Module 2: Cryptographic Tools, Key Distribution and Management

Dr. Natarajan Meghanathan
Associate Professor of Computer Science
Jackson State University, Jackson, MS 39232
E-mail: natarajan.meghanathan@jsums.edu

Simplified Model for Symmetric Key Encryption

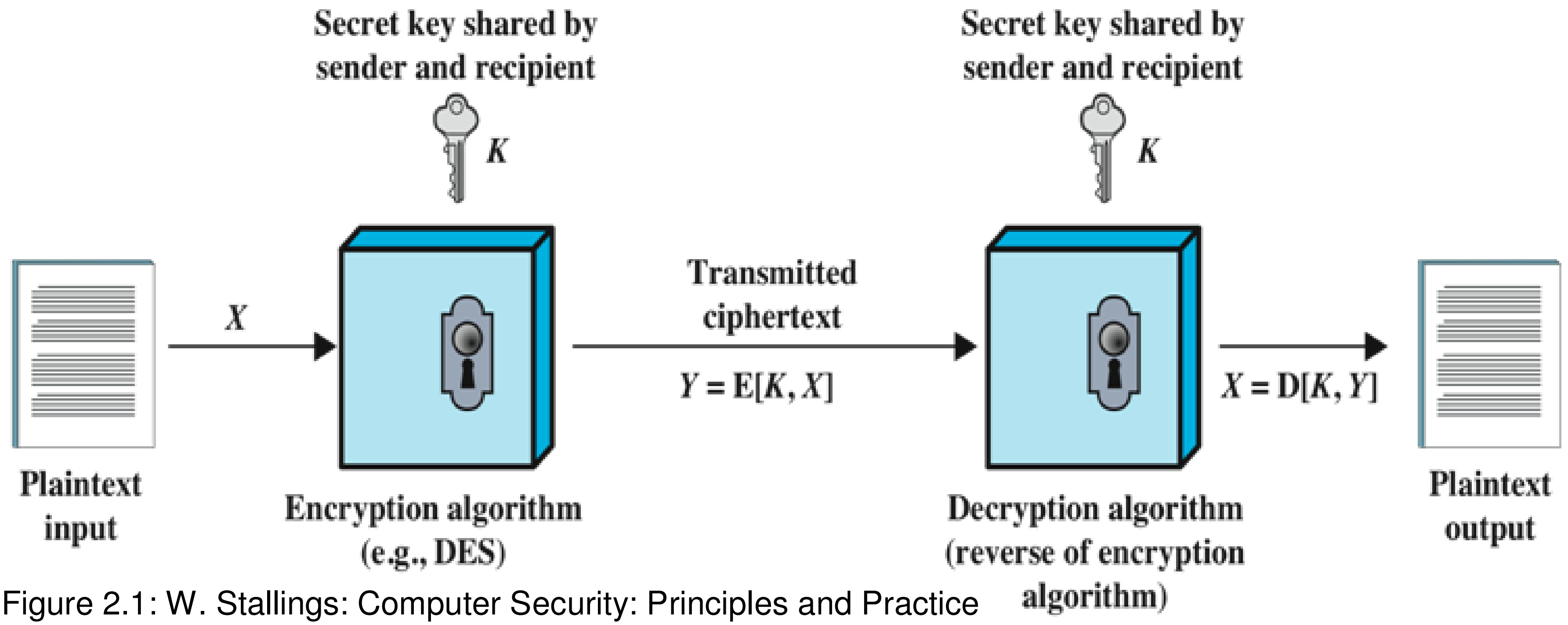

Secret key shared by sender and recipient
K
Secret key shared by sender and recipient
K
X
Transmitted ciphertext
Y = E[K, X]
X = D[K, Y]
Plaintext input
Encryption algorithm (e.g., DES)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

Figure 2.1: W. Stallings: Computer Security: Principles and Practice

- <u>Strong requirements for a symmetric key encryption algorithm (cipher):</u>
  - An opponent having one or more ciphertexts along with their corresponding plaintexts produced using a particular key should not be able to decrypt a given ciphertext or deduce the key.
  - The secret key is known only to the sender and receiver.

Attacking Symmetric Ciphers

- <u>Cryptanalytic Attacks</u>
  - rely on:
    - nature of the algorithm
    - some knowledge of the general characteristics of the plaintext
    - some sample plaintext-ciphertext pairs
  - exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or the key being used
    - if successful all future and past messages encrypted with that key are compromised
- <u>Brute Force Attacks</u>
  - try all possible keys on some ciphertext until an intelligible translation into plaintext is obtained
  - on average half of all possible keys must be tried to achieve success

Average Time for Brute Force Key Search

Key Size (bits)	Number of Alternative Keys	Time Required at 1 Decryption/µs	Time Required at 10^6 Decryptions/µs
32	$2^{32} = 4.3 \times 10^9$	2^{31} µs = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	2^{55} µs = 1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	2^{127} µs = 5.4×10^{24} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	2^{167} µs = 5.9×10^{36} years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	2×10^{26} µs = 6.4×10^{12} years	6.4×10^6 years

Table 2.1: W. Stallings: Computer Security: Principles and Practice

Symmetric Block Encryption Algorithms

- The most commonly used symmetric encryption algorithms are block ciphers
- A block cipher processes the plaintext input in fixed-size blocks and produces a block of ciphertext of equal size for each plaintext block.
- Larger plaintext inputs are processed as a sequence of fixed-size blocks.

	DES	Triple DES	AES
Plaintext block size (bits)	64	64	128
Ciphertext block size (bits)	64	64	128
Key size (bits)	56	112 or 168	128, 192, or 256

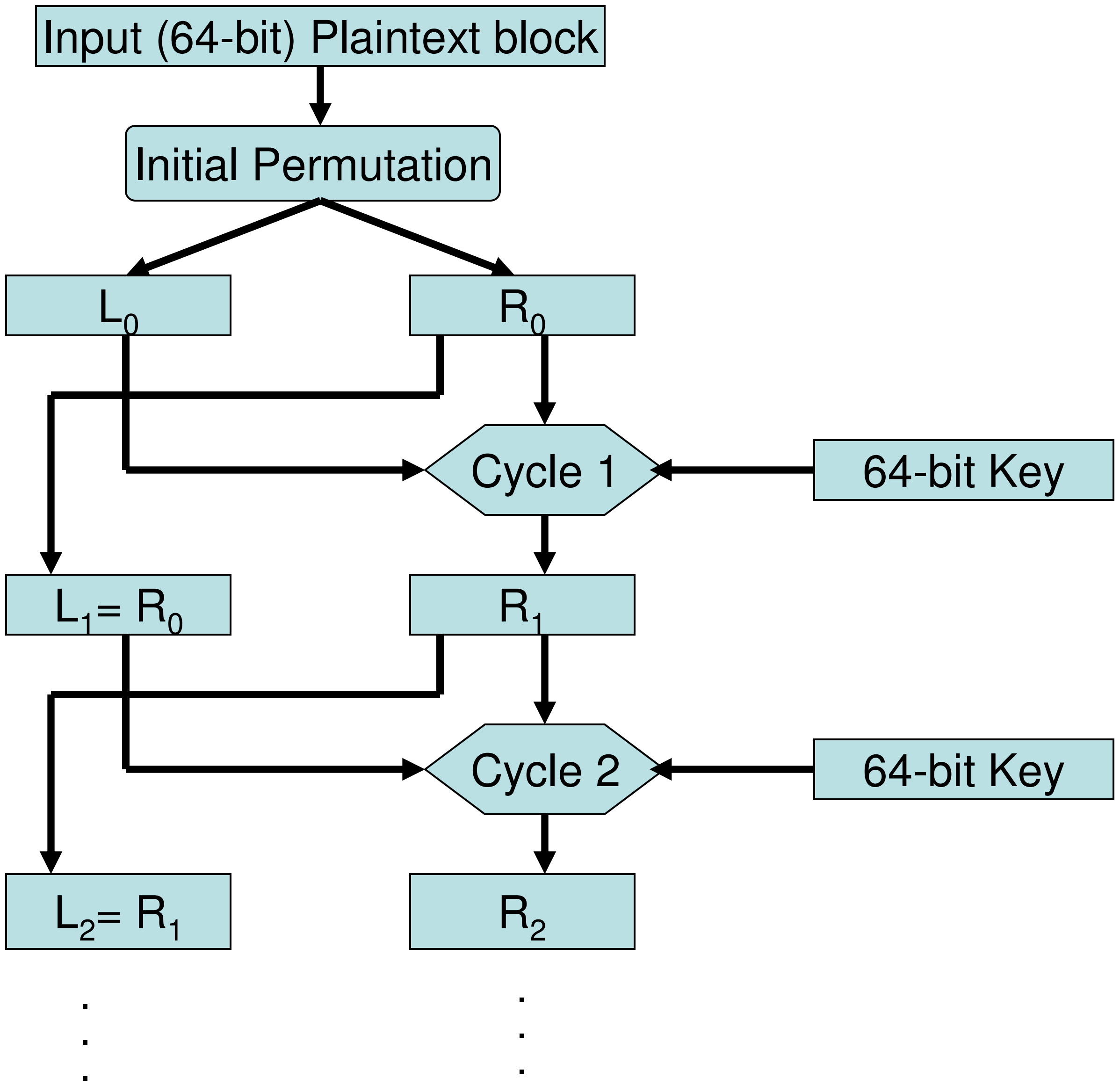
Table 2.2: W. Stallings: Computer Security: Principles and Practice

