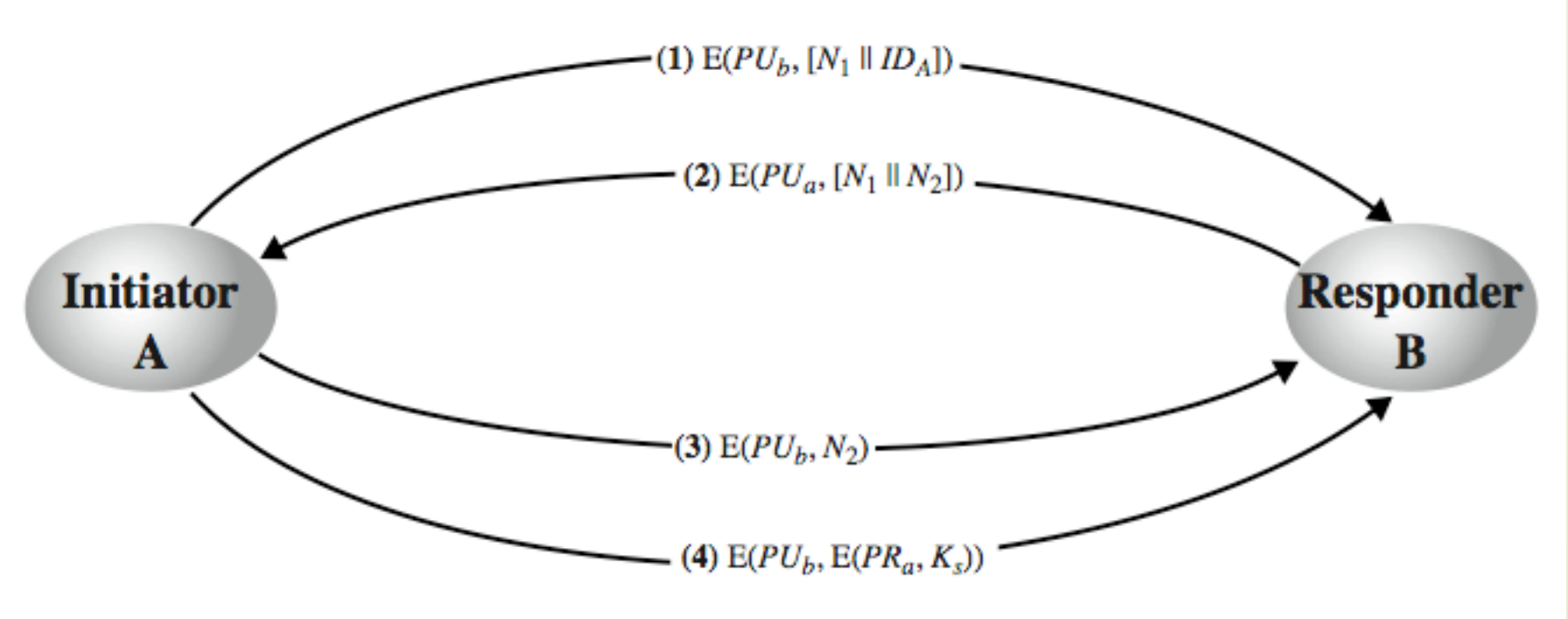


Man-in-the-Middle Attack

- this very simple scheme is vulnerable to an active man-in-the-middle attack



Secret Key Distribution with Confidentiality and Authentication



Hybrid Key Distribution

- retain use of private-key KDC
- shares secret master key with each user
- distributes session key using master key
- public-key used to distribute master keys
  - especially useful with widely distributed users
- rationale
  - performance
  - backward compatibility

Distribution of Public Keys

- can be considered as using one of:
  - public announcement
  - publicly available directory
  - public-key authority
  - public-key certificates

Public Announcement

- users distribute public keys to recipients or broadcast to community at large
  - eg. append PGP keys to email messages or post to news groups or email list
- major weakness is forgery
  - anyone can create a key claiming to be someone else and broadcast it
  - until forgery is discovered can masquerade as claimed user

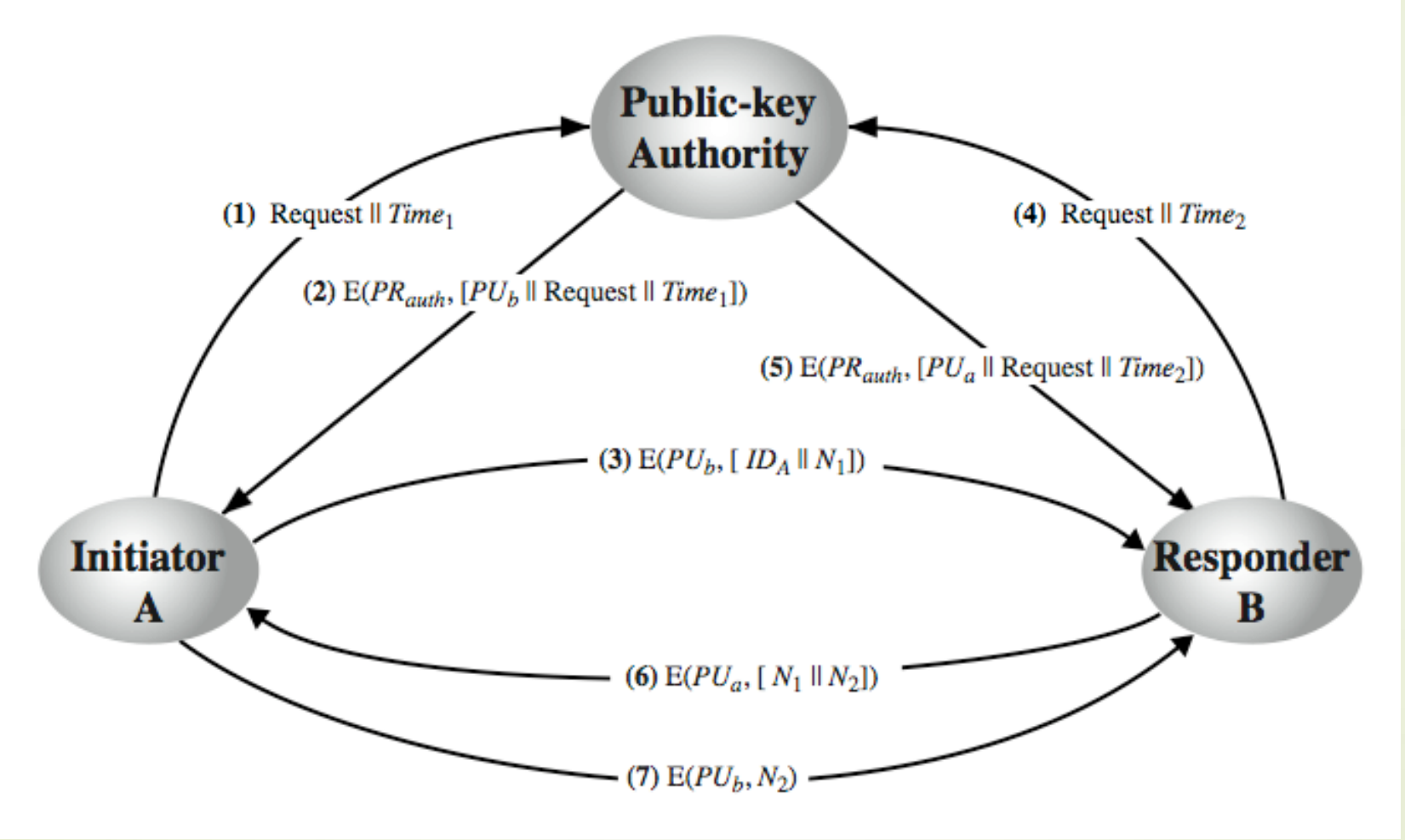
Publicly Available Directory

- can obtain greater security by registering keys with a public directory
- directory must be trusted with properties:
  - contains {name,public-key} entries
  - participants register securely with directory
  - participants can replace key at any time
  - directory is periodically published
  - directory can be accessed electronically
- still vulnerable to tampering or forgery

Public-Key Authority

- improve security by tightening control over distribution of keys from directory
- has properties of directory
- and requires users to know public key for the directory
- then users interact with directory to obtain any desired public key securely
  - does require real-time access to directory when keys are needed
  - may be vulnerable to tampering

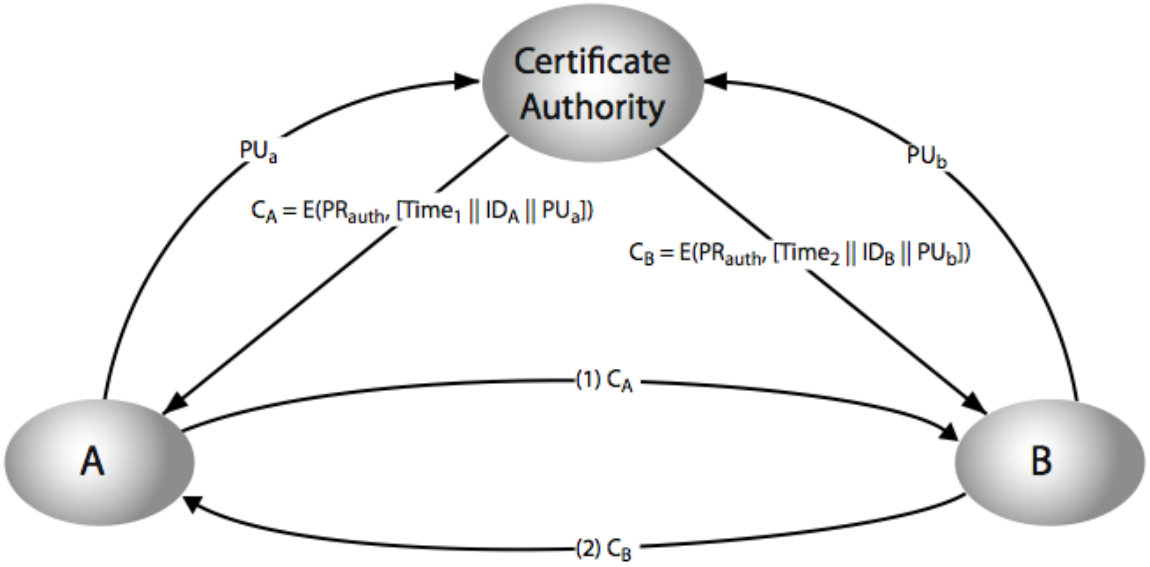
Public-Key Authority



Public-Key Certificates

- certificates allow key exchange without real-time access to public-key authority
- a certificate binds **identity** to **public key**
  - usually with other info such as period of validity, rights of use etc
- with all contents **signed** by a trusted Public-Key or Certificate Authority (CA)
- can be verified by anyone who knows the public-key authorities public-key

Public-Key Certificates



Public-Key Distribution of Secret Keys

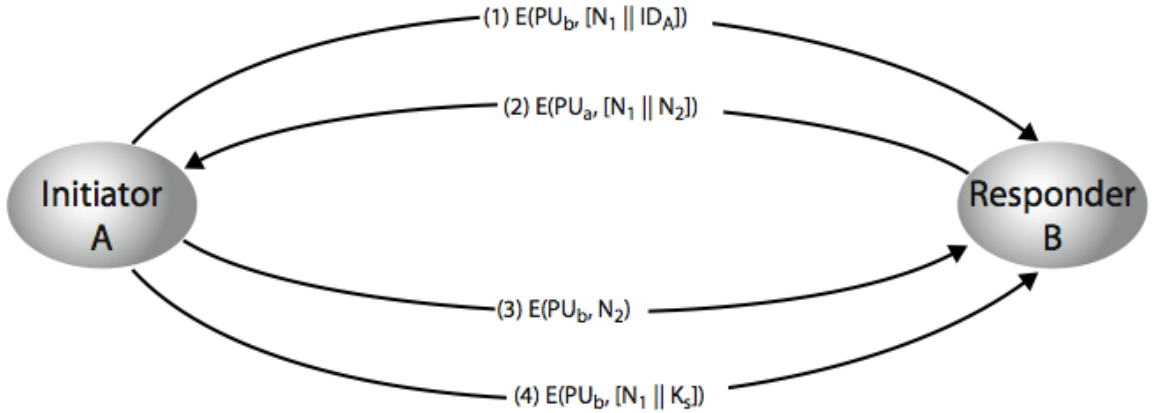
- use previous methods to obtain public-key
- can use for secrecy or authentication
- but public-key algorithms are slow
- so usually want to use private-key encryption to protect message contents
- hence need a session key
- have several alternatives for negotiating a suitable session

Simple Secret Key Distribution

- proposed by Merkle in 1979
  - A generates a new temporary public key pair
  - A sends B the public key and their identity
  - B generates a session key K sends it to A encrypted using the supplied public key
  - A decrypts the session key and both use
- problem is that an opponent can intercept and impersonate both halves of protocol

Public-Key Distribution of Secret Keys

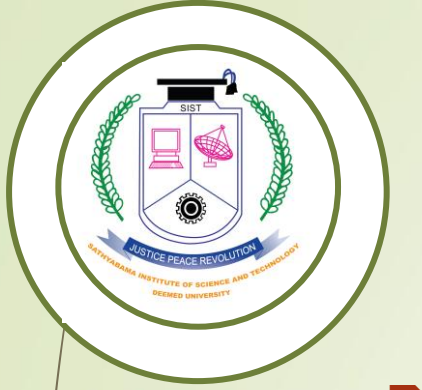
- if have securely exchanged public-keys:



Hybrid Key Distribution

- retain use of private-key KDC
- shares secret master key with each user
- distributes session key using master key
- public-key used to distribute master keys
  - especially useful with widely distributed users
- rationale
  - performance
  - backward compatibility

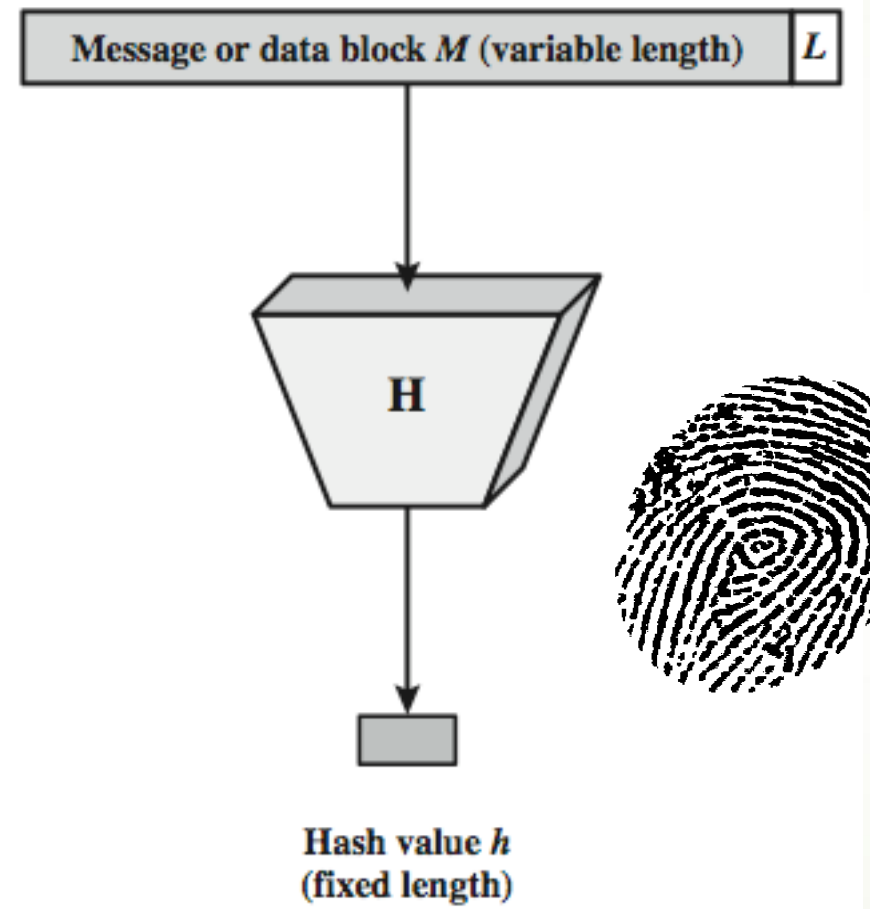
Cryptographic Hash Functions



Topics

- **Overview of Cryptography Hash Function**
- Usages
- Properties
- Hashing Function Structure
- Attack on Hash Function
- The Road to new Secure Hash Standard