DES = Data Encryption Standard
AES = Advanced Encryption Standard

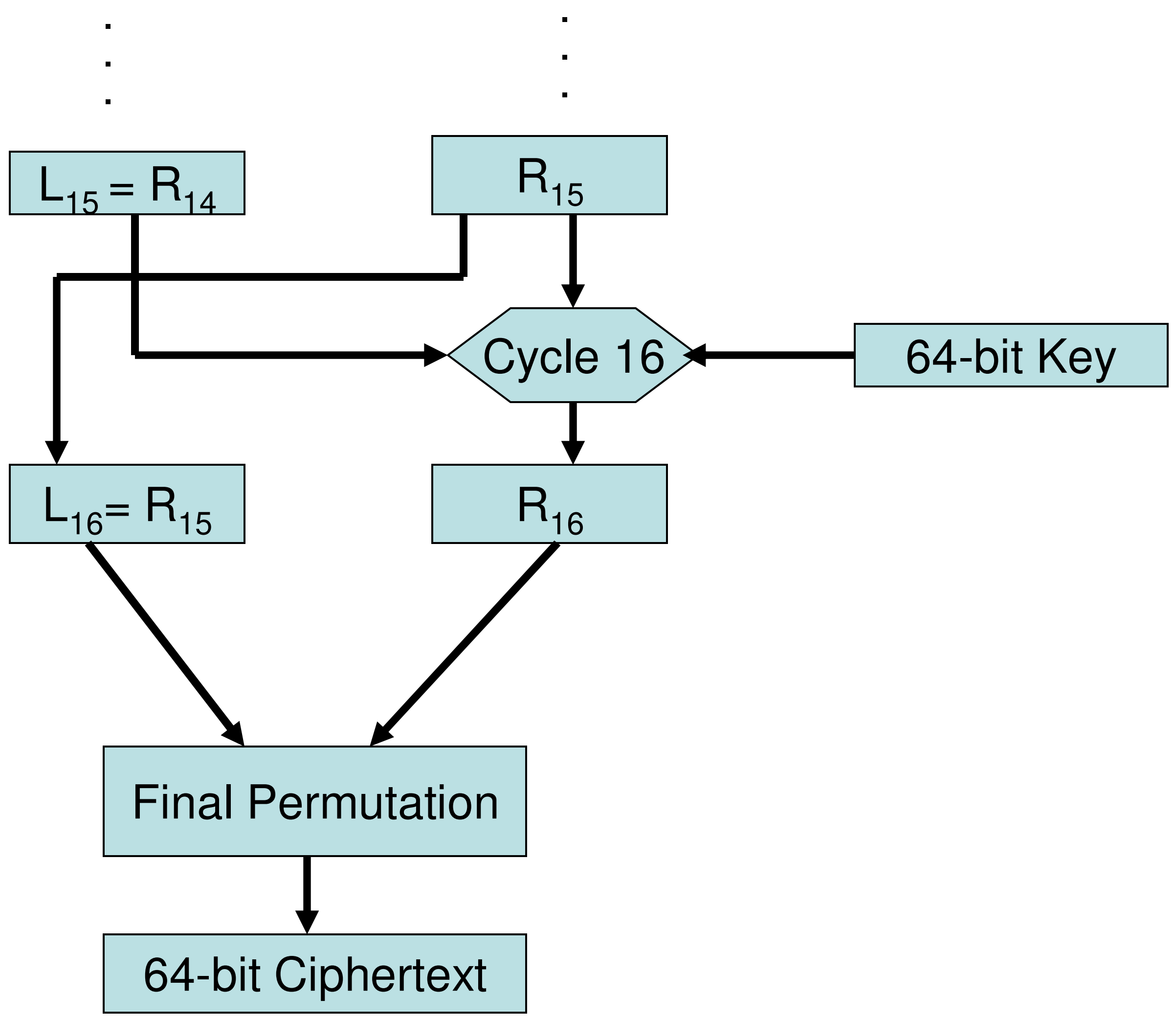
DES (Data Encryption Standard)

- DES is the most widely used symmetric key encryption algorithm
- DES takes a 64-bit plaintext block and a 56-bit key as inputs and produces a 64-bit ciphertext block.
  - Other than brute-force approach, no effective cryptanalytic attack on DES has been found.
- Triple DES: Repeats thrice with three keys
  - To encrypt: C = E(K3, D(K2, E(K1, P)))
  - To decrypt: P = D(K1, E(K2, D(K3, C)))
  - The encryption/ decryption algorithm used is DES.
  - **Attractions:**
    - 168-bit key length overcomes the vulnerability to brute-force attack of DES
    - underlying encryption algorithm is the same as in DES
  - **Drawbacks:**
    - algorithm is sluggish in software

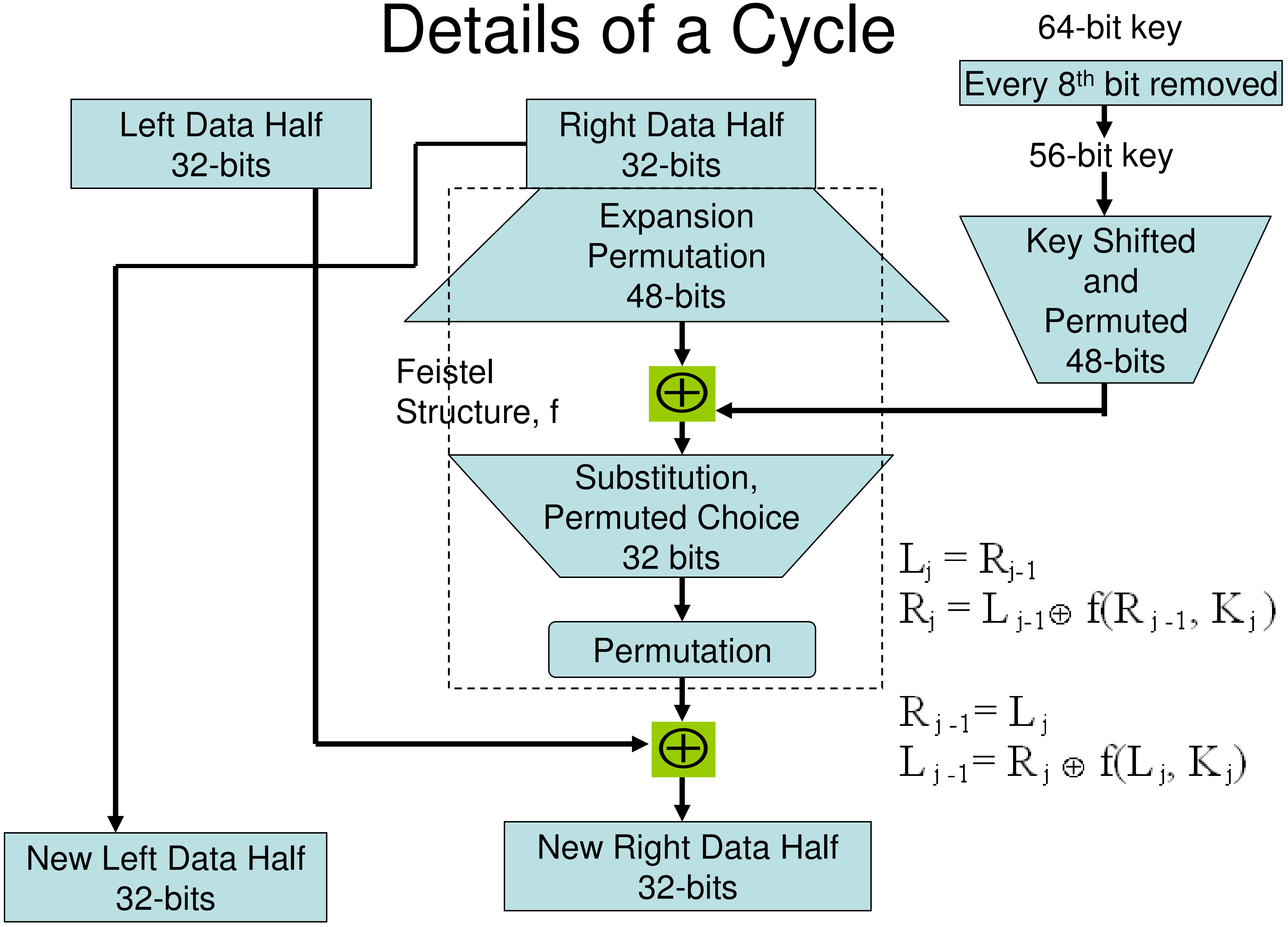
Cycles of Substitution and Permutation



Cycles of Substitution and Permutation



Details of a Cycle



64-bit key
Every 8th bit removed
56-bit key
Key Shifted and Permuted 48-bits
Left Data Half 32-bits
Right Data Half 32-bits
Expansion Permutation 48-bits
Feistel Structure, f
Substitution, Permuted Choice 32 bits
Permutation
New Left Data Half 32-bits
New Right Data Half 32-bits

$L_j = R_{j-1}$

$R_j = L_{j-1} \oplus f(R_{j-1}, K_j)$

$R_{j-1} = L_j$

$L_{j-1} = R_j \oplus f(L_j, K_j)$

Advanced Encryption Standard (AES)

- Needed a replacement for 3DES for long-term use.
- The NIST in 1997 issued a Call for Proposals for a new Advanced Ecnryption Standard (AES):
  - Should have a security strength equal or better than 3DES
  - Significantly improved efficiency
  - Symmetric block cipher with a block length of 128 bits
  - Support for key lengths of 128, 192 and 256 bits.

Multiple Blocks of Encryption

- **Typically symmetric encryption is applied to a unit of data larger than a single 64-bit or 128-bit block**
  - **electronic codebook (ECB) mode is the simplest approach to multiple-block encryption**
  - **each block of plaintext is encrypted using the same key**
  - **cryptanalysts may be able to exploit regularities in the plaintext**
- **Modes of operation**
  - **alternative techniques developed to increase the security of symmetric block encryption for large sequences**
  - **Example: Cipher Block Chaining**

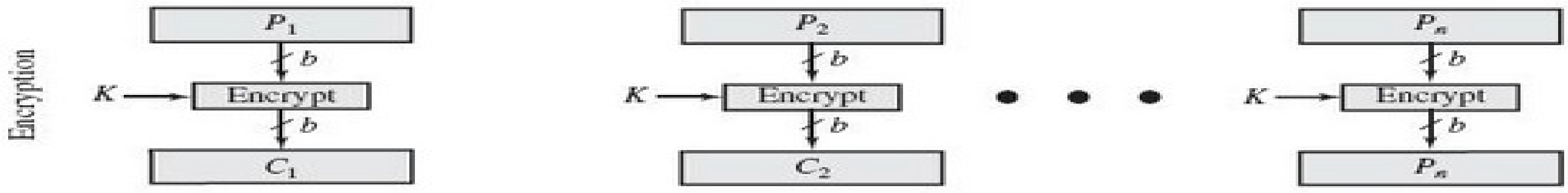
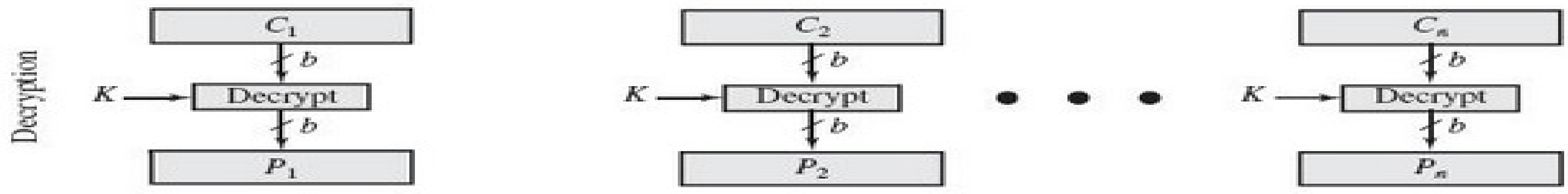
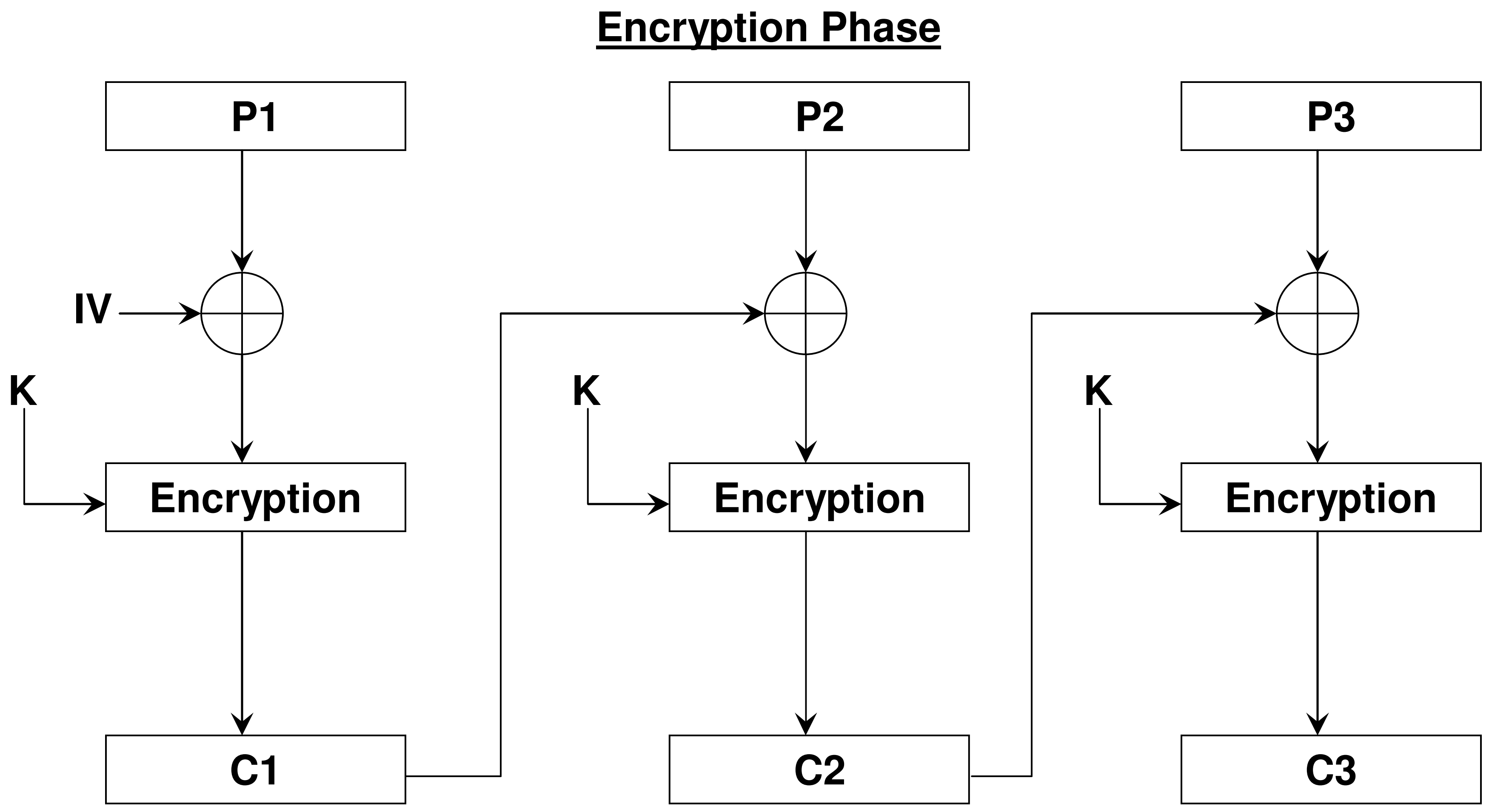


Figure 2.3: W. Stallings: Computer Security: Principles and Practice



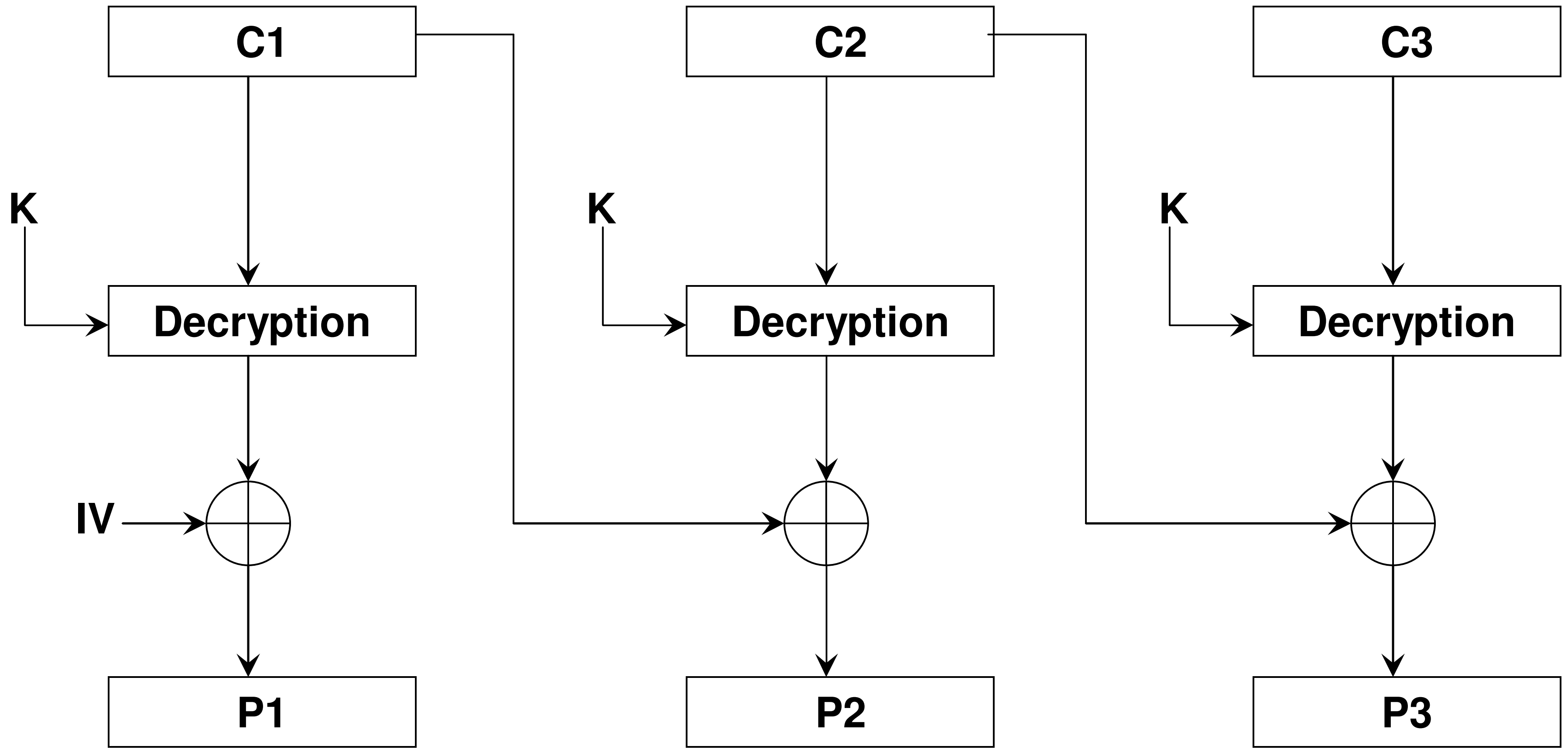
Cipher Block Chaining (CBC) to Create Avalanche Effect