Hash Function



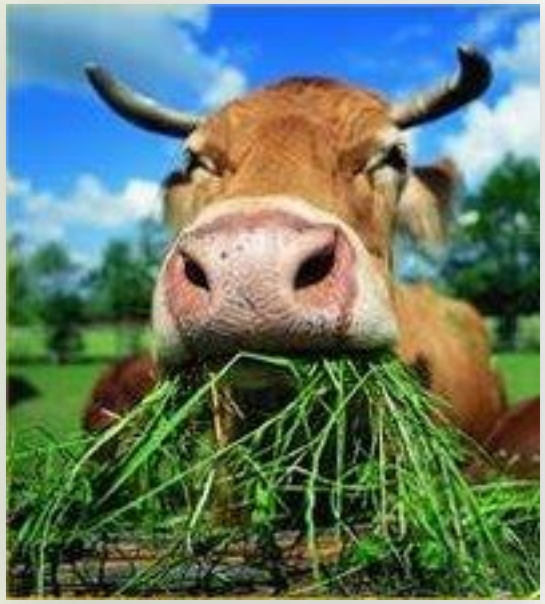
- The hash value represents concisely the longer message
  - may called the *message digest*
- A message digest is as a ``digital fingerprint'' of the original document

condenses arbitrary message to fixed size

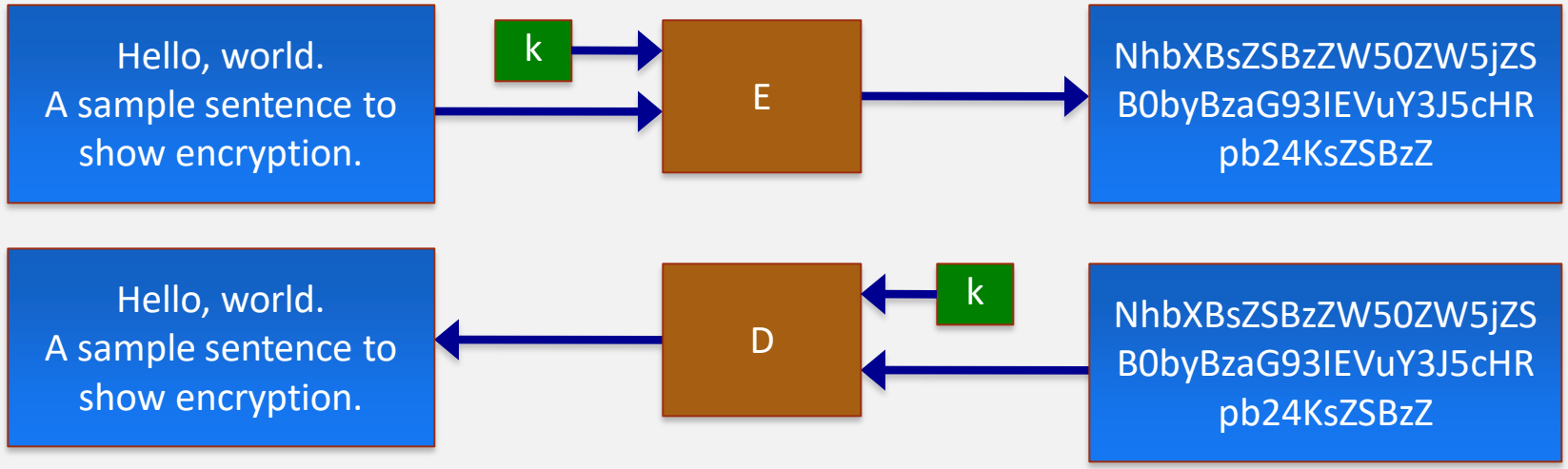
$$h = H(M)$$

Chewing functions

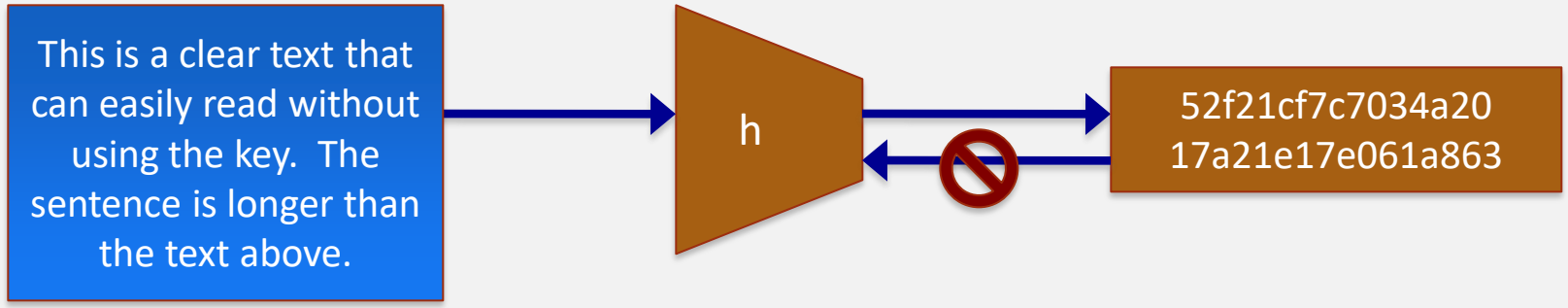
- Hashing function as “chewing” or “digest” function



Hashing V.S. Encryption



- Encryption is two way, and requires a key to encrypt/decrypt



- Hashing is one-way. There is no 'de-hashing'

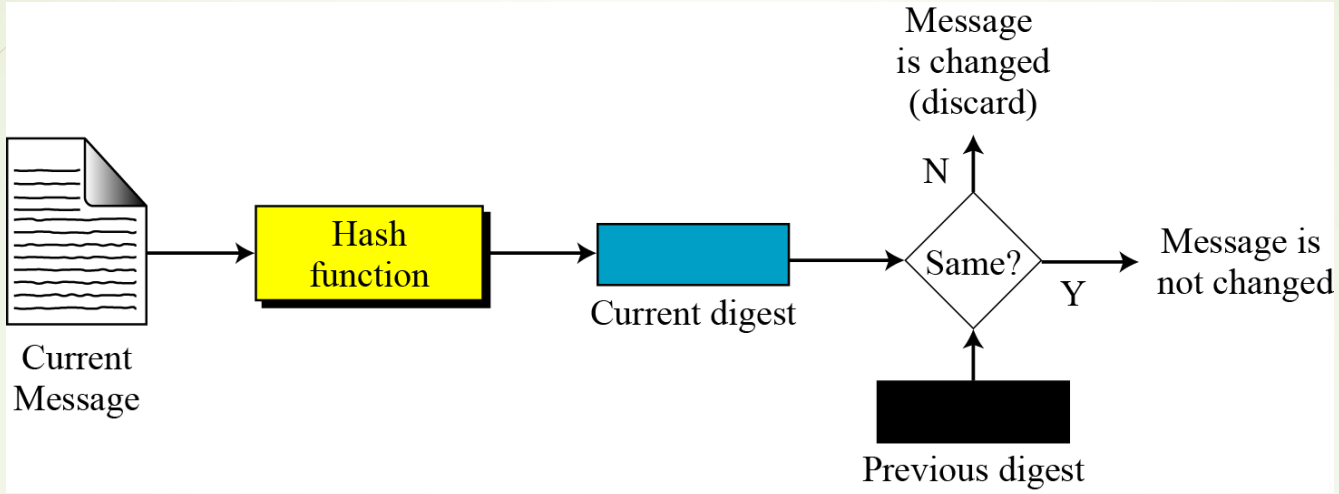
Motivation for Hash Algorithms

- Intuition
  - Limitation on non-cryptographic checksum
  - Very possible to construct a message that matches the checksum
- Goal
  - Design a code where the original message can not be inferred based on its checksum
  - such that an accidental or intentional change to the message will change the hash value

Hash Function Applications

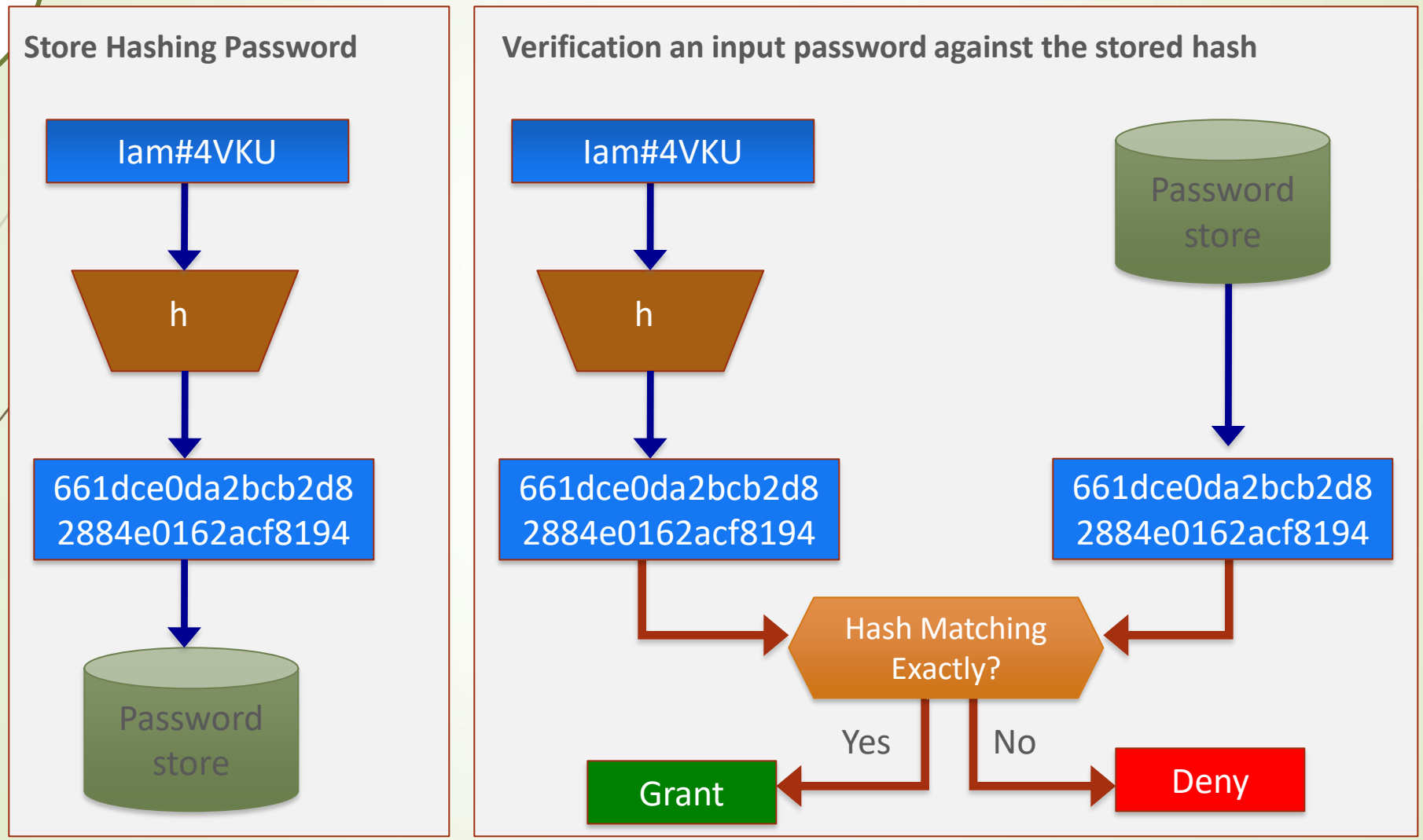
- Used Alone
  - Fingerprint -- file integrity verification, public key fingerprint
  - Password storage (one-way encryption)
- Combined with encryption functions
  - Hash based Message Authentication Code (HMAC)
    - protects both a message's integrity and confideltaility
  - Digital signature
    - Ensuring Non-repudiation
    - Encrypt hash with private (signing) key and verify with public (verification) key

Integrity



- to create a one-way password file
  - store hash of password not actual password
- for intrusion detection and virus detection
  - keep & check hash of files on system

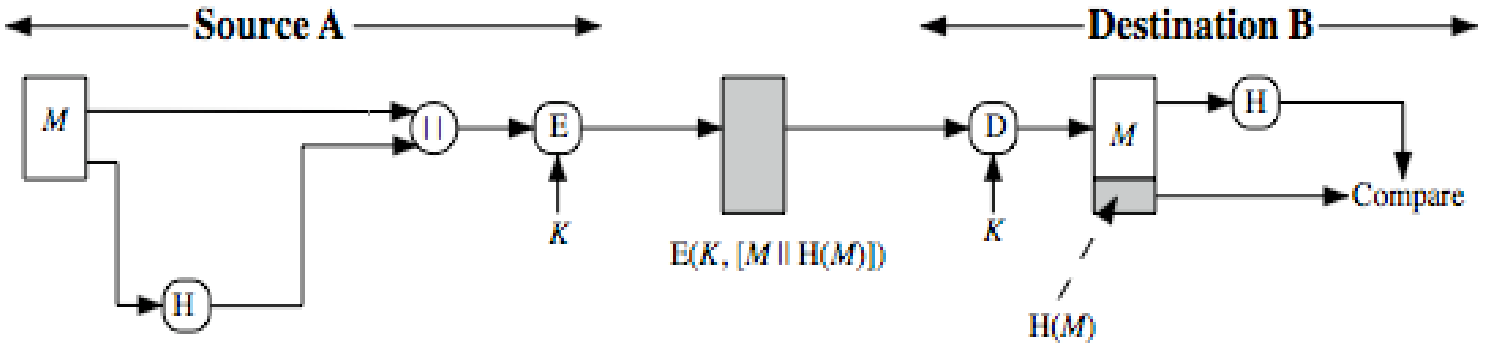
Password Verification



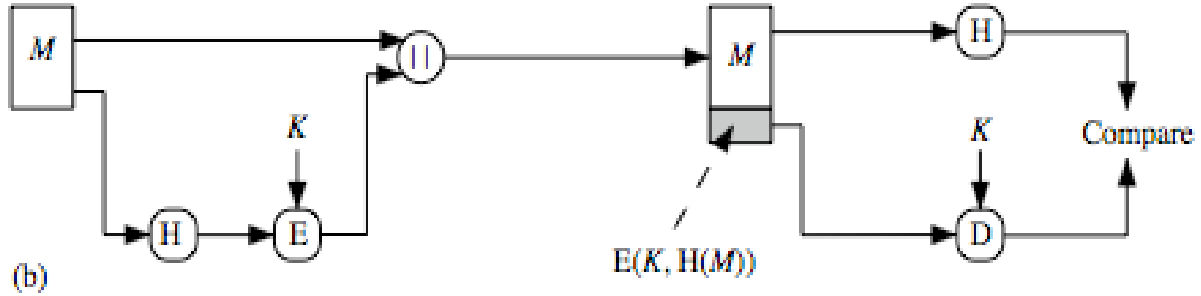
Topics

- Overview of Cryptography Hash Function
- **Usages**
- Properties
- Hashing Function Structure
- Attack on Hash Function
- The Road to new Secure Hash Standard

Hash Function Usages (I)

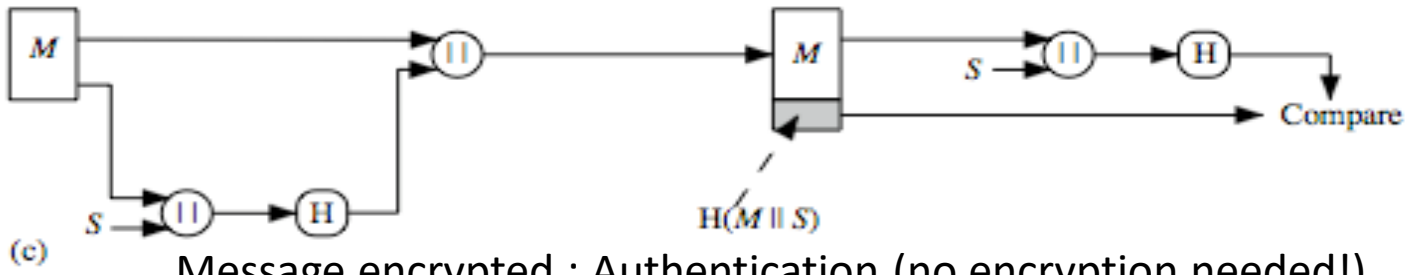


Message encrypted : Confidentiality and authentication

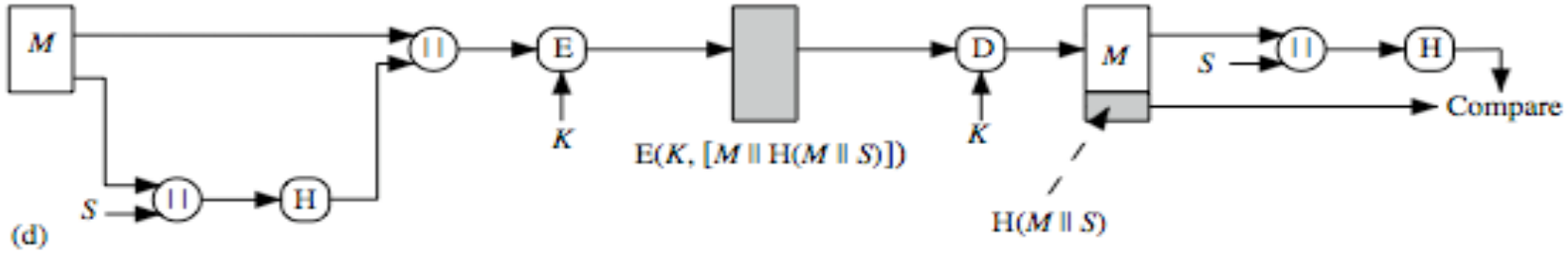


Message unencrypted: Authentication

Hash Function Usages (II)

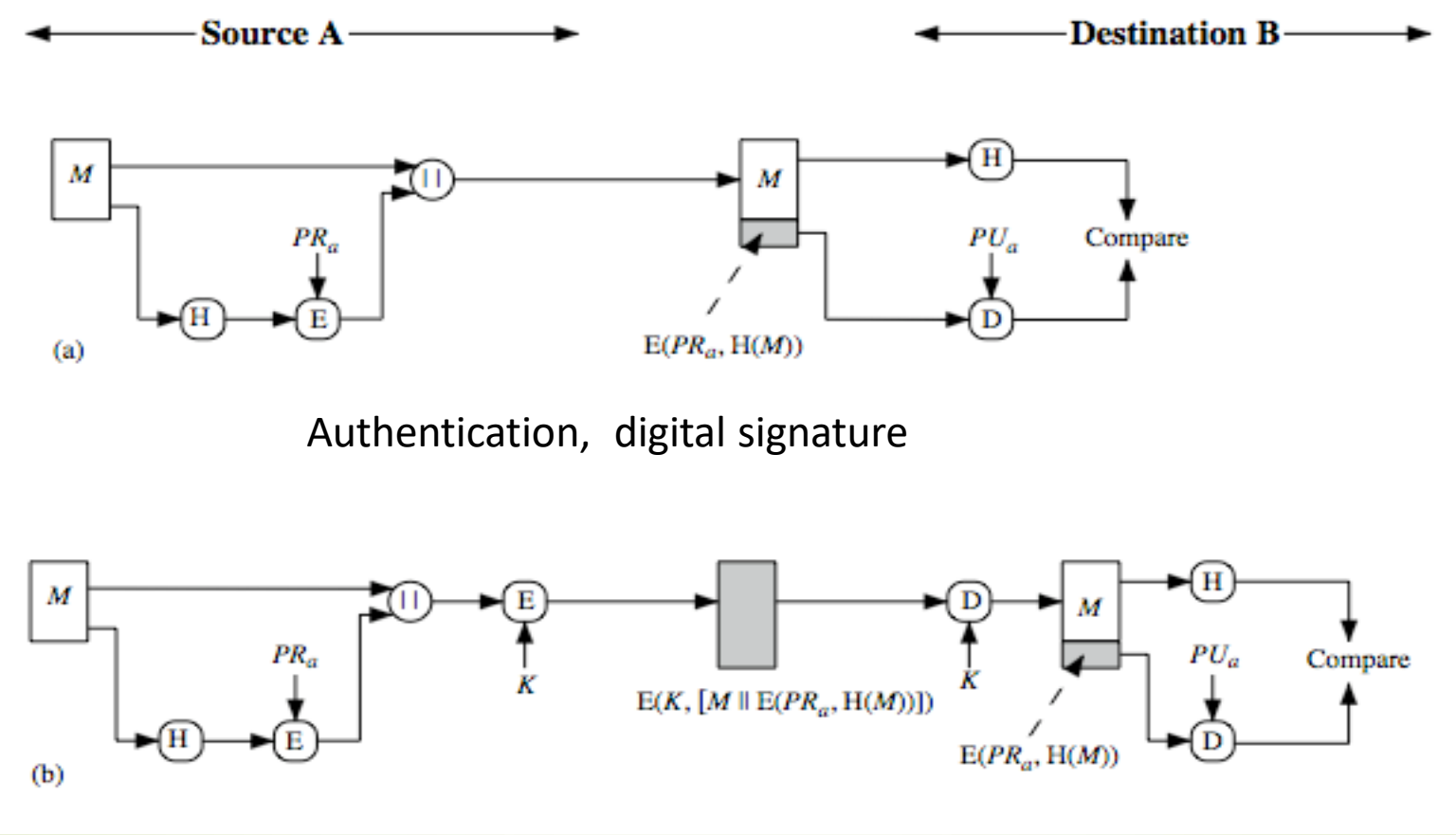


Message encrypted : Authentication (no encryption needed!)



Message unencrypted: Authentication, confidentiality

Hash Function Usages (III)



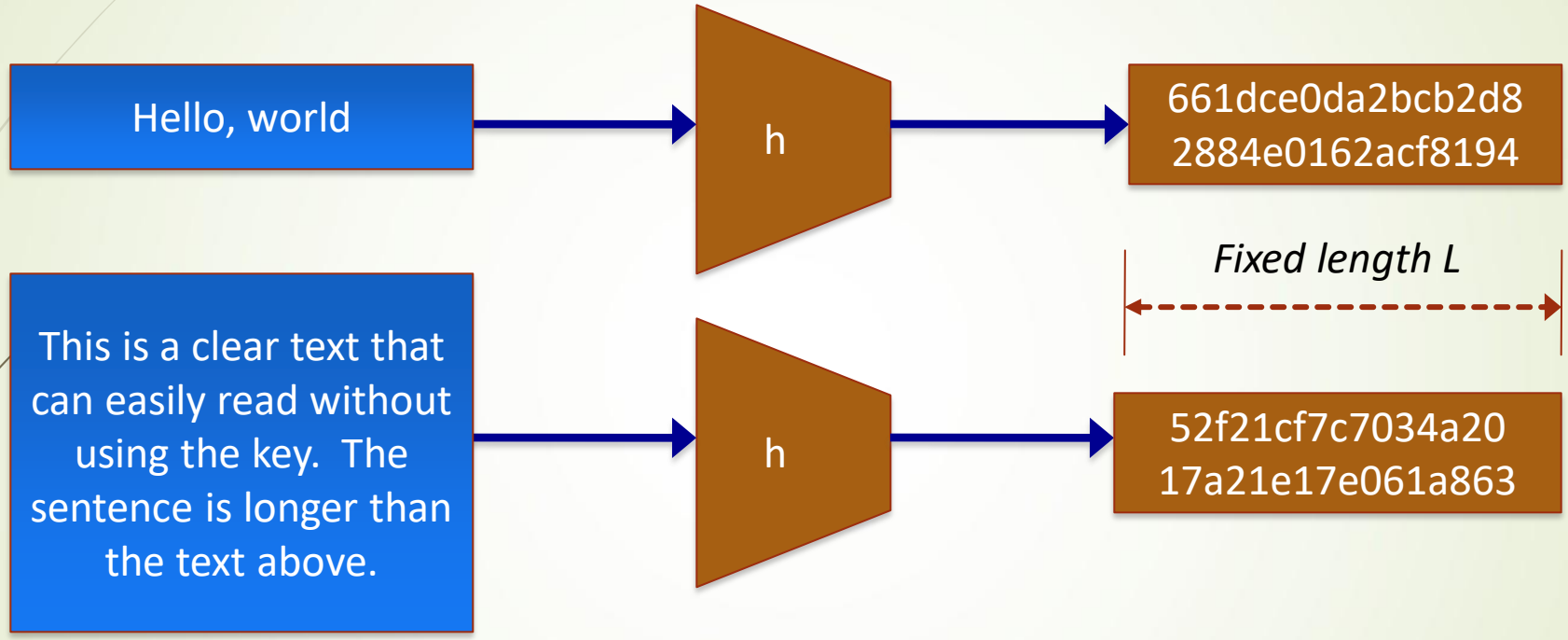
Authentication, digital signature

Authentication, digital signature, confidentiality

Hash Function Properties

- Arbitrary-length message to fixed-length digest
- Preimage resistant (**One-way property**)
- Second preimage resistant (**Weak collision resistant**)
- Collision resistant (**Strong collision resistance**)

Properties : Fixed length



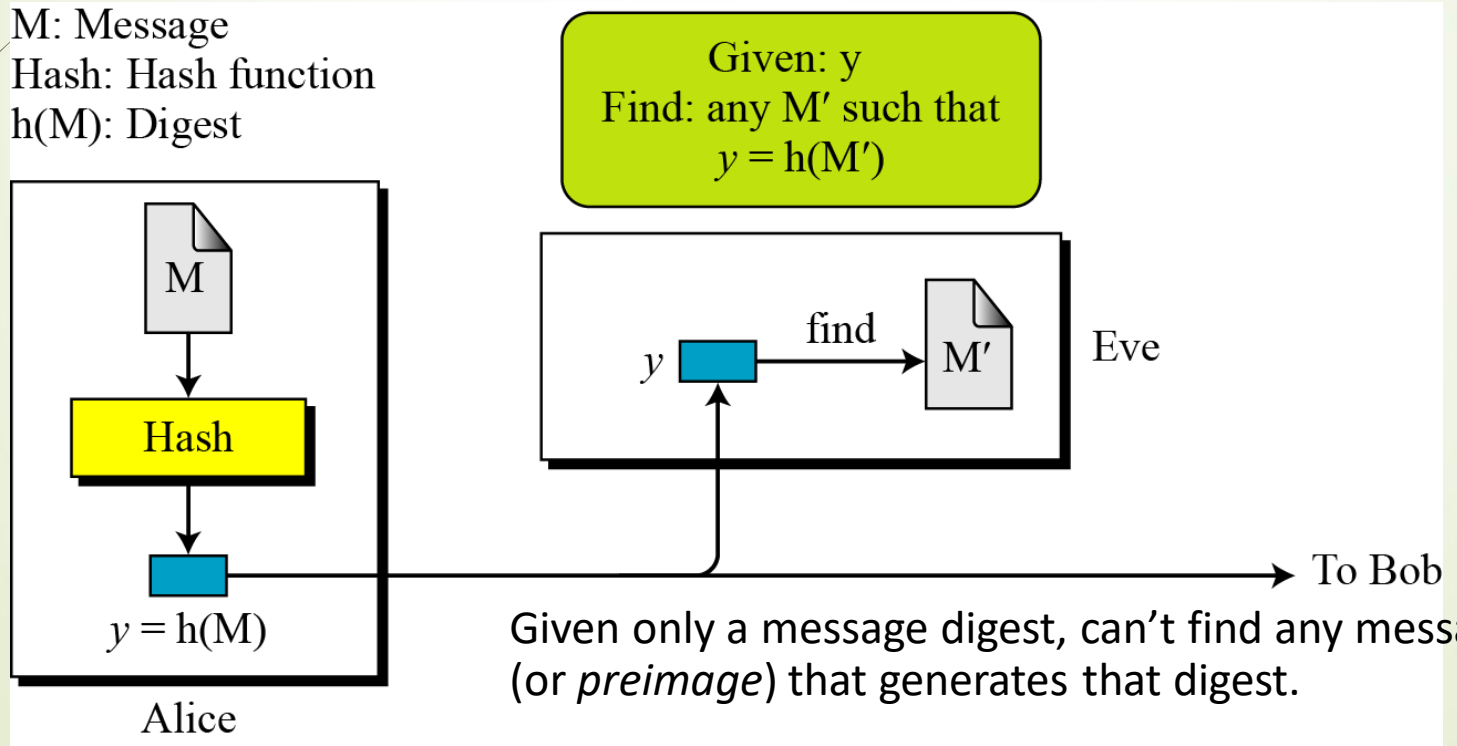
- Arbitrary-length message to fixed-length digest

Preimage resistant

- This measures how difficult to devise a message which hashes to the known digest
- Roughly speaking, the hash function must be one-way.

Preimage Attack

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



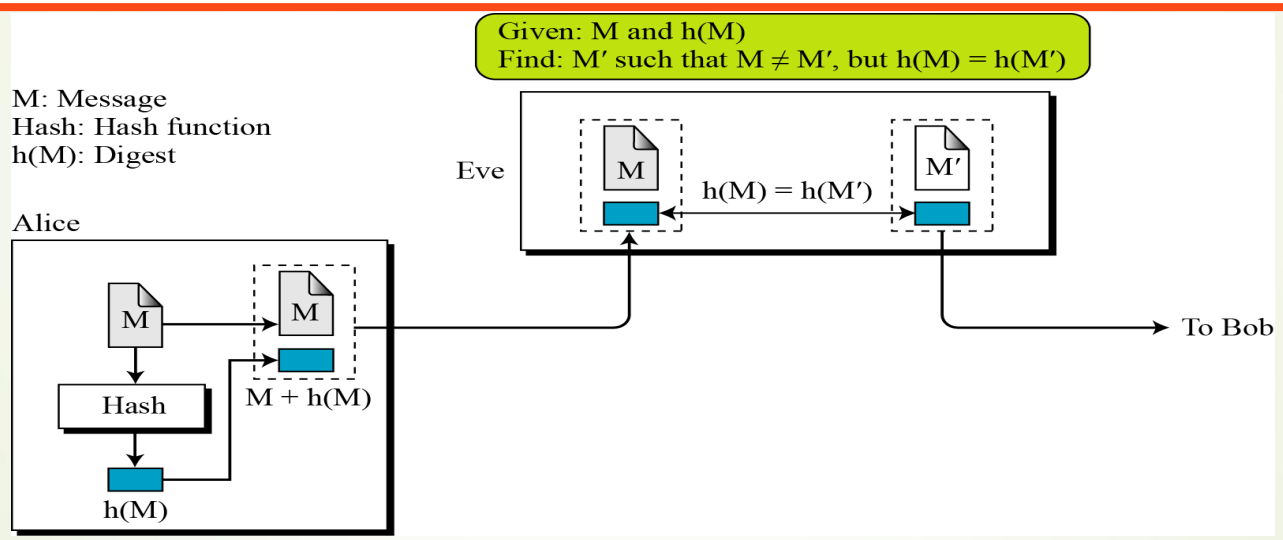
Given only a message digest, can't find any message (or *preimage*) that generates that digest.

Second preimage resistant

- This measures how difficult to devise a message which hashes to the known digest and its message

Second Preimage Attack

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



- Given one message, can't find another message that has the same message digest. An attack that finds a second message with the same message digest is a *second pre-image* attack.
  - It would be easy to forge new digital signatures from old signatures if the hash function used weren't second preimage resistant

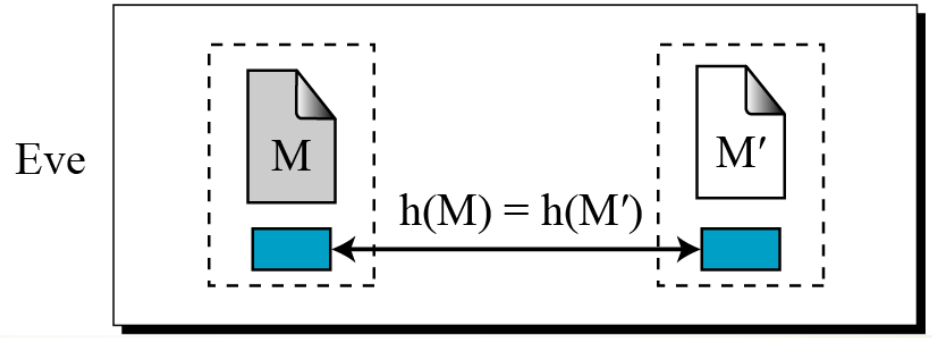
Collision Resistant

Collision Attack

Given: none **Find: $M' \neq M$ such that $h(M) = h(M')$**

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

Find: M and M′ such that $M \neq M'$, but $h(M) = h(M')$

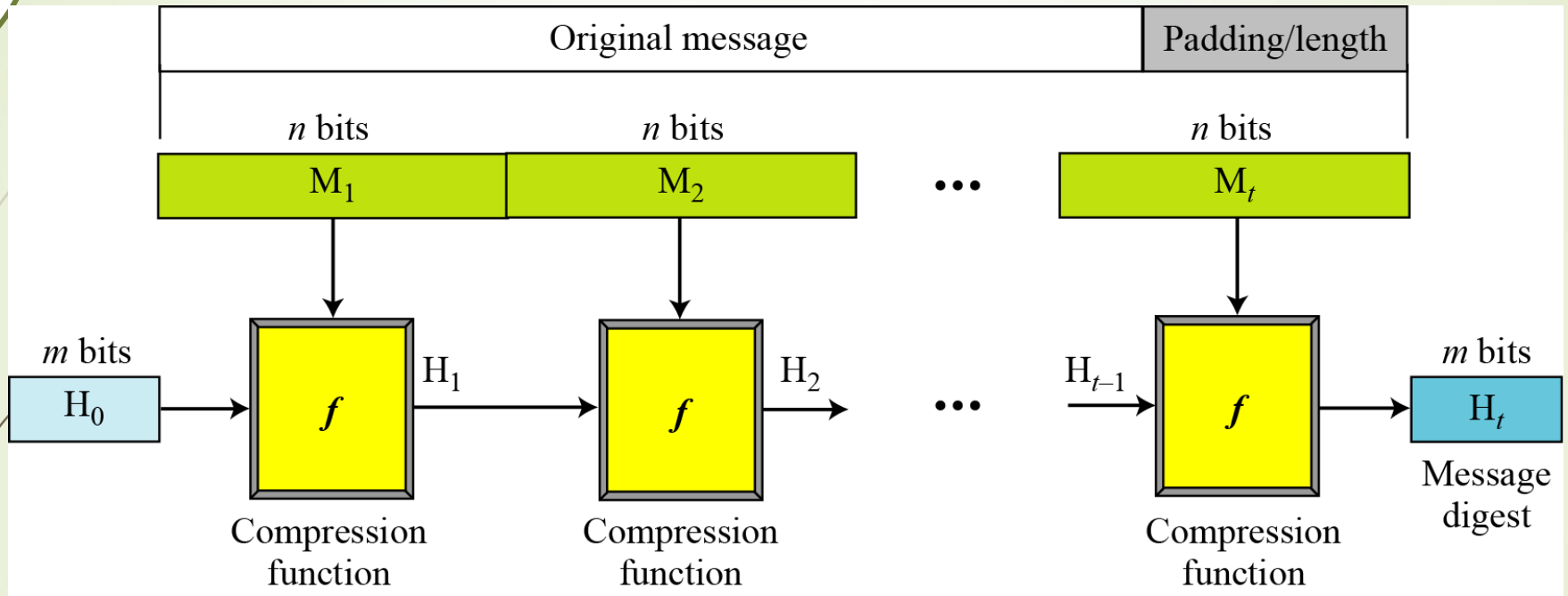


- Can't find any two different messages with the same message digest
  - Collision resistance implies second preimage resistance
  - Collisions, if we could find them, would give signatories a way to repudiate their signatures

Two Group of Compression Functions

- The compression function is made from scratch
  - Message Digest
- A symmetric-key block cipher serves as a compression function
  - Whirlpool

Merkle-Damgard Scheme



- Well-known method to build cryptographic hash function
- A message of arbitrary length is broken into blocks
  - length depends on the compression function *f*
  - padding the size of the message into a multiple of the block size.
  - sequentially process blocks , taking as input the result of the hash so far and the current message block, with the final fixed length output

Hash Functions Family

- **MD (Message Digest)**
  - Designed by Ron Rivest
  - Family: MD2, MD4, MD5
- **SHA (Secure Hash Algorithm)**
  - Designed by NIST
  - Family: SHA-0, SHA-1, and SHA-2
    - SHA-2: SHA-224, SHA-256, SHA-384, SHA-512
    - SHA-3: New standard in competition
- **RIPEMD (Race Integrity Primitive Evaluation Message Digest)**
  - Developed by Katholieke University Leuven Team
  - Family : RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320