Source (adapted from): Figure 6.4 from William Stallings – Cryptography and Network Security, 5th Edition

Cipher Block Chaining (CBC) to Create Avalanche Effect

Decryption Phase



Source: Figure 6.4 from William Stallings – Cryptography and Network Security, 5th Edition

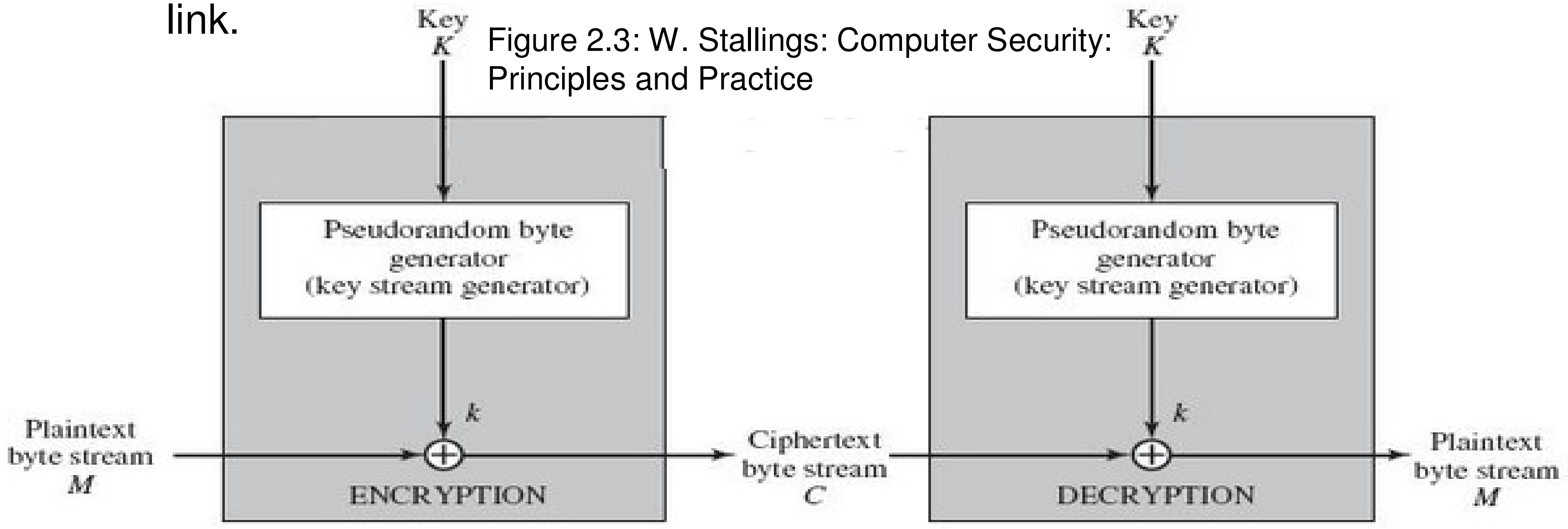
Advantages and Limitations of CBC

- Advantages: A ciphertext block depends on all blocks before it
- Any change to a plaintext block affects all of the succeeding ciphertext blocks – creates an Avalanche effect. This property can be used to compute a "Message Authentication Code" (MAC) for the entire plaintext and sent as part of the message.
- If the "integrity" of the message is the only required criterion, then we can send IV, P1, P2, …, P_{last_block}, MAC
  - The MAC could be actually the last ciphertexct block.
  - If any intruder changes any of the plaintext, the Avalanche Effect property of CBC requires that the MAC value computed by the destination to be different than what is sent by the sender as part of the message.
- Limitations: Needs an Initialization Vector (IV), which must be known to sender & receiver
- Since a recovered plaintext block (from CBC decryption) is not used for further decryptions and it is the only ciphertext of a previous block that is being used to decrypt the ciphertext of the current block, with CBC decryption, a change in a ciphertext block only affects two of the recovered plaintext blocks.
- The IV cannot be sent in clear text. Hence, the IV must be either a fixed value or must be sent encrypted offline (using schemes such as public-key encryption) before the rest of the message.

Stream Cipher

- A block cipher encrypts and decrypts a block (of several bytes) at a time.
  - Suitable for applications that deal with blocks of data, such as file transfer, e-mail and database
- A Stream cipher encrypts and decrypts one element at a time (typically, one byte; could be even a bit or more than a byte)
  - Relies on the use of a pseudorandom bit generator (on whose design the strength of the cipher depends on) to generate the key bits/bytes.
  - Fast and easier implementation
  - Suitable for applications that require encryption/ decryption of a stream of data, such as: over a data communications channel or a browser/web link.

Figure 2.3: W. Stallings: Computer Security: Principles and Practice


Key
K
Pseudorandom byte generator (key stream generator)
k
Plaintext byte stream
M
ENCRYPTION
Ciphertext byte stream
C
Key
K
Pseudorandom byte generator (key stream generator)
k
DECRYPTION
Plaintext byte stream
M

Message or Data Authentication

- Encryption protects against passive attacks (eavesdropping).
- Message or Data authentication is to protect against active attacks (like falsification of data and transactions).
  - Verify that the contents of the message have not been altered.
  - Verify that the source is authentic.
  - Verify the message's timeliness (that it has not been artificially delayed and replayed)
  - Verify the sequence relative to other messages flowing between two parties.

Message Authentication

- Idea: Generate an authentication tag and append to each transmitted message.
  - The tag is used to authenticate the sender and the integrity of the message.
- Scenarios:
  - The confidentiality of the message is not of concern.
  - Destination is heavily loaded (or limited with computation resources) and cannot do decryption for all messages