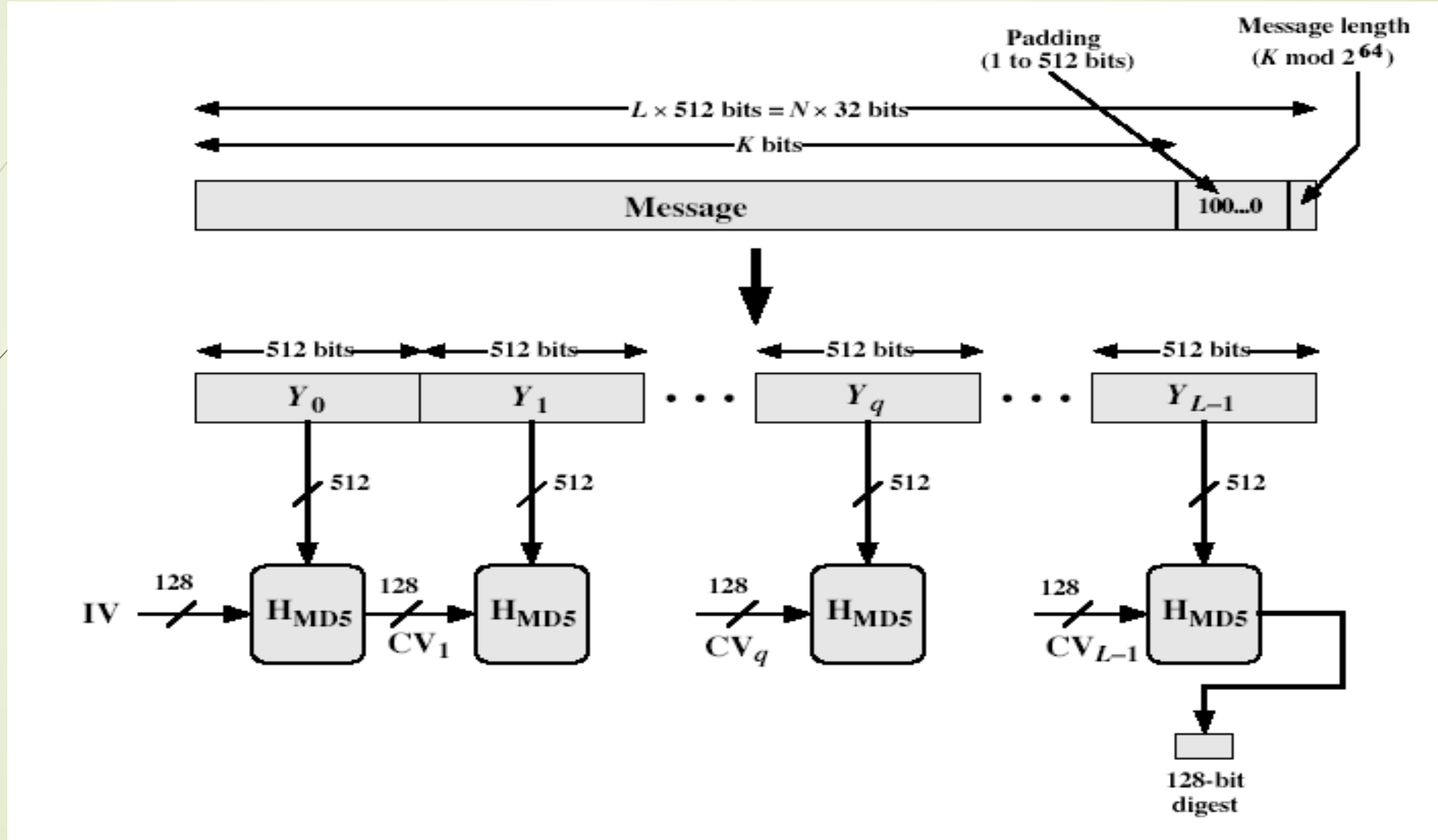
MD5, SHA-1, and RIPEMD-160

	MD5	SHA-1	RIPEMD-160
Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)	160 (5 paired rounds of 16)
Maximum message size	∞	$2^{64} - 1$ bits	$2^{64} - 1$ bits
Primitive logical functions	4	4	5
Additive constants used	64	4	9
Endianness	Little-endian	Big-endian	Little-endian

MD2, MD4 and MD5

- Family of one-way hash functions by Ronald Rivest
  - All produces 128 bits hash value
- **MD2: 1989**
  - Optimized for 8 bit computer
  - Collision found in 1995
- **MD4: 1990**
  - Full round collision attack found in 1995
- **MD5: 1992**
  - Specified as Internet standard in RFC 1321
  - since 1997 it was theoretically not so hard to create a collision
  - Practical Collision MD5 has been broken since 2004
  - CA attack published in 2007

MD5 Overview



MD5 Algorithm

- MD5 algorithm consists of **5 steps**:
- **Step 1. Appending Padding Bits.** The original message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. The padding rules are:
  - The original message is always padded with one bit "1" first.
  - Then zero or more bits "0" are padded to bring the length of the message up to 64 bits fewer than a multiple of 512.
- **Step 2. Appending Length.** 64 bits are appended to the end of the padded message to indicate the length of the original message in bytes. The rules of appending length are:
  - The length of the original message in bytes is converted to its binary format of 64 bits. If overflow happens, only the low-order 64 bits are used.
  - Break the 64-bit length into 2 words (32 bits each).
  - The low-order word is appended first and followed by the high-order word.

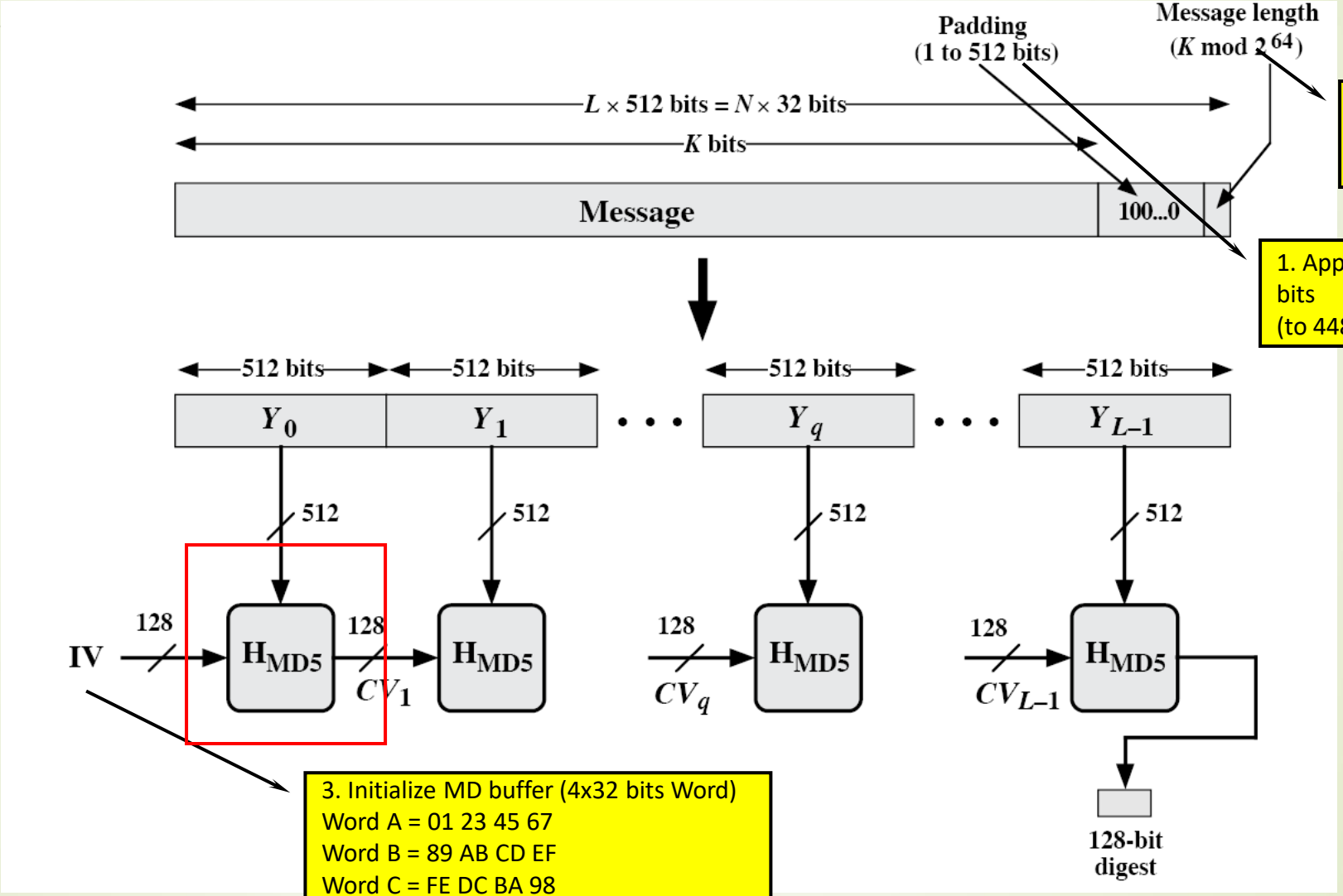
MD5 Algorithm

- **Step 3. Initializing MD Buffer.** MD5 algorithm requires a 128-bit buffer with a specific initial value. The rules of initializing buffer are:
  - The buffer is divided into 4 words (32 bits each), named as A, B, C, and D.
  - Word A is initialized to: 0x67452301.
  - Word B is initialized to: 0xEFCDAB89.
  - Word C is initialized to: 0x98BADCFE.
  - Word D is initialized to: 0x10325476.

Step 4. Processing Message in 512-bit Blocks. This is the main step of MD 5 algorithm, which loops through the padded and appended message in blocks of 512 bits each. For each input block, 4 rounds of operations are performed with 16 operations in each round.

Step 5. Output. The contents in buffer words A, B, C, D are returned in sequence with low-order byte first.

MD5 Overview



2. Append length (64bits)

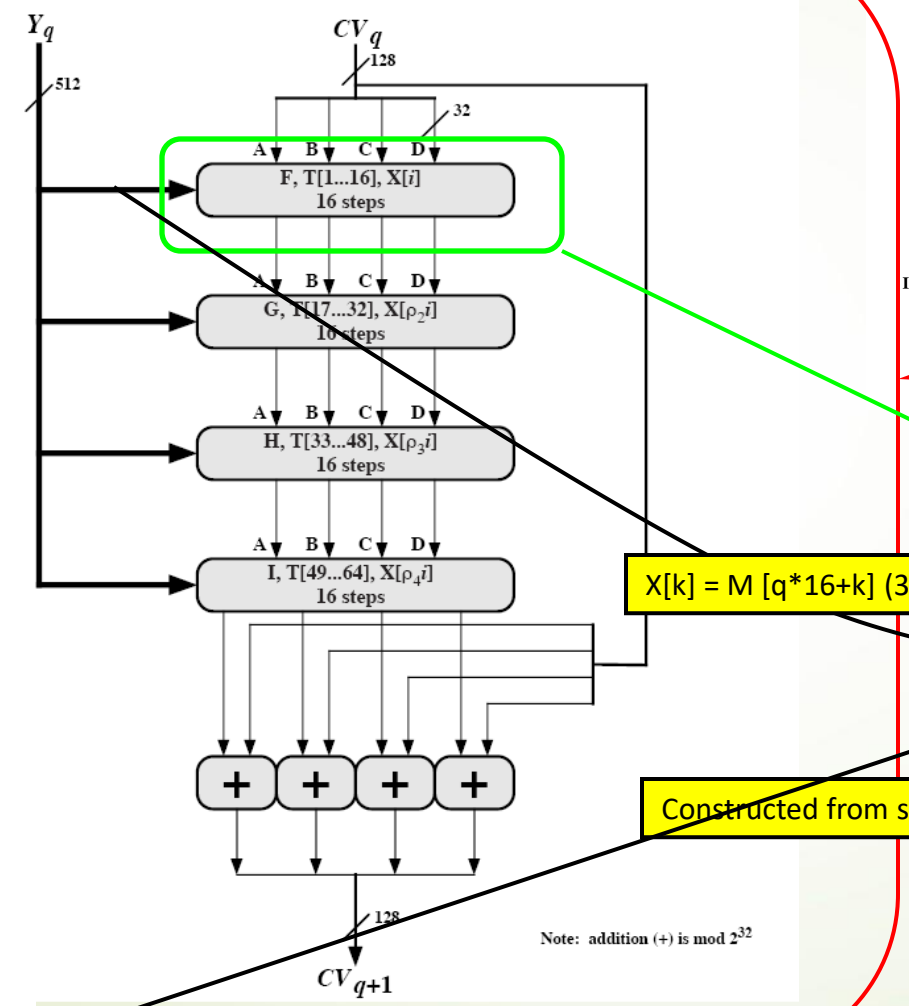
1. Append padding bits (to 448 mod 512)

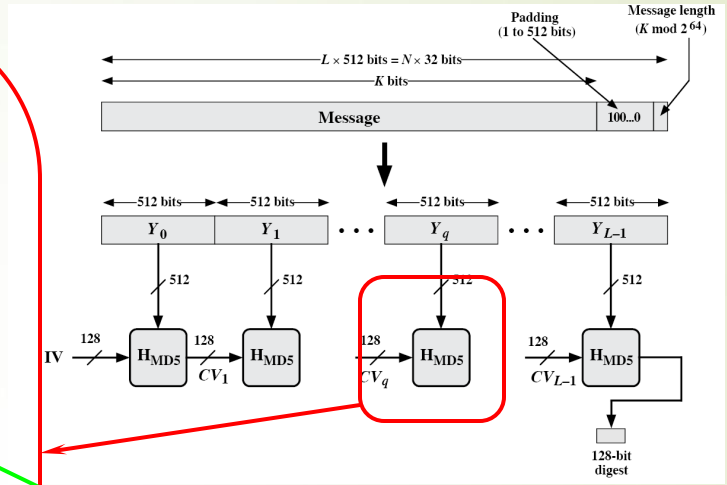
3. Initialize MD buffer (4x32 bits Word)
Word A = 01 23 45 67
Word B = 89 AB CD EF
Word C = FE DC BA 98
Word D = 76 54 32 10

Step 4 . Processing Message in 512-bit Blocks.

```
Input and predefined functions:
   A, B, C, D: initialized buffer words

   F(X,Y,Z) = (X AND Y ) OR (NOT X AND Z)
   G(X,Y,Z) = (X AND Z ) OR (Y AND NOT Z)
   H(X,Y,Z) = X XOR Y XOR Z
   I(X,Y,Z) = Y XOR (X OR NOT Z)

   T[1, 2, ..., 64]: Array of special constants (32-bit integers) as:
      T[i] = int(abs(sin(i)) * 2**32)

   M[1, 2, ..., N]: Blocks of the padded and appended message

   R1(a,b,c,d,X,s,i): Round 1 operation defined as:
      a = b + ((a + F(b,c,d) + X + T[i]) <<< s)

   R2(a,b,c,d,X,s,i): Round 1 operation defined as:
      a = b + ((a + G(b,c,d) + X + T[i]) <<< s)

   R3(a,b,c,d,X,s,i): Round 1 operation defined as:
      a = b + ((a + H(b,c,d) + X + T[i]) <<< s)

   R4(a,b,c,d,X,s,i): Round 1 operation defined as:
      a = b + ((a + I(b,c,d) + X + T[i]) <<< s)
```

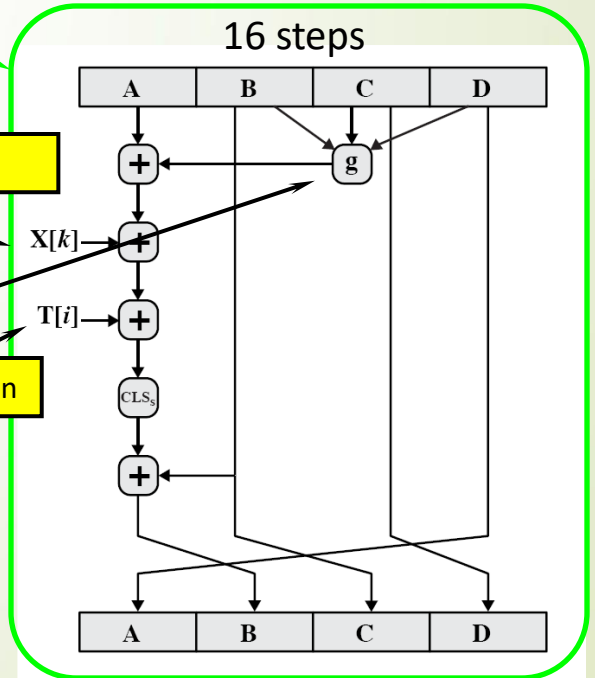
Hash Algorithm Design – MD5





X[k] = M [q*16+k] (32 bit)