Message Authentication using Message Authentication Code

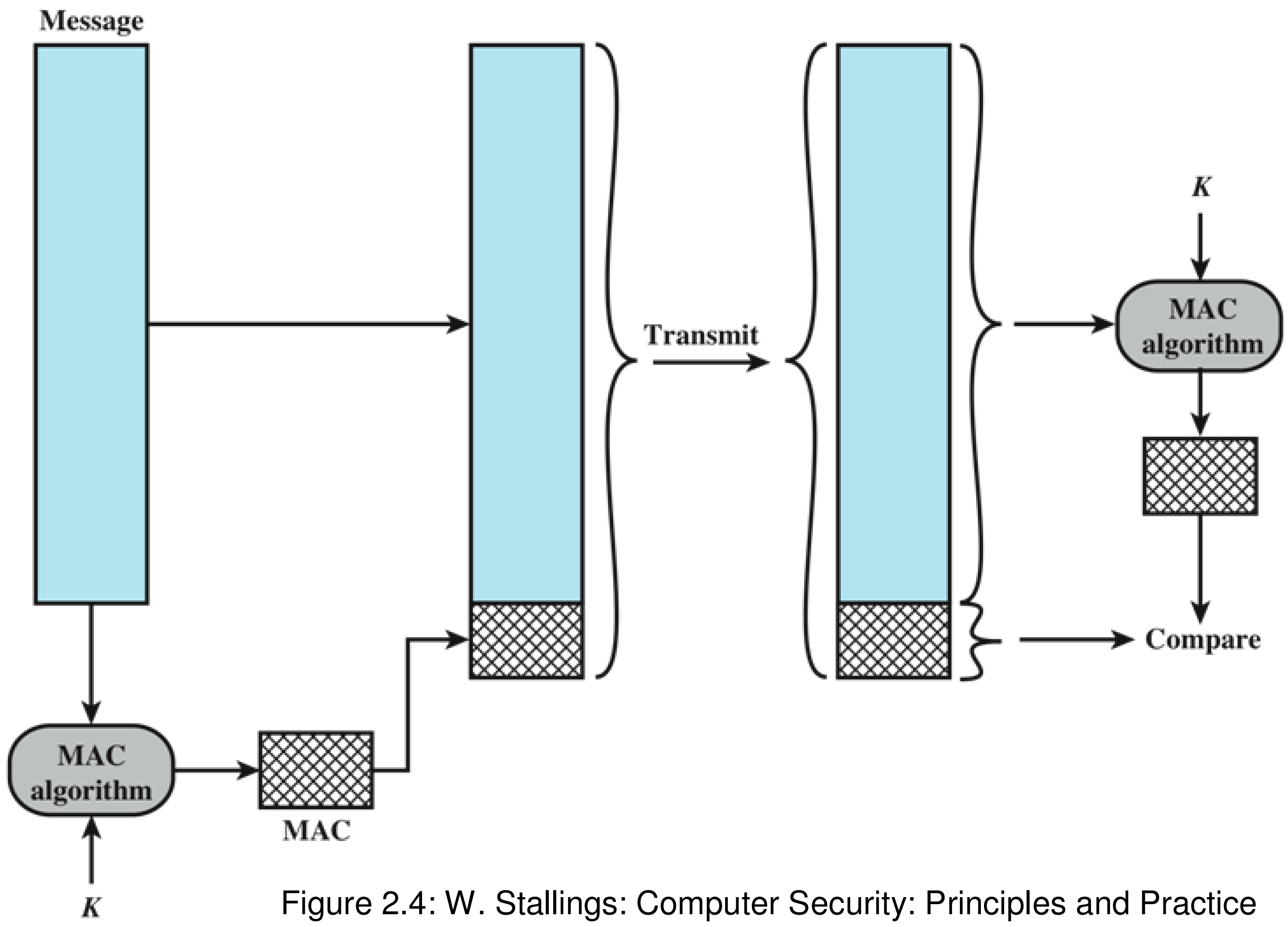


Figure 2.4: W. Stallings: Computer Security: Principles and Practice

Message Authentication Code (MAC)

- Compute the ciphertext of the plaintext using DES and the shared secret key.
- Use the last 16 or 32-bits of the resulting ciphertext as the message authentication code.
  - Almost impossible to decrypt back to the plaintext
  - Authenticates the sender since nobody else could encrypt using the particular secret key
  - Facilitates checking for message integrity, because an intruder changing the contents of the message could not accordingly update the MAC (because the intruder does not know the secret key).

One-way Hash Function

- A hash function accepts a variable-size message *M* as input and produces a fixed-size message digest H(*M*) as output.
- Unlike a MAC, a hash function does not take a secret key as input; but, can still be used to authenticate the sender as well as the integrity.
- The length of the message (in bits) is padded along with the message to compute the hash value. This is to make it complicated for an attacker to come up with a message of the same hash value.
- The hash value is a "finger print" of the file, message or block of data.

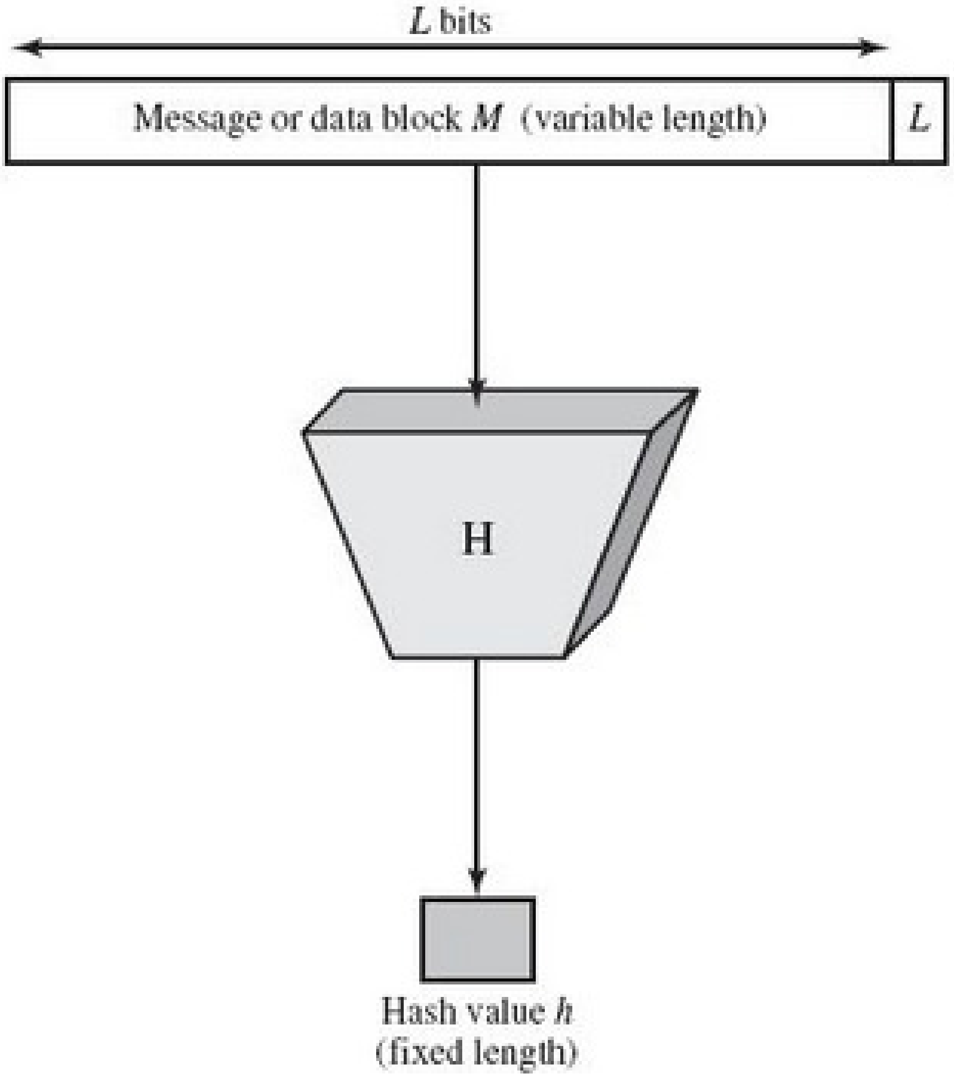
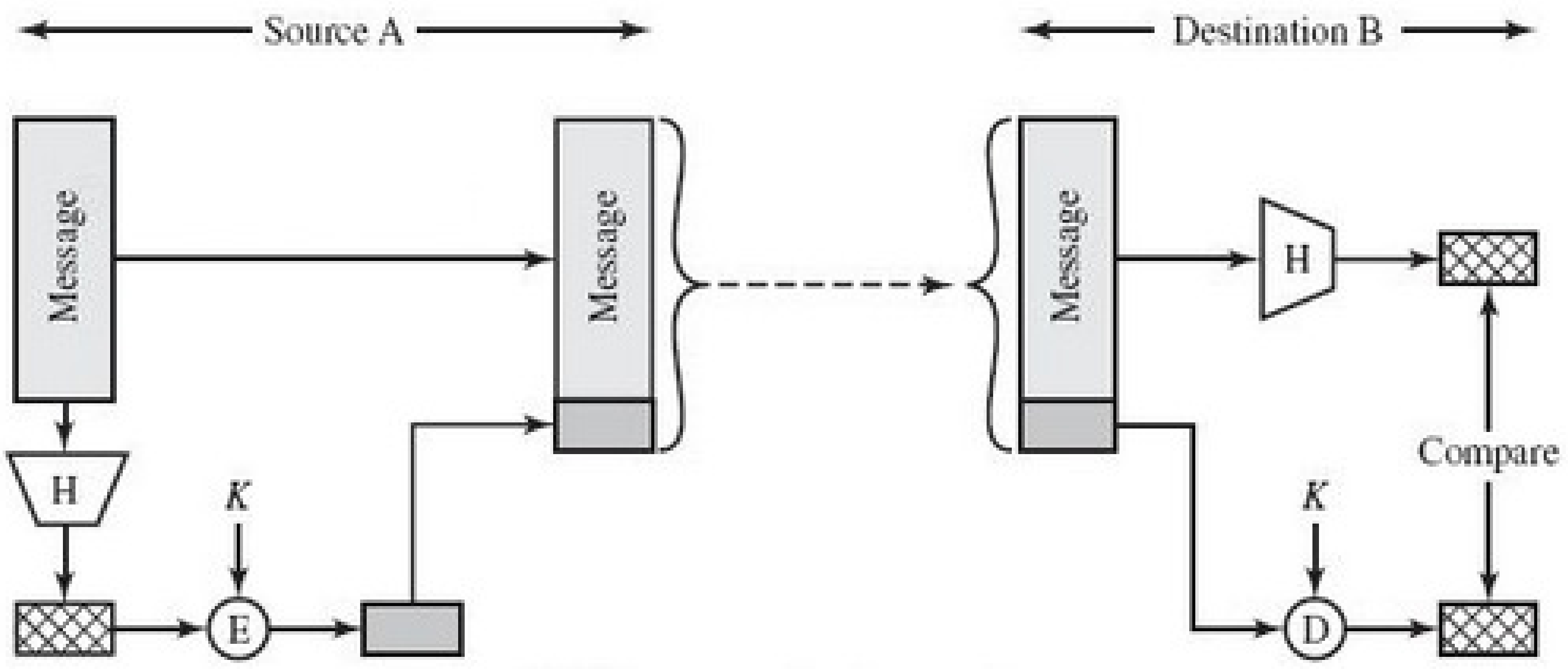


Figure 2.5: W. Stallings: Computer Security: Principles and Practice

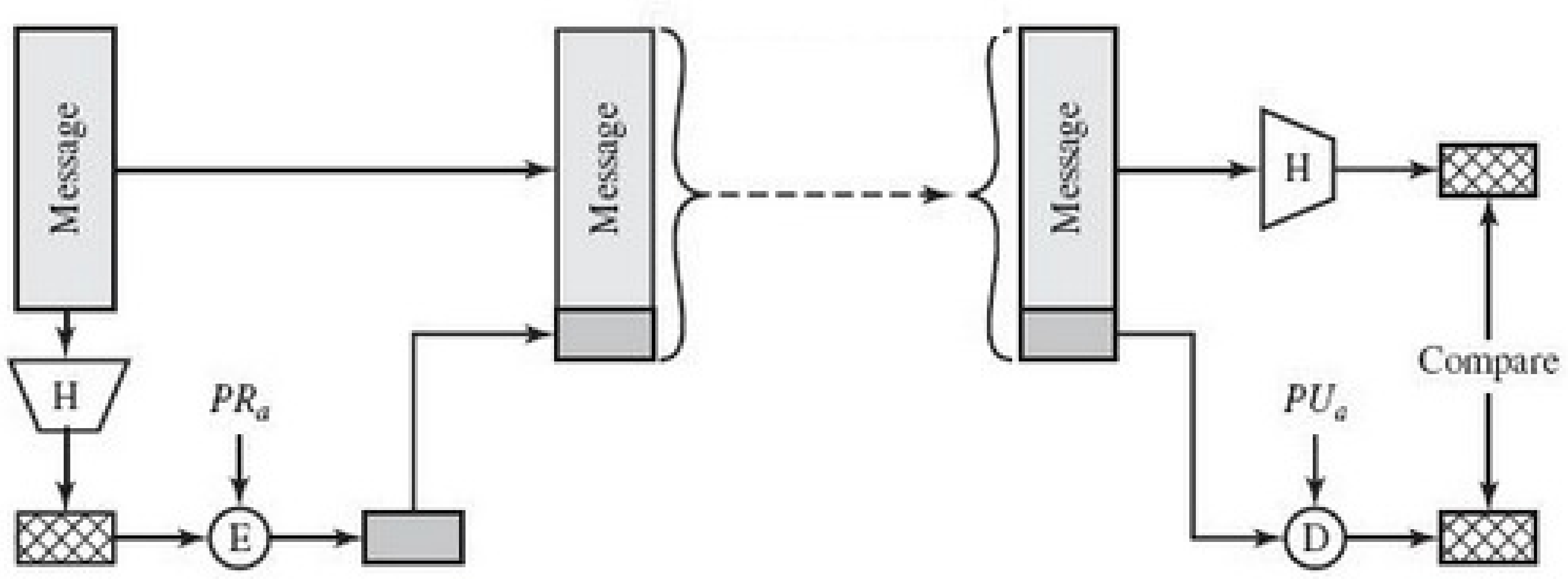
Message Authentication using One-way Hash Function



Using Conventional Encryption

Figure 2.6 (a): W. Stallings: Computer Security: Principles and Practice

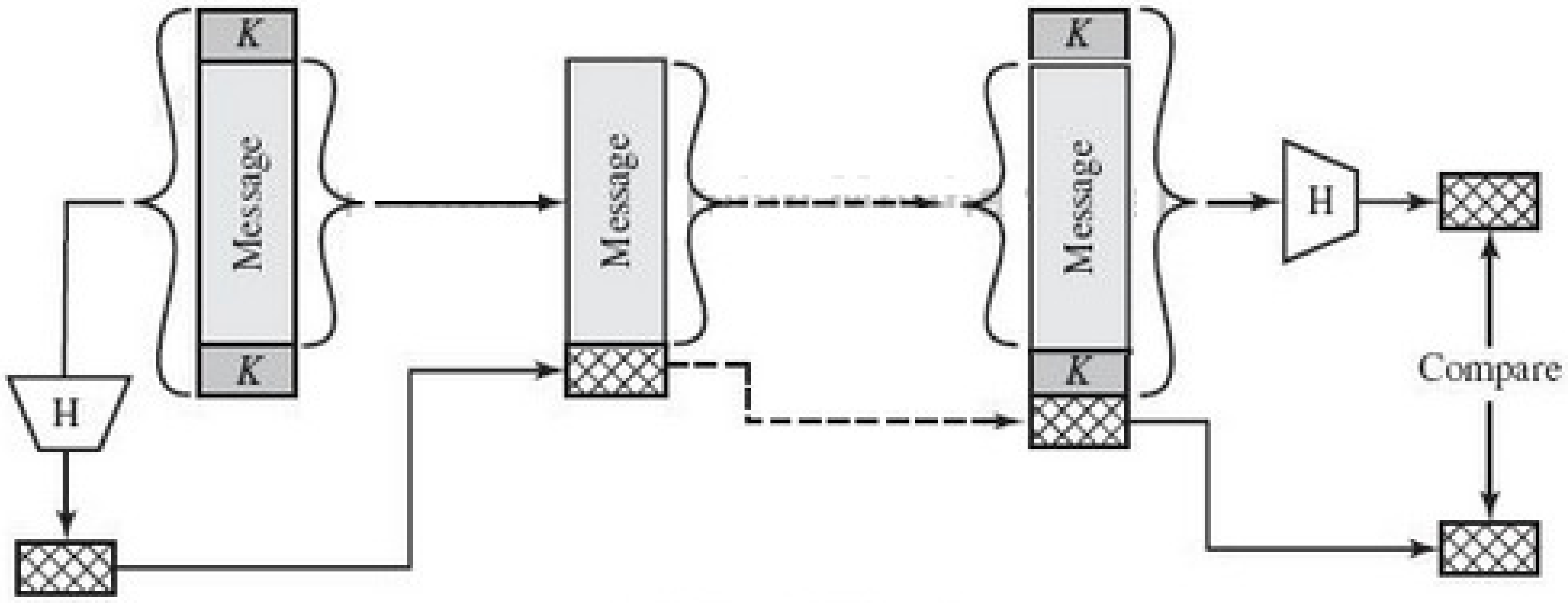
Message Authentication using One-way Hash Function



Using Public-Key Encryption

Figure 2.6 (b): W. Stallings: Computer Security: Principles and Practice

Message Authentication using One-way Hash Function



Note that no encryption is involved in this scheme

Using Secret Key (Variant of HMAC)

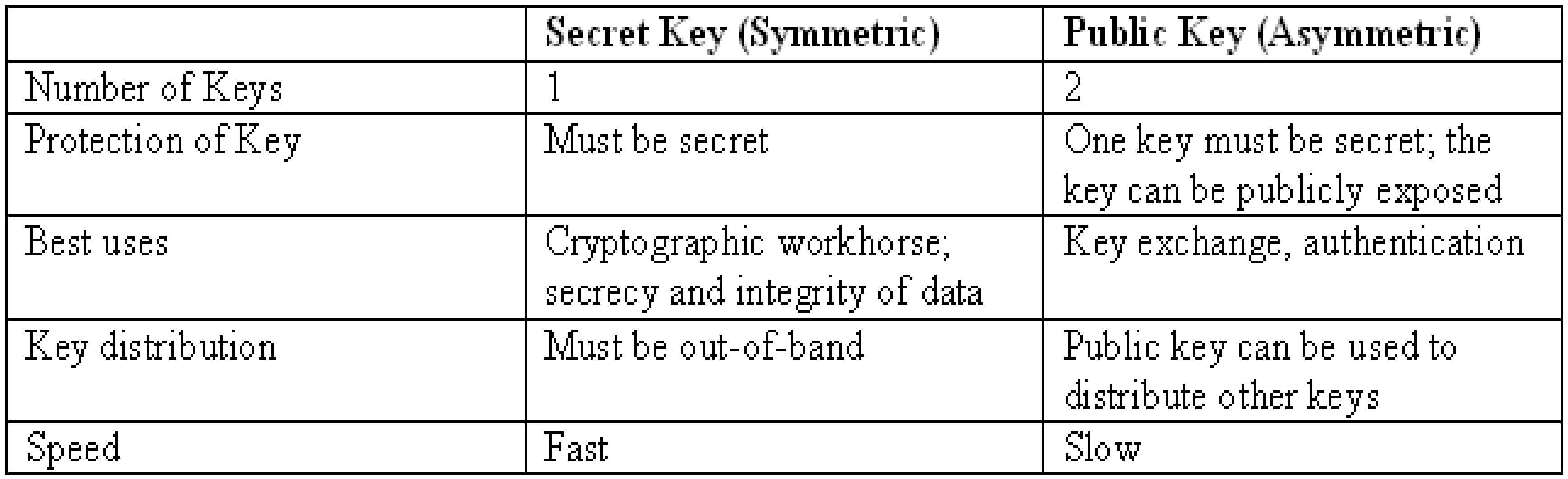
Figure 2.6 (c): W. Stallings: Computer Security: Principles and Practice