Constructed from sine function

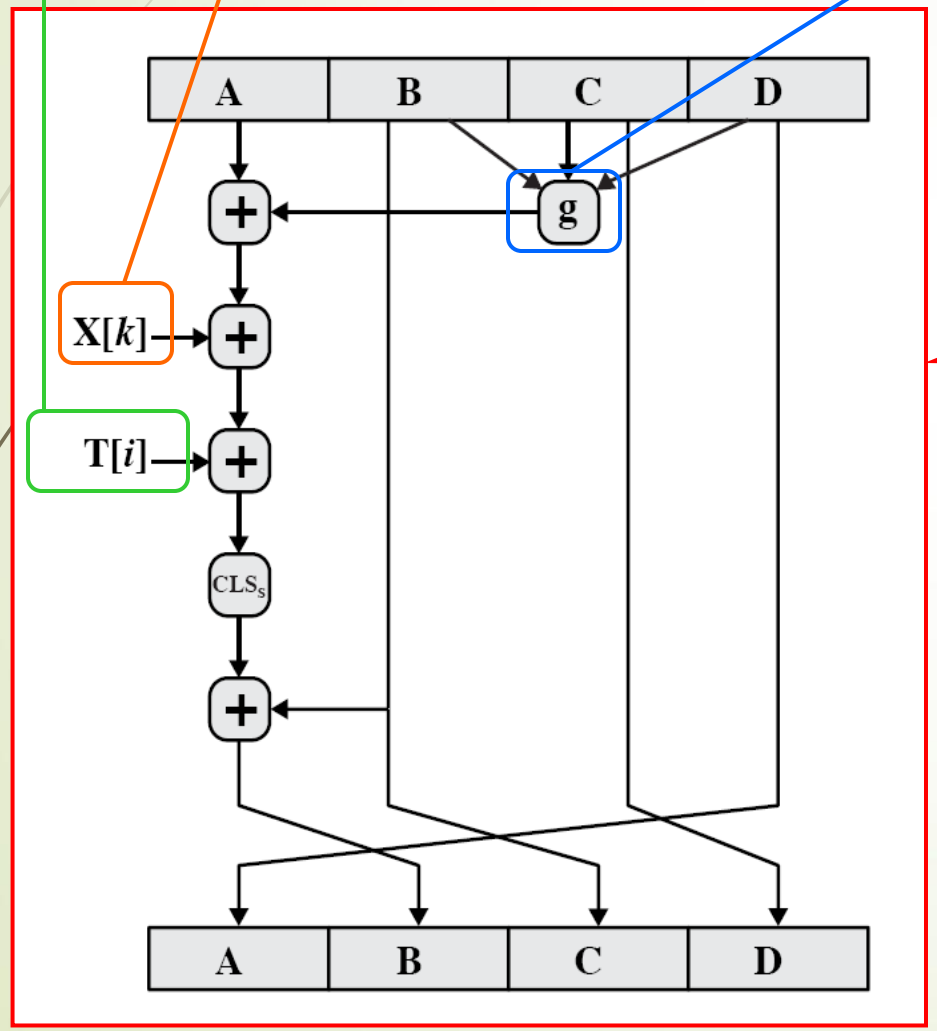


b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

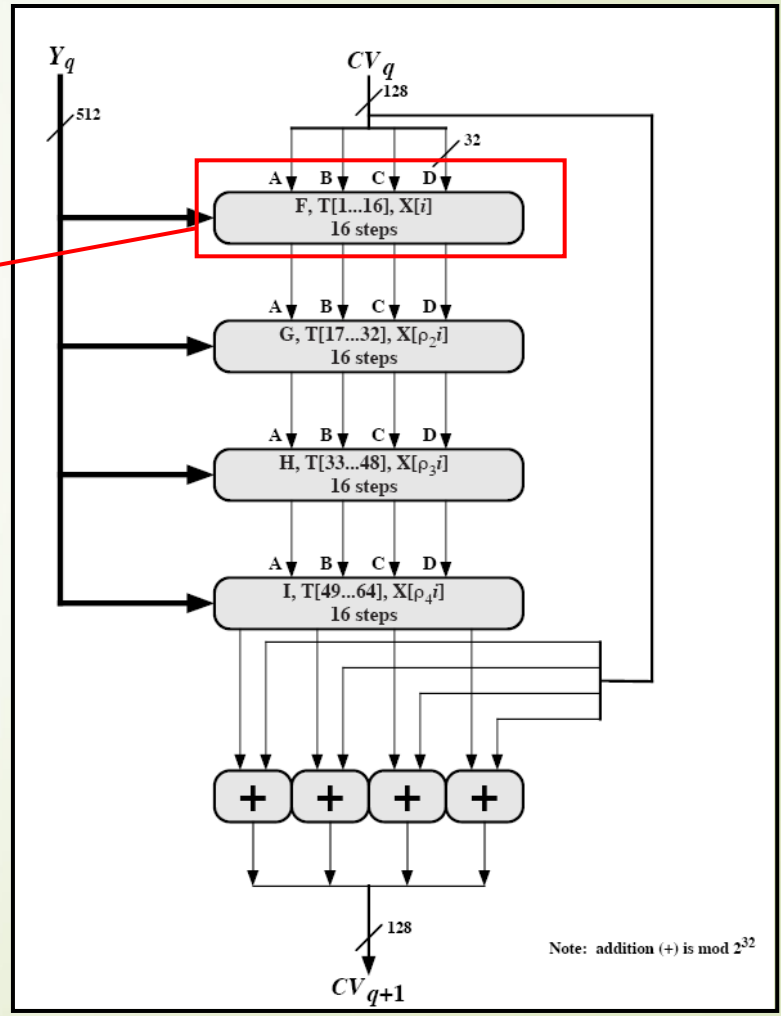
The ith 32-bit word in matrix T, constructed from the sine function

M [q*16+k] = the kth 32-bit word from the qth 512-bit block of the msg

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$



Single step



Secure Hash Algorithm

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

Revised SHA

- 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

SHA Versions

	MD5	SHA-0	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Digest size	128	160	160	224	256	384	512
Message size	$2^{64}-1$	$2^{64}-1$	$2^{64}-1$	$2^{64}-1$	$2^{64}-1$	$2^{128}-1$	$2^{128}-1$
Block size	512	512	512	512	512	1024	1024
Word size	32	32	32	32	32	64	64
# of steps	64	64	80	64	64	80	80

Full collision found

SHA 1 Algorithm

- **SHA1 (Secure Hash Algorithm 1) is message-digest algorithm, which takes an input message of any length < 2^64 bits and produces a 160-bit output as the message digest.**
- SHA1 algorithm consists of **6 tasks:**
- **Task 1. Appending Padding Bits.** The original message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. The padding rules are:
  - The original message is always padded with one bit "1" first.
  - Then zero or more bits "0" are padded to bring the length of the message up to 64 bits fewer than a multiple of 512.
- **Task 2. Appending Length.** 64 bits are appended to the end of the padded message to indicate the length of the original message in bytes. The rules of appending length are:
  - The length of the original message in bytes is converted to its binary format of 64 bits. If overflow happens, only the low-order 64 bits are used.
  - Break the 64-bit length into 2 words (32 bits each).
  - The low-order word is appended first and followed by the high-order word.

SHA 1 Algorithm

- **Task 3. Preparing Processing Functions**. SHA1 requires 80 processing functions defined as:

```
f(t;B,C,D) = (B AND C) OR ((NOT B) AND D)          ( 0 <= t <= 19)
f(t;B,C,D) = B XOR C XOR D                         (20 <= t <= 39)
f(t;B,C,D) = (B AND C) OR (B AND D) OR (C AND D)   (40 <= t <= 59)
f(t;B,C,D) = B XOR C XOR D                         (60 <= t <= 79)
```

- **Task 4. Preparing Processing Constants.** SHA1 requires 80 processing constant words defined as:

```
K(t) = 0x5A827999          ( 0 <= t <= 19)
K(t) = 0x6ED9EBA1          (20 <= t <= 39)
K(t) = 0x8F1BBCDC          (40 <= t <= 59)
K(t) = 0xCA62C1D6          (60 <= t <= 79)
```

- **Task 5. Initializing Buffers**. SHA1 algorithm requires 5 word buffers with the following initial values:

```
H0 = 0x67452301
H1 = 0xEFCDAB89
H2 = 0x98BADCFE
H3 = 0x10325476
H4 = 0xC3D2E1F0
```

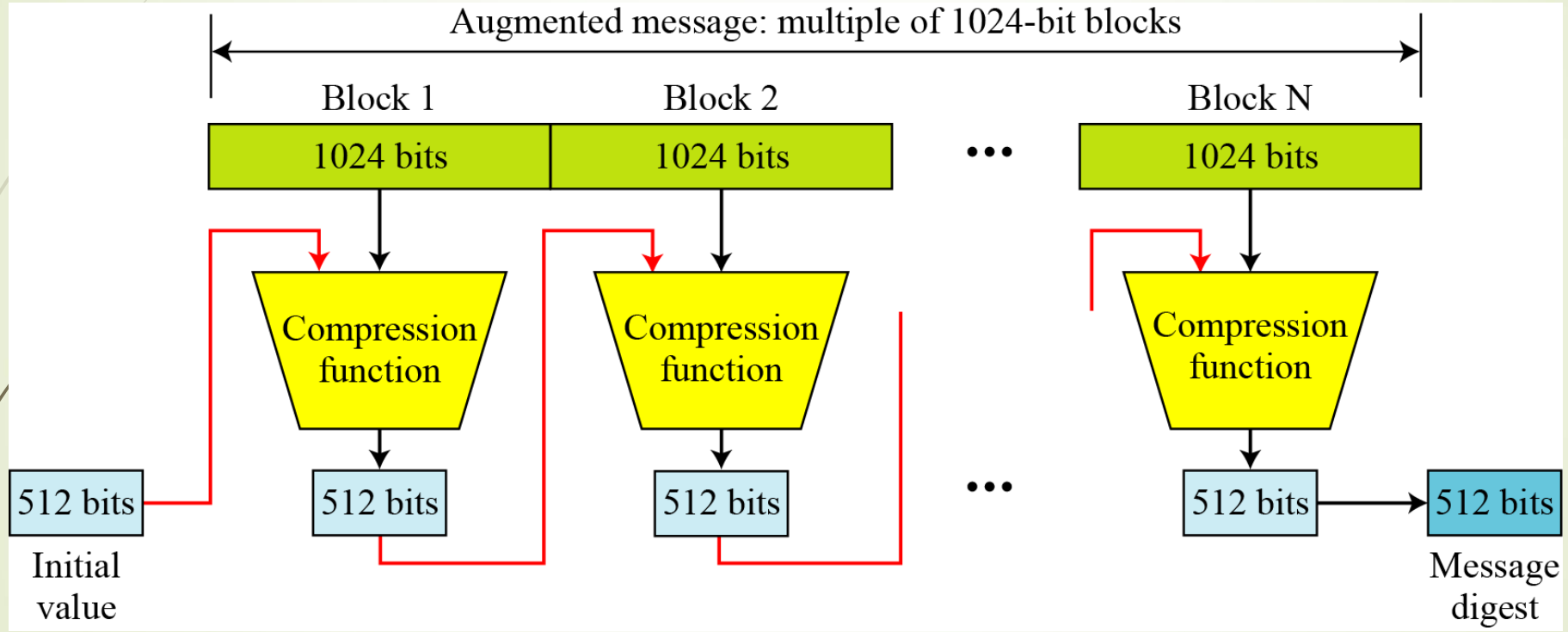
SHA 1 Algorithm

- **Task 6. Processing Message in 512-bit Blocks.** This is the main task of SHA1 algorithm, which loops through the padded and appended message in blocks of 512 bits each. For each input block, a number of operations are performed.

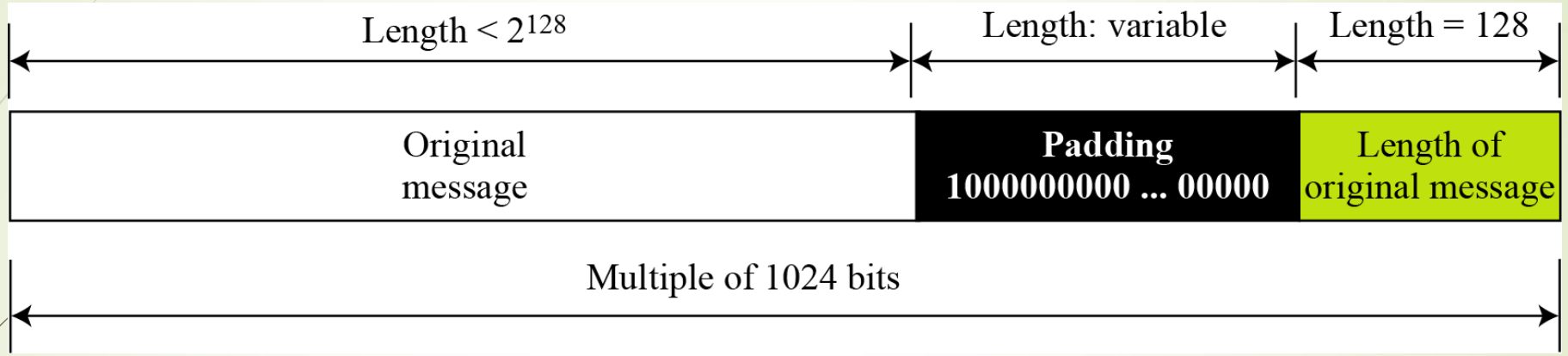
```
Input and predefined functions:
   M[1, 2, ..., N]: Blocks of the padded and appended message
   f(0;B,C,D), f(1,B,C,D), ..., f(79,B,C,D): Defined as above
   K(0), K(1), ..., K(79): Defined as above
   H0, H1, H2, H3, H4, H5: Word buffers with initial values

Algorithm:
   For loop on k = 1 to N

     (W(0),W(1),...,W(15)) = M[k] /* Divide M[k] into 16 words */

     For t = 16 to 79 do:
         W(t) = (W(t-3) XOR W(t-8) XOR W(t-14) XOR W(t-16)) <<< 1

     A = H0, B = H1, C = H2, D = H3, E = H4

     For t = 0 to 79 do:
         TEMP = A<<<5 + f(t;B,C,D) + E + W(t) + K(t)
         E = D, D = C, C = B<<<30, B = A, A = TEMP
     End of for loop

     H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4 + E
   End of for loop

Output:
   H0, H1, H2, H3, H4, H5: Word buffers with final message digest
```

- **Output.** The contents in H0, H1, H2, H3, H4, H5 are returned in sequence the message digest.

SHA-512 Overview



Padding and length field in SHA-512

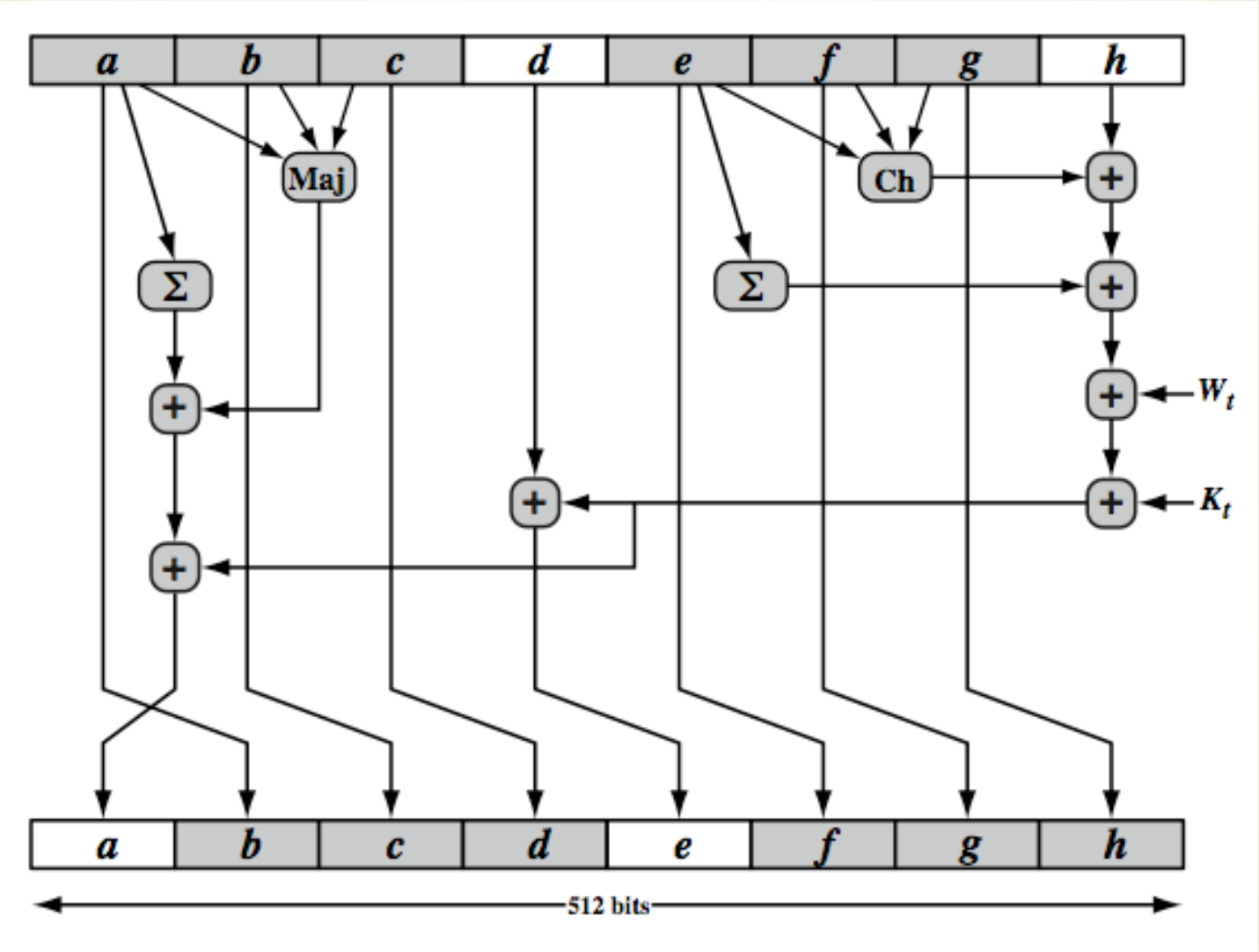


- **What is the number of padding bits if the length of the original message is 2590 bits?**
- We can calculate the number of padding bits as follows:

$$|P| = (-2590 - 128) \bmod 1024 = -2718 \bmod 1024 = 354$$

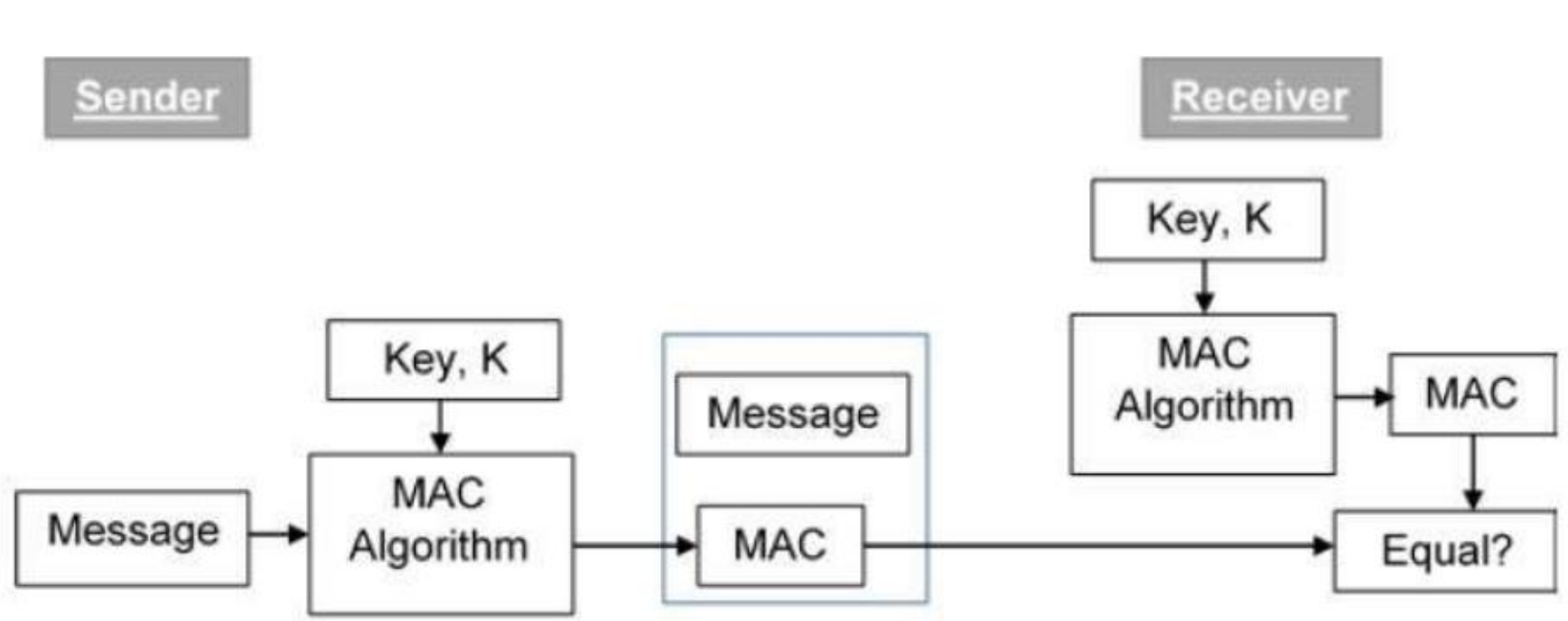
- The padding consists of one 1 followed by 353 0's.

SHA-512 Round Function



Message Authentication Code

- 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.



Message Authentication Code

- The sender uses some publicly known 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 major difference between hash and MAC is that **MAC uses secret key during the compression**.
- The sender forwards the message along with the MAC. Here, we assume that the message is sent in the clear, as we are concerned of providing message origin authentication, not confidentiality. If confidentiality is required then the message needs encryption.
- 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.

Limitations of MAC

- **Establishment of Shared Secret**
- **Inability to Provide Non-Repudiation**

HMAC - Hashed or Hash based MAC

- specified as Internet standard RFC2104
- uses hash function on the message:
  - $HMAC_K$ = Hash[(K^+ XOR opad) ||
  - Hash[(K^+ XOR ipad)||M)]]

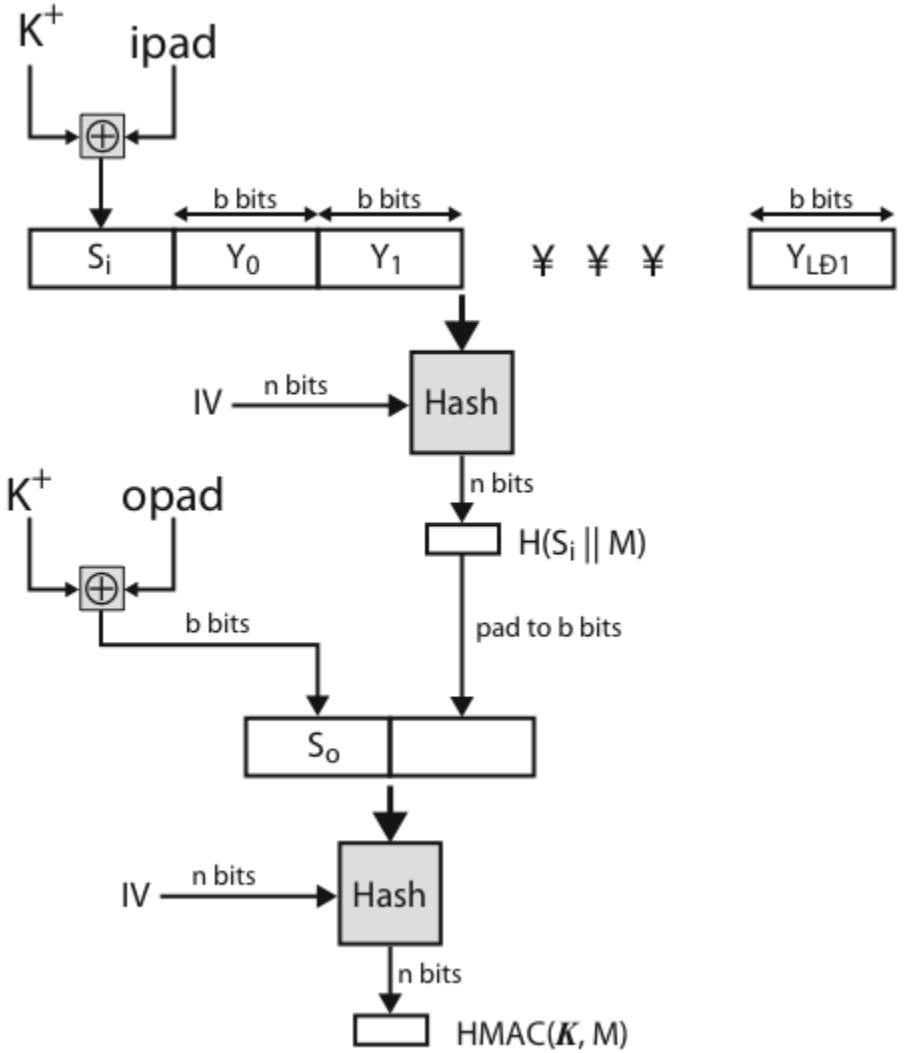
  where K^+ is the key padded out to size

  and opad, ipad are specified padding constants
- overhead is just 3 more hash calculations than the message needs alone
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC Algorithm

The working of HMAC starts with

- taking a message M containing blocks of length *b* bits.
- An input signature is padded to the left of the message and
- the whole is given as input to a hash function which gives us a temporary message digest MD'.
- MD' again is appended to an output signature and
- the whole is applied a hash function again, the result is our final message digest MD.



HMAC Algorithm

Here, H stands for Hashing function,

M is original message

Si and So are input and output signatures respectively,

Yi is the ith block in original message M, where i ranges from [1, L)

L = the count of blocks in M

K is the secret key used for hashing

IV is an initial vector (some constant)

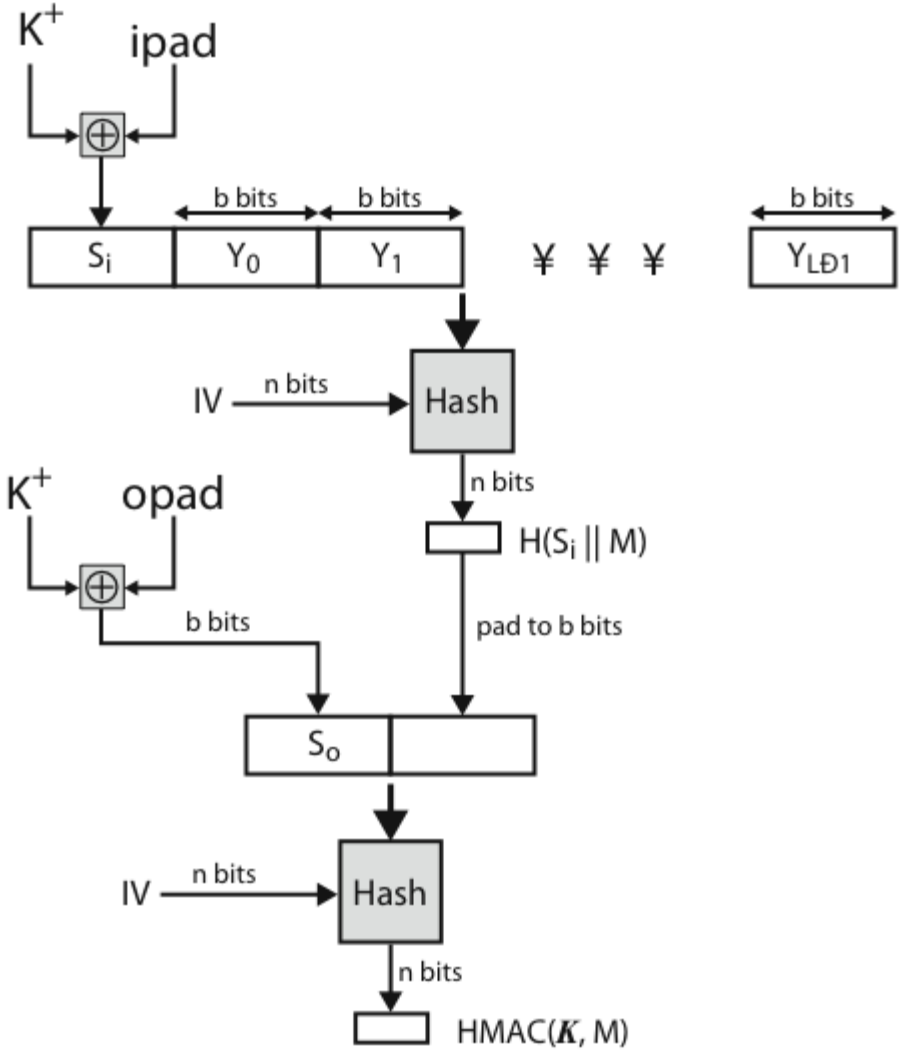
The generation of input signature and output signature *Si* and *So* respectively.

$S_i = K^+ \oplus ipad$ 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 Security

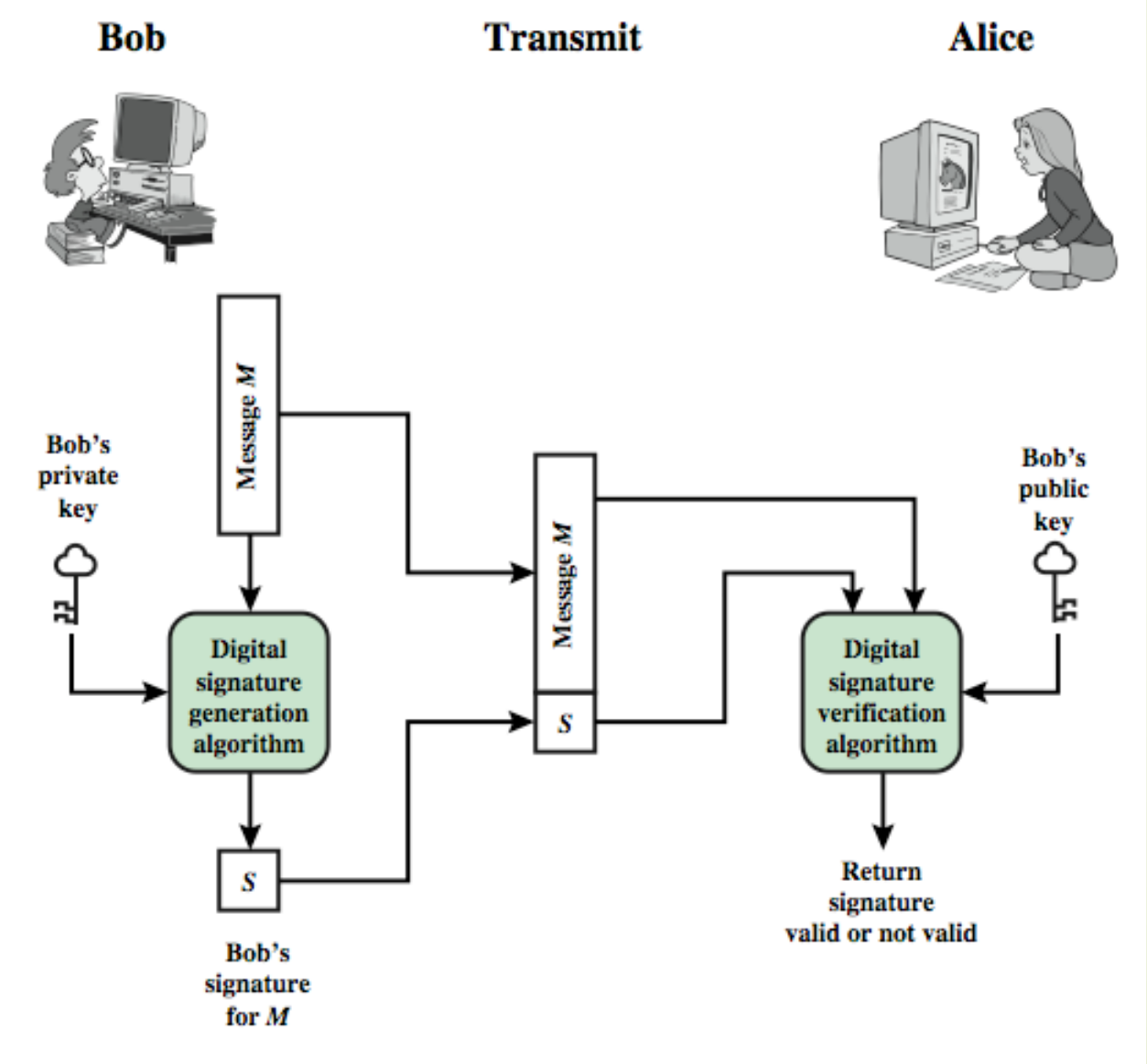
- proved security of HMAC relates to that of the underlying hash algorithm
- attacking HMAC requires either:
  - brute force attack on key used
  - birthday attack (but since keyed would need to observe a very large number of messages)
- choose hash function used based on speed verses security constraints

DIGITAL SIGNATURES

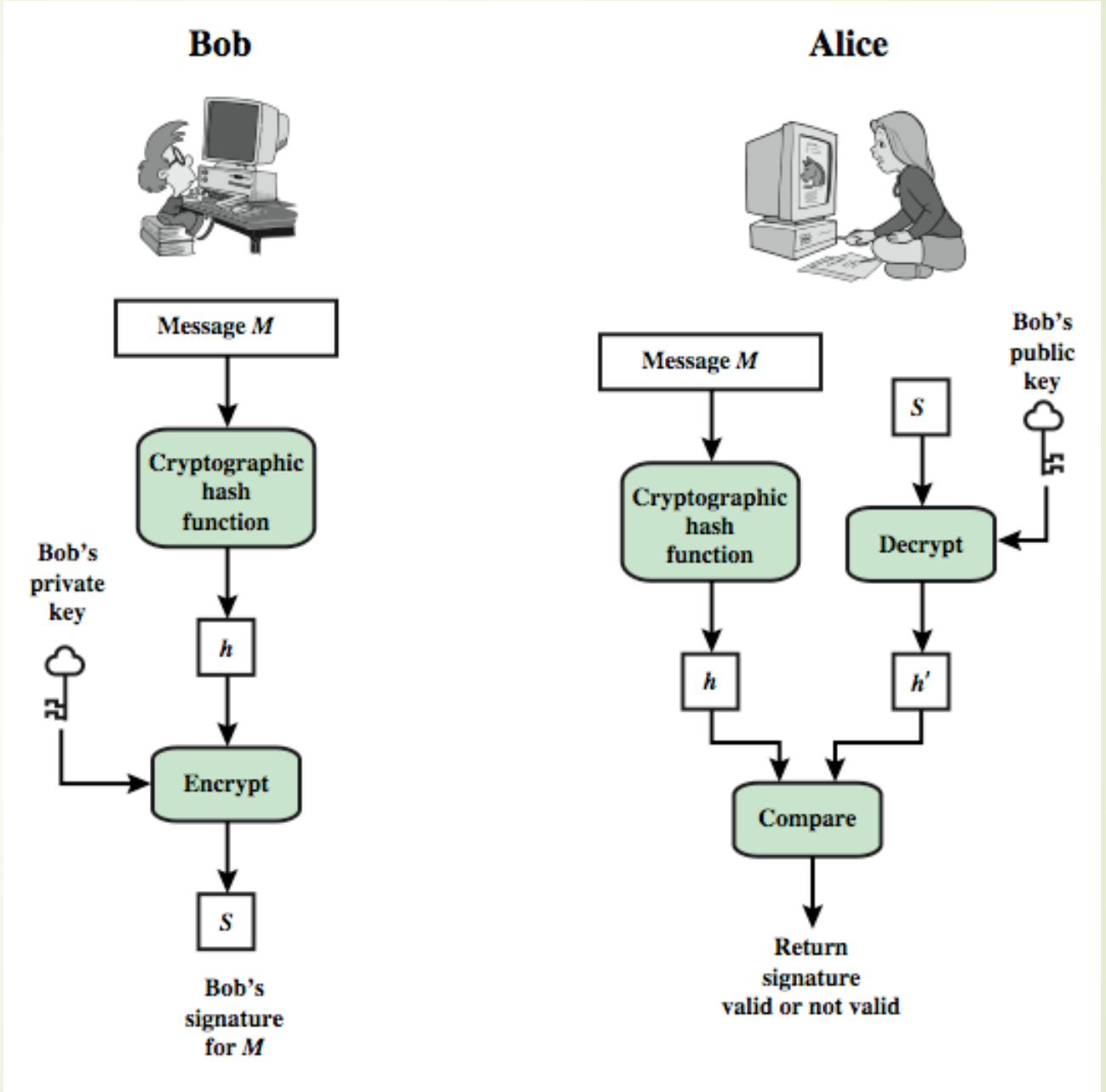
Digital Signatures

- have looked at message authentication using hash functions
  - but does not address issues of lack of trust
- digital signatures provide the ability to:
  - verify author, date & time of signature
  - authenticate message contents
  - be verified by third parties to resolve disputes
- hence include authentication function with additional capabilities

Digital Signature Model



Digital Signature Model



Digital Signature Properties

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

Direct Digital Signatures

- involve only sender & receiver
- assumed receiver has sender's public-key
- digital signature made by sender signing entire message or hash with private-key
- can encrypt using receiver's public-key
- important that sign first then encrypt message & signature
- security depends on sender's private-key

Arbitrated Digital Signatures

- involves use of arbiter A
  - validates any signed message
  - then dated and sent to recipient
- requires suitable level of trust in arbiter
- can be implemented with either private or public-key algorithms
- arbiter may or may not see message


SIST
JUSTICE PEACE REVOLUTION
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
DEEMED UNIVERSITY

Kerberos

Kerberos

- a **key distribution and user authentication service** developed at MIT
- provides centralised private-key third-party authentication in a distributed network
  - allows users access to services distributed through network
  - without needing to trust all workstations
  - rather all trust a central authentication server
- two versions in use: 4 & 5

Kerberos Requirements

- its first report identified requirements as:
  - secure
  - reliable
  - transparent
  - scalable
- implemented using an authentication protocol based on Needham-Schroeder

Kerberos v4 Overview

- a basic third-party authentication scheme
- have an Authentication Server (AS)
  - users initially negotiate with AS to identify self
  - AS provides a non-corruptible authentication credential (ticket granting ticket TGT)
- have a Ticket Granting server (TGS)
  - users subsequently request access to other services from TGS on basis of users TGT

Kerberos v4 Dialogue – Simple authentication

- obtain ticket granting ticket from AS
  - once per session
- obtain service granting ticket from TGT
  - for each distinct service required
- client/server exchange to obtain service
  - on every service request

$$\textbf{(1)}\ \mathrm{C} \rightarrow \mathrm{AS}: \quad ID_\mathrm{C} \| P_\mathrm{C} \| ID_\mathrm{V}$$

$$\textbf{(2)}\ \mathrm{AS} \rightarrow \mathrm{C}: \quad Ticket$$

$$\textbf{(3)}\ \mathrm{C} \rightarrow \mathrm{V}: \quad ID_\mathrm{C} \| Ticket$$

$$Ticket = \mathrm{E}(K_v, [ID_\mathrm{C} \| AD_\mathrm{C} \| ID_\mathrm{V}])$$

where

C	= client
AS	= authentication server
V	= server
ID_C	= identifier of user on C
ID_V	= identifier of V
P_C	= password of user on C
AD_C	= network address of C
K_v	= secret encryption key shared by AS and V

Kerberos v4 Dialogue – Secure authentication

Once per user logon session:

(1) $C \rightarrow AS: \quad ID_C \| ID_{tgs}$

(2) $AS \rightarrow C: \quad E(K_c, Ticket_{tgs})$

Once per type of service:

(3) $C \rightarrow TGS: \quad ID_C \| ID_V \| Ticket_{tgs}$

(4) $TGS \rightarrow C: \quad Ticket_v$

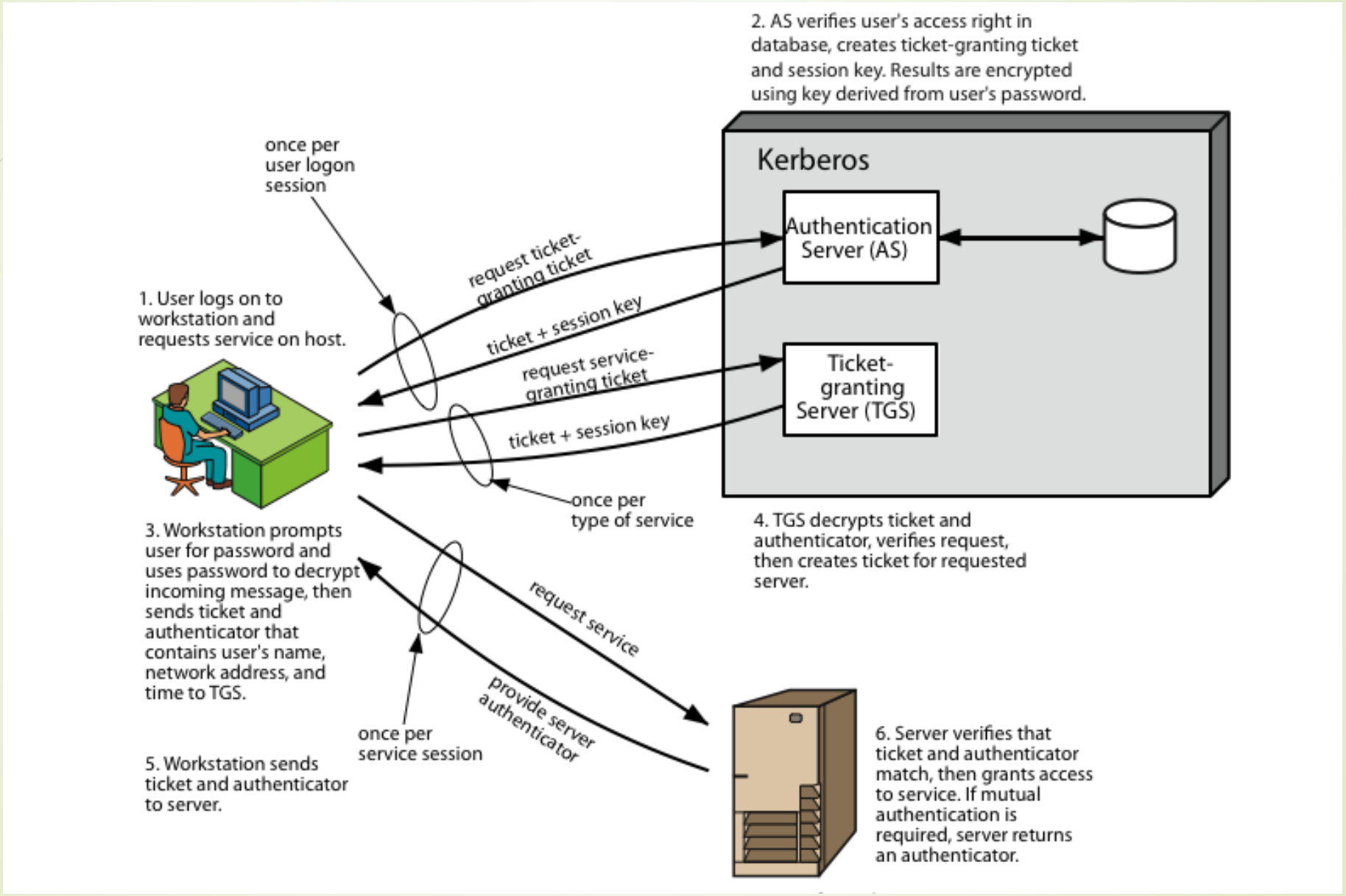
Once per service session:

(5) $C \rightarrow V: \quad ID_C \| Ticket_v$

$$Ticket_{tgs} = E(K_{tgs}, [ID_C \| AD_C \| ID_{tgs} \| TS_1 \| Lifetime_1])$$

$$Ticket_v = E(K_v, [ID_C \| AD_C \| ID_v \| TS_2 \| Lifetime_2])$$

Kerberos 4 Overview



Kerberos v4 Dialogue

Table 4.1 Summary of Kerberos Version 4 Message Exchanges

(1) C → AS $ID_c \| ID_{tgs} \| TS_1$

(2) AS → C $E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $ID_v \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,tgs}, [ID_C \| AD_C \| TS_3])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Ticket_v \| Authenticator_c$

(6) V → C $E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,v}, [ID_C \| AD_C \| TS_5])$$

(c) Client/Server Authentication Exchange to obtain service

Kerberos v4 Dialogue

Table 4.2 Rationale for the Elements of the Kerberos Version 4 Protocol

Message (1)	Client requests ticket-granting ticket.
ID_C	Tells AS identity of user from this client.
ID_{tgs}	Tells AS that user requests access to TGS.
TS_1	Allows AS to verify that client's clock is synchronized with that of AS.
Message (2)	AS returns ticket-granting ticket.
K_c	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2).
$K_{c,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key.
ID_{tgs}	Confirms that this ticket is for the TGS.
TS_2	Informs client of time this ticket was issued.
$Lifetime_2$	Informs client of the lifetime of this ticket.
$Ticket_{tgs}$	Ticket to be used by client to access TGS.

(a) Authentication Service Exchange

Kerberos v4 Dialogue

Message (3)	Client requests service-granting ticket.
ID_V	Tells TGS that user requests access to server V.
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (4)	TGS returns service-granting ticket.
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4).
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.
ID_V	Confirms that this ticket is for server V.
TS_4	Informs client of time this ticket was issued.
$Ticket_V$	Ticket to be used by client to access server V.
$Ticket_{tgs}$	Reusable so that user does not have to reenter password.
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering.
$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_{tgs}	Assures server that it has decrypted ticket properly.
TS_2	Informs TGS of time this ticket was issued.
$Lifetime_2$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay.
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_C	Must match address in ticket to authenticate ticket.
TS_3	Informs TGS of time this authenticator was generated.

(b) Ticket-Granting Service Exchange

Kerberos v4 Dialogue

Message (5)	Client requests service.
$Ticket_V$	Assures server that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (6)	Optional authentication of server to client.
$K_{c,v}$	Assures C that this message is from V.
$TS_5 + 1$	Assures C that this is not a replay of an old reply.
$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server.
K_v	Ticket is encrypted with key known only to TGS and server, to prevent tampering.
$K_{c,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_V	Assures server that it has decrypted ticket properly.
TS_4	Informs server of time this ticket was issued.
$Lifetime_4$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay.
$K_{c,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_c	Must match address in ticket to authenticate ticket.
TS_5	Informs server of time this authenticator was generated.

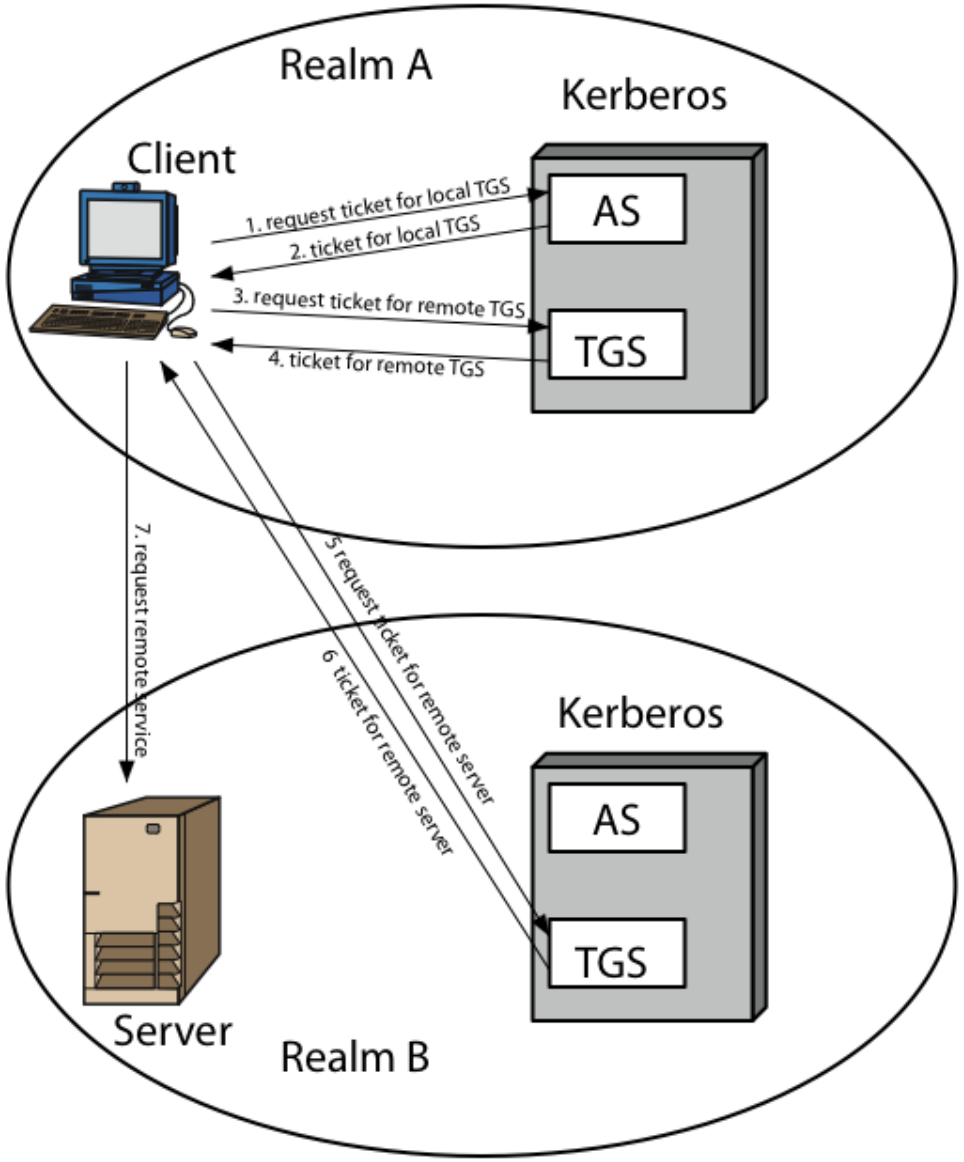
(c) Client/Server Authentication Exchange

Kerberos Realms

A Kerberos realm is a set of managed nodes that share the same Kerberos database

- a Kerberos environment consists of:
  - a Kerberos server
  - a number of clients, all registered with server
  - application servers, sharing keys with server
- this is termed a realm
  - typically a single administrative domain
- if have multiple realms, their Kerberos servers must share keys and trust

Kerberos Realms



(1) C → AS: $ID_C \| ID_{tgs} \| TS_1$

(2) AS → C: $E(K_C, [K_{C,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

(3) C → TGS: $ID_{tgsrem} \| Ticket_{tgs} \| Authenticator_C$

(4) TGS → C: $E(K_{C,tgs}, [K_{C,tgsrem} \| ID_{tgsrem} \| TS_4 \| Ticket_{tgsrem}])$

(5) C → TGS_{rem}: $ID_{Vrem} \| Ticket_{tgsrem} \| Authenticator_C$

(6) TGS_{rem} → C: $E(K_{C,tgsrem}, [K_{C,Vrem} \| ID_{Vrem} \| TS_6 \| Ticket_{Vrem}])$