Secure Hash Function: Requirements

- To be useful for message authentication, a hash function *H* must have the following properties:
  - H can be applied to a block of data of any size
  - H produces a fixed-length output
  - H(x) is relatively easy to compute for any given x.
  - <u>One-way property</u>: Given a hash value *h*, it is computationally infeasible to compute the underlying message x such that H(x) = h.
  - <u>Weak-collision resistant</u>: For any given block x, it is computationally infeasible to find another block y, where $y \neq x$ and H(y) = H(x).
  - <u>Strong-collision resistant</u>: It is computationally infeasible to find any pair of blocks x and y, such that $y \neq x$ and H(y) = H(x).
- Hash functions that satisfy the first five properties (listed above) are said to be *weak hash functions*. Hash functions that satisfy all of the above properties are said to be *strong hash functions*.
- Secure Hash Algorithm (SHA) and its variants (SHA-256, 384, 512) are the commonly used hash functions.
- Other uses: (1) Store passwords for operating systems; (2) Periodically compute/ verify the hash values of files; the hash values are stored in a secure location or disc.

Public-Key Encryption

- Public-key cryptography is based on mathematical functions, unlike the symmetric ciphers that are based on operations on bit patterns.
- Public-key cryptography is asymmetric, involving the use of two separate keys.
- Each user has two keys: Public key of a user is known to others; private key of the user is known only to the user.
- Either of the two keys can be used for encryption/ decryption. However, a message encrypted with one key can be decrypted only with the other key.
- Examples: RSA, Elliptic Curve Cryptography



	Secret Key (Symmetric)	**Public Key (Asymmetric)**
Number of Keys	1	2
Protection of Key	Must be secret	One key must be secret; the key can be publicly exposed
Best uses	Cryptographic workhorse; secrecy and integrity of data	Key exchange, authentication
Key distribution	Must be out-of-band	Public key can be used to distribute other keys
Speed	Fast	Slow

Public-Key Encryption
(Encryption with Public Key of the Receiver)

Provides confidentiality

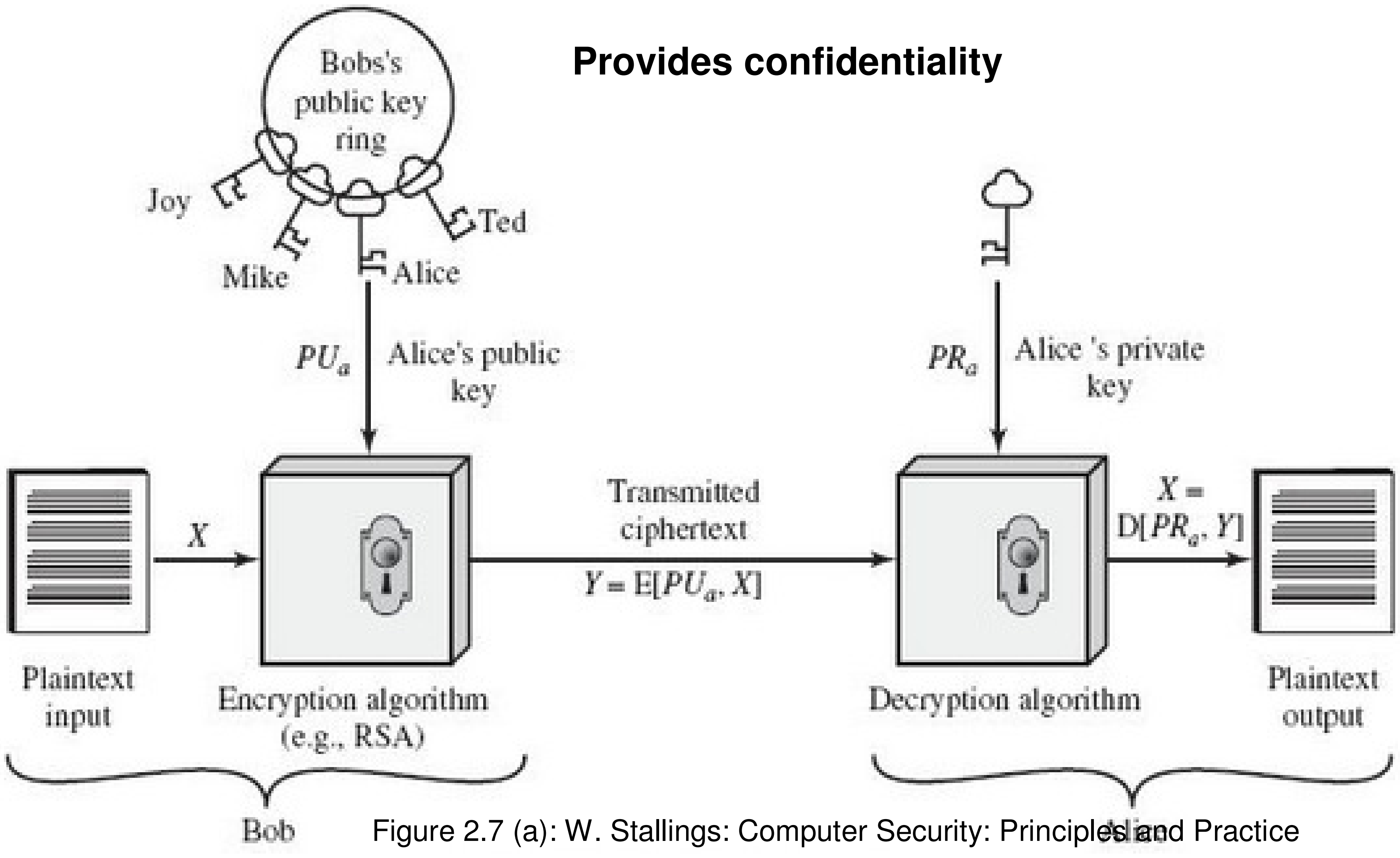


Figure 2.7 (a): W. Stallings: Computer Security: Principles and Practice

Public-Key Encryption

(Encryption with Private Key of the Sender)

Instead of plaintext, one could even simply use the hash value of the plaintext for encryption and send the encrypted hash value along with the plaintext. This will be called the **digital signature**.