(7) C → V_{rem}: $Ticket_{Vrem} \| Authenticator_C$

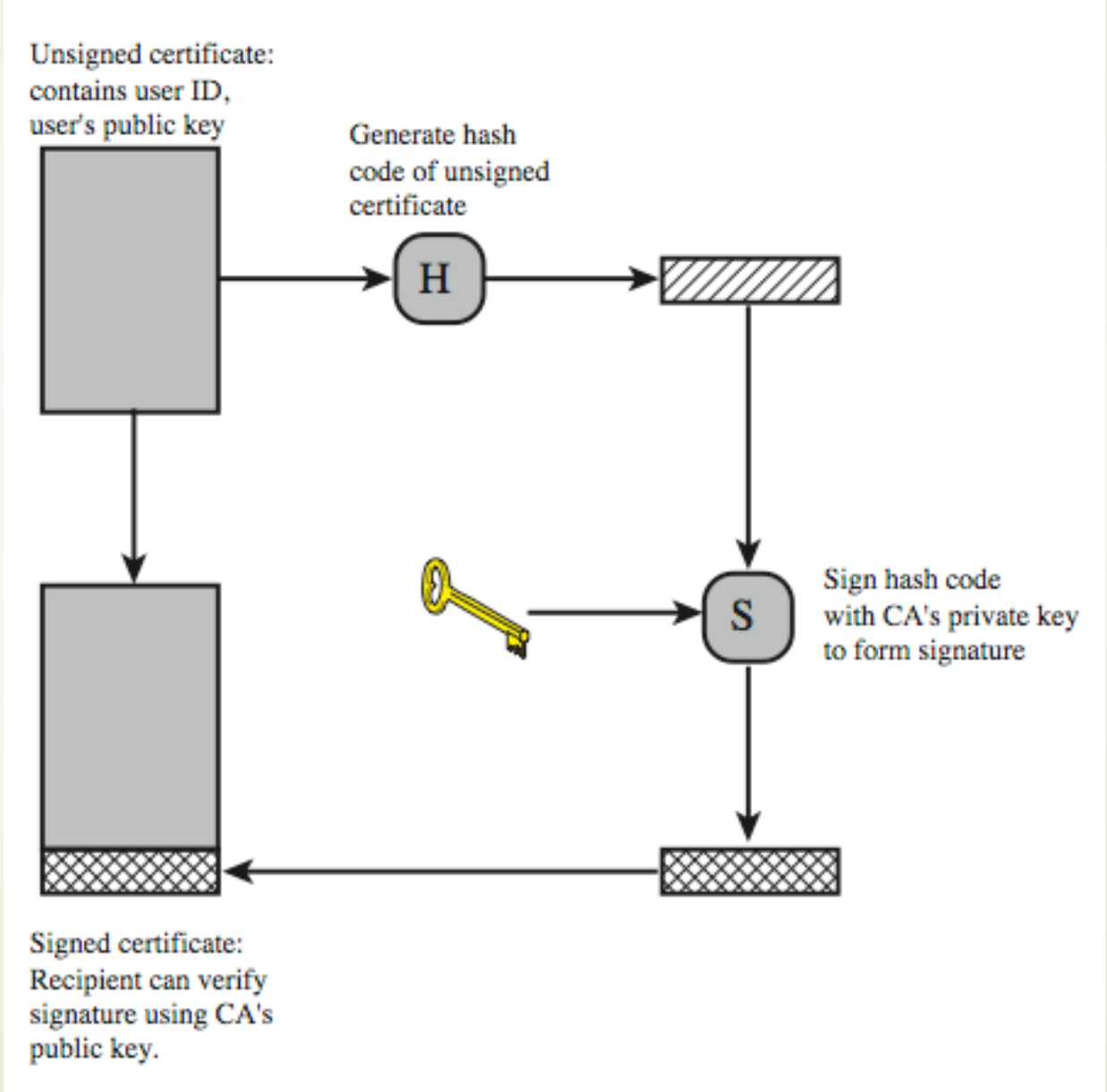
Kerberos Version 5

- developed in mid 1990's
- specified as Internet standard RFC 1510
- provides improvements over v4
  - addresses environmental shortcomings
    - encryption alg., network protocol, byte order, ticket lifetime, authentication forwarding, inter-realm auth.
  - and technical deficiencies
    - double encryption, non-std mode of use, session keys, password attacks

X.509 Authentication Service

- part of CCITT X.500 directory service standards
  - distributed servers maintaining user info database
- defines framework for authentication services
  - directory may store public-key certificates
  - with public key of user signed by certification authority
- also defines authentication protocols
- uses public-key crypto & digital signatures
  - algorithms not standardised, but RSA recommended
- X.509 certificates are widely used

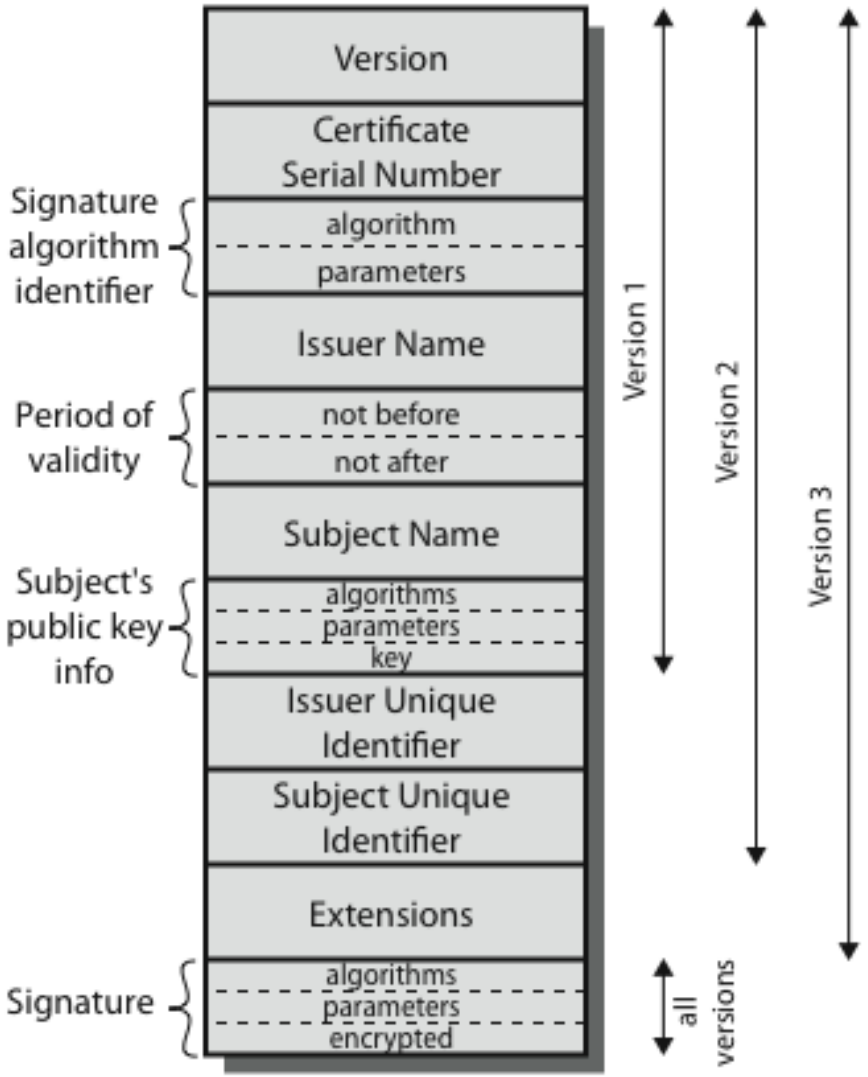
X.509 Certificate Use

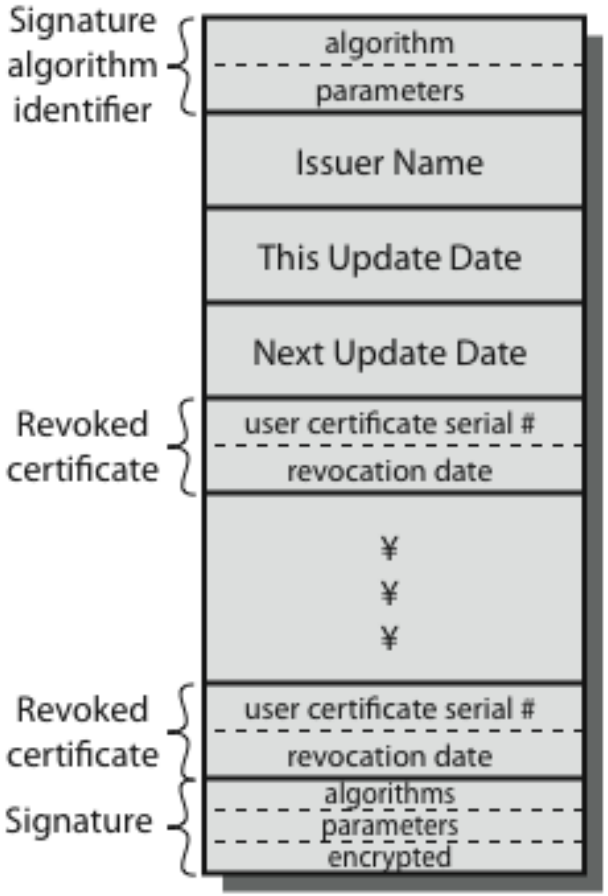


X.509 Certificates

- issued by a Certification Authority (CA), containing:
  - version (1, 2, or 3)
  - serial number (unique within CA) identifying certificate
  - signature algorithm identifier
  - issuer X.500 name (CA)
  - period of validity (from - to dates)
  - subject X.500 name (name of owner)
  - subject public-key info (algorithm, parameters, key)
  - issuer unique identifier (v2+)
  - subject unique identifier (v2+)
  - extension fields (v3)
  - signature (of hash of all fields in certificate)
- notation `CA<<A>>` denotes certificate for A signed by CA

X.509 Certificates





(b) Certificate Revocation List

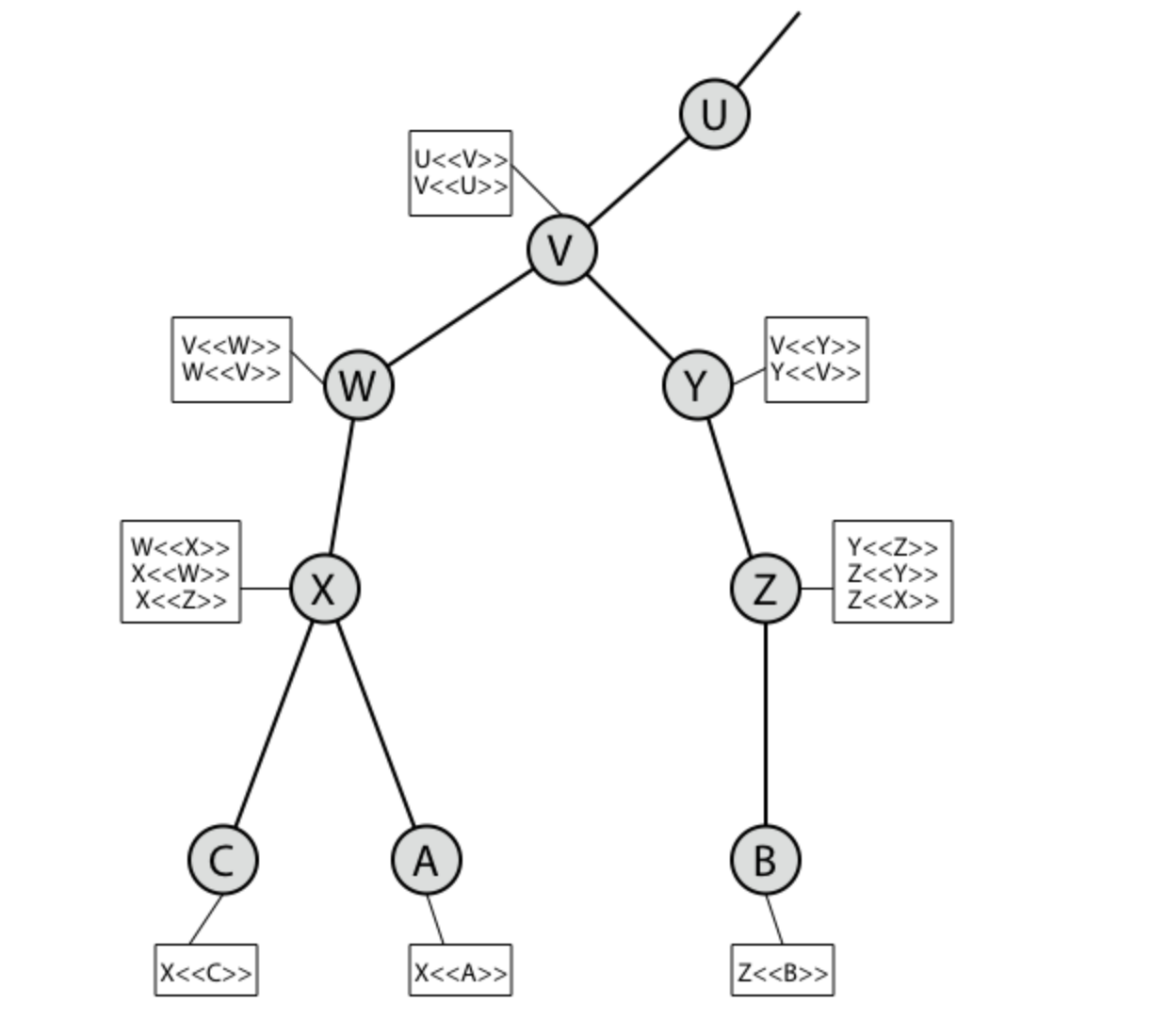
Obtaining a Certificate

- any **user with access to CA** can get any certificate from it
- only the **CA can modify** a certificate
- because cannot be forged, certificates can be **placed in a public directory**

CA Hierarchy

- if both users share a common CA then they are assumed to know its public key
- otherwise CA's must form a hierarchy
- use certificates linking members of hierarchy to validate other CA's
  - each CA has certificates for clients (forward) and parent (backward)
- each client trusts parents certificates
- enable verification of any certificate from one CA by users of all other CAs in hierarchy

CA Hierarchy Use



Certificate Revocation

- certificates have a period of validity
- may need to revoke before expiry, eg:
  - user's private key is compromised
  - user is no longer certified by this CA
  - CA's certificate is compromised
- CA's maintain list of revoked certificates
  - the Certificate Revocation List (CRL)
- users should check certificates with CA's CRL

Authentication Procedures

- X.509 includes three alternative authentication procedures:
  - One-Way Authentication
  - Two-Way Authentication
  - Three-Way Authentication
- all use public-key signatures

One-Way Authentication

- 1 message (A->B) used to establish
  - the identity of A and that message is from A
  - message was intended for B
  - integrity & originality of message
- message must include timestamp, nonce, B's identity and is signed by A
- may include additional info for B
  - eg session key

Two-Way Authentication

- 2 messages (A->B, B->A) which also establishes in addition:
  - the identity of B and that reply is from B
  - that reply is intended for A
  - integrity & originality of reply
- reply includes original nonce from A, also timestamp and nonce from B
- may include additional info for A

Three-Way Authentication

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks
- has reply from A back to B containing signed copy of nonce from B
- means that timestamps need not be checked or relied upon

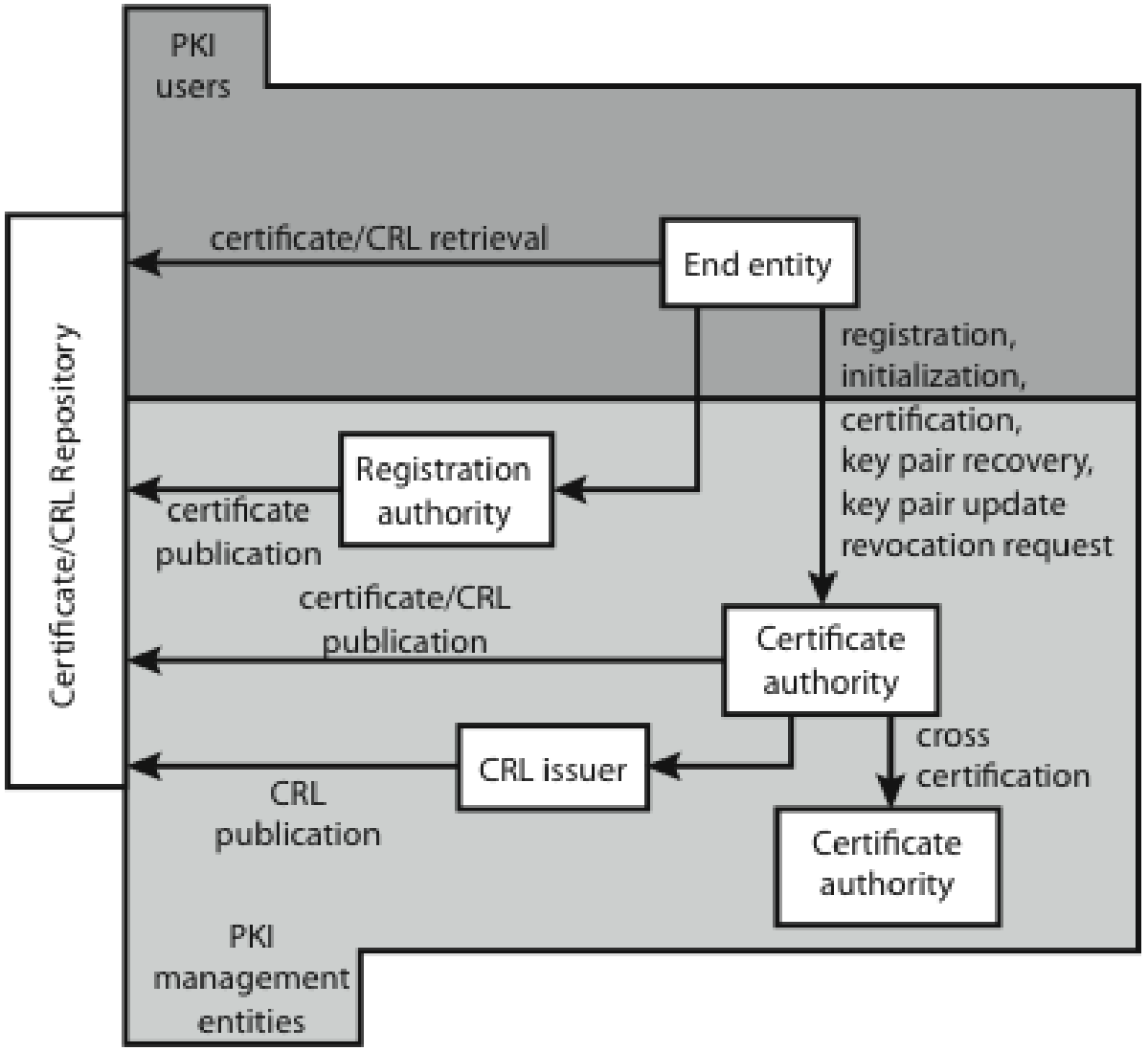
X.509 Version 3

- has been recognised that additional information is needed in a certificate
  - email/URL, policy details, usage constraints
- rather than explicitly naming new fields defined a general extension method
- extensions consist of:
  - extension identifier
  - criticality indicator
  - extension value

Certificate Extensions

- key and policy information
  - convey info about subject & issuer keys, plus indicators of certificate policy
- certificate subject and issuer attributes
  - support alternative names, in alternative formats for certificate subject and/or issuer
- certificate path constraints
  - allow constraints on use of certificates by other CA's

Public Key Infrastructure



PKIX Management

- functions:
  - registration
  - initialization
  - certification
  - key pair recovery
  - key pair update
  - revocation request
  - cross certification
- protocols: CMP, CMC