Provides authentication and data integrity

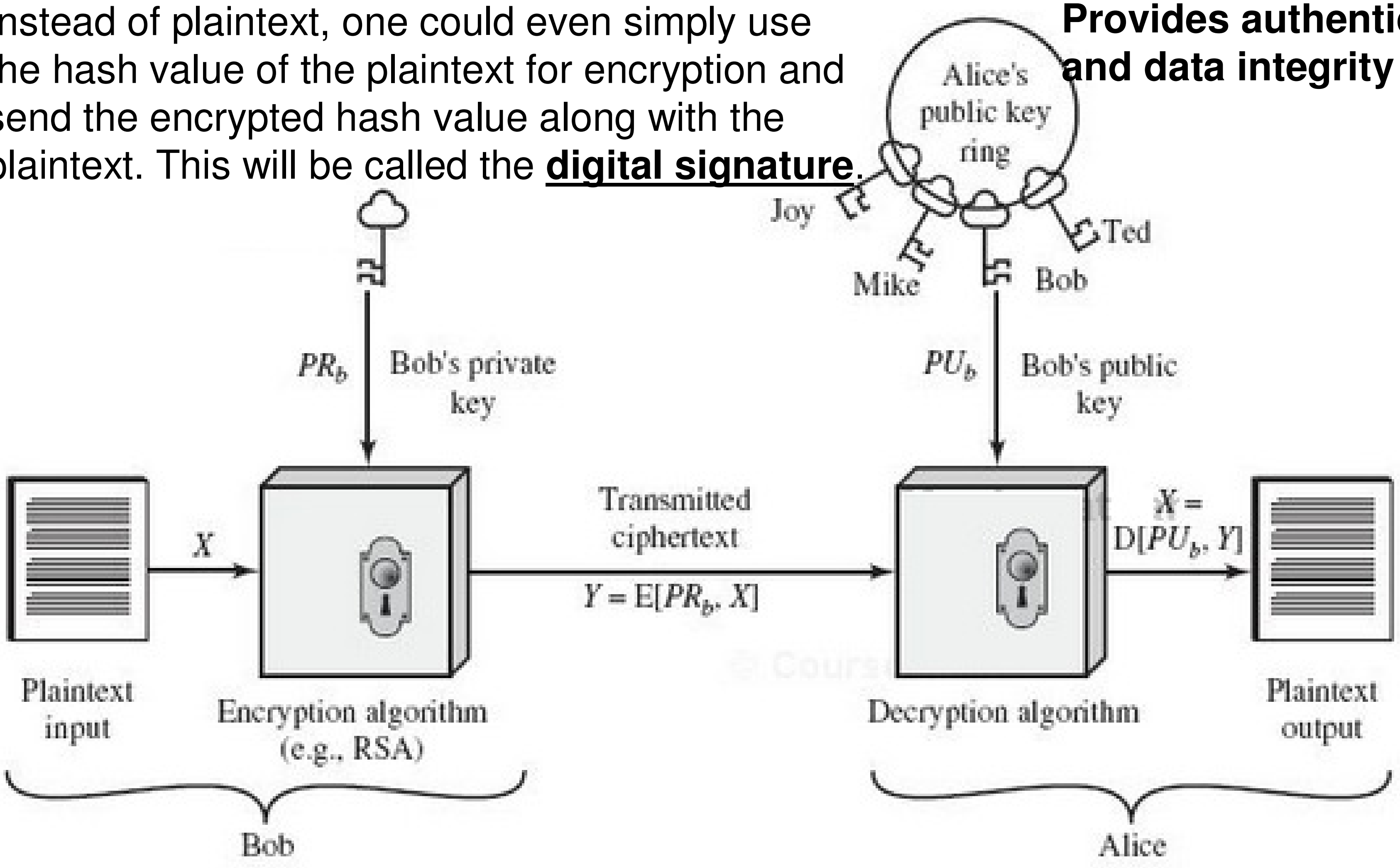


Figure 2.7 (b): W. Stallings: Computer Security: Principles and Practice

Man-in-the-Middle Attack

- Man-in-the-middle (MITM) attack is an attack in which an attacker is able to read, insert and modify at will, messages between two parties without either party knowing that the link between them has been compromised.
- The attacker must be able to observe and intercept messages going between the two victims.
- Example: (MITM attack on public-key cryptography)
  - Suppose Alice wishes to communicate with Bob.
  - Mallory wants to eavesdrop their conversation or also possibly deliver a false message to Bob.
  - First, Alice must ask Bob for his public key.
  - If Bob sends his public key to Alice, but Mallory is able to intercept it, a MITM attack can begin.
  - Mallory sends a forged message to Alice that claims to have come from Bob, but contains Mallory's public key
  - Alice believes the public key received to be that of Bob's. So, Alice encrypts the message she wishes to send to Bob using the public key received and transmits on the link to Bob.
  - Mallory could now intercept the message, decrypt it with his private key and get the actual contents of the message.
  - Mallory now again encrypts the message (could be even altered too) with Bob's public key and transmits the message to Bob.
  - Bob on receiving the message, decrypts the message with his private key and reads the contents of the message assuming it came from Alice

Public-Key Certificates

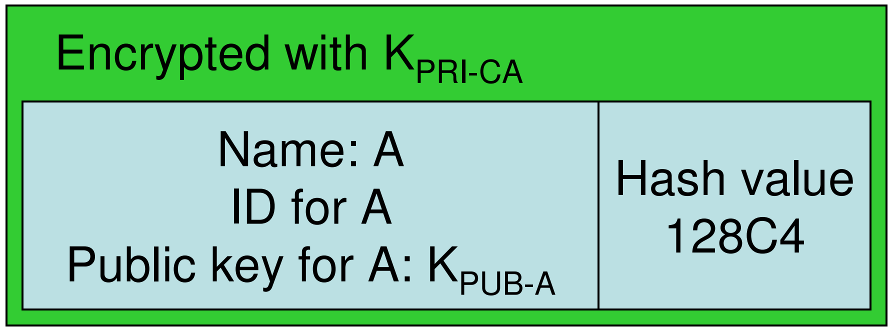
- Each of us adopt a “trust threshold” – a degree to which we are willing to believe an unidentified individual.
- We will use the concept of “vouching for” by a third party as the basis of thrust in settings where two parties do not know about each other.

- <u>Certification Authority (CA):</u> Is an entity that issues digital certificates that contain a public key and the identity of the owner.
- The CA attests that the public key contained in the digital certificate belongs to the person (CA is a sort of digital notary).

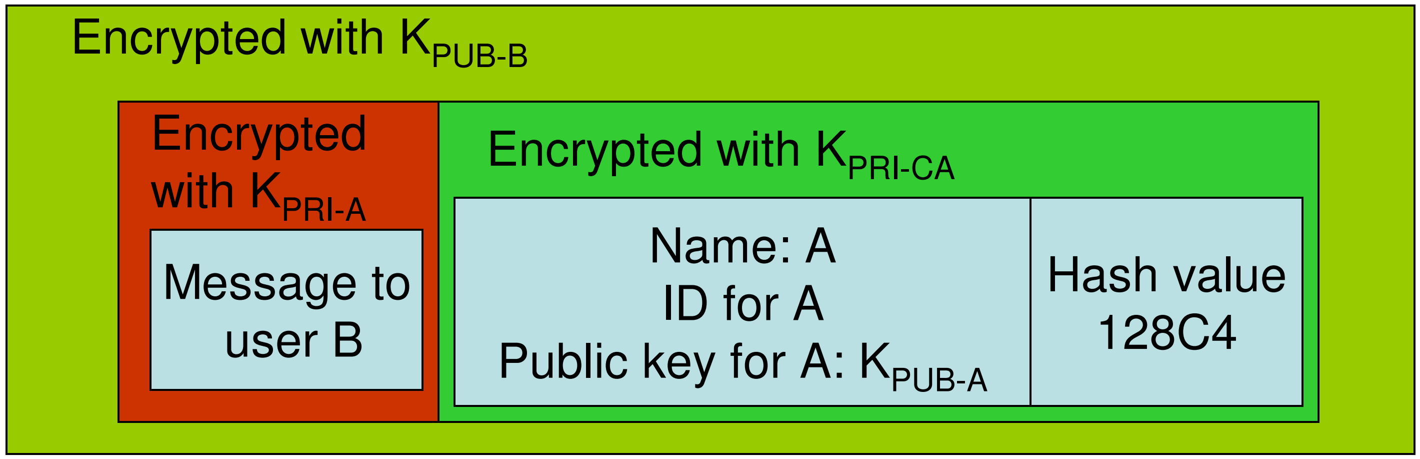
- X.509 is the universally accepted standard for the public-key certificate scheme.

Certificates

Digital Certificate for the Public Key of A



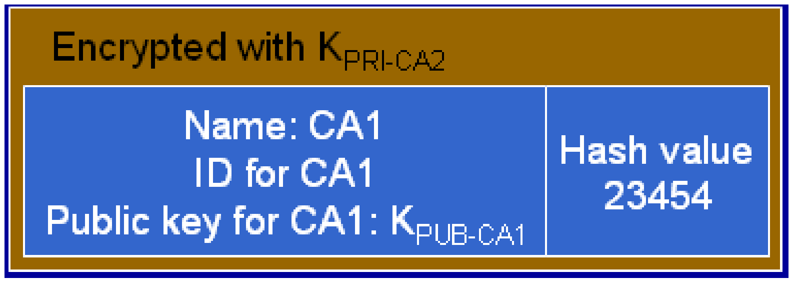
User A sending to user B


Encrypted with K_{PUB-B}
Encrypted with K_{PRI-A}
Message to user B
Encrypted with K_{PRI-CA}
Name: A
ID for A
Public key for A: K_{PUB-A}
Hash value
128C4

Note: The certificates are created and formatted based on the X.509 standard, which outlines the necessary fields of a certificate and the possible values that can be inserted into these fields. The latest X.509 version is v.3.

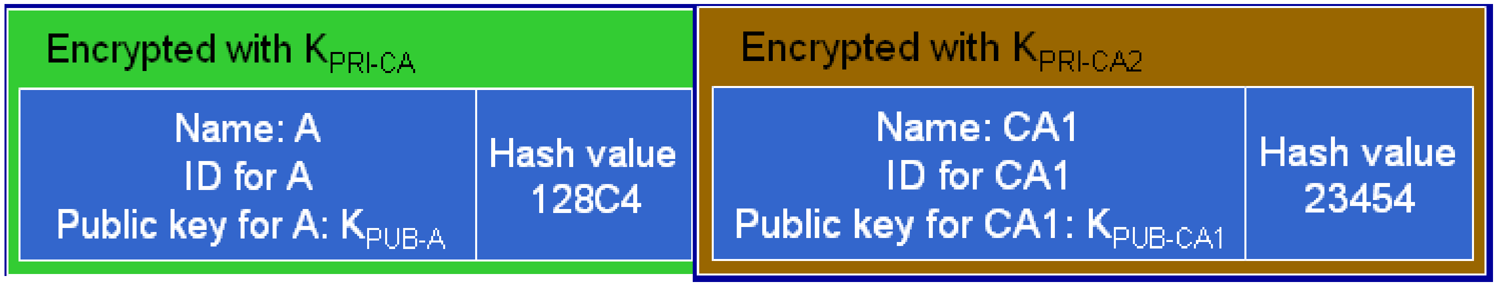
Certificates

Digital Certificate for the Public Key of CA1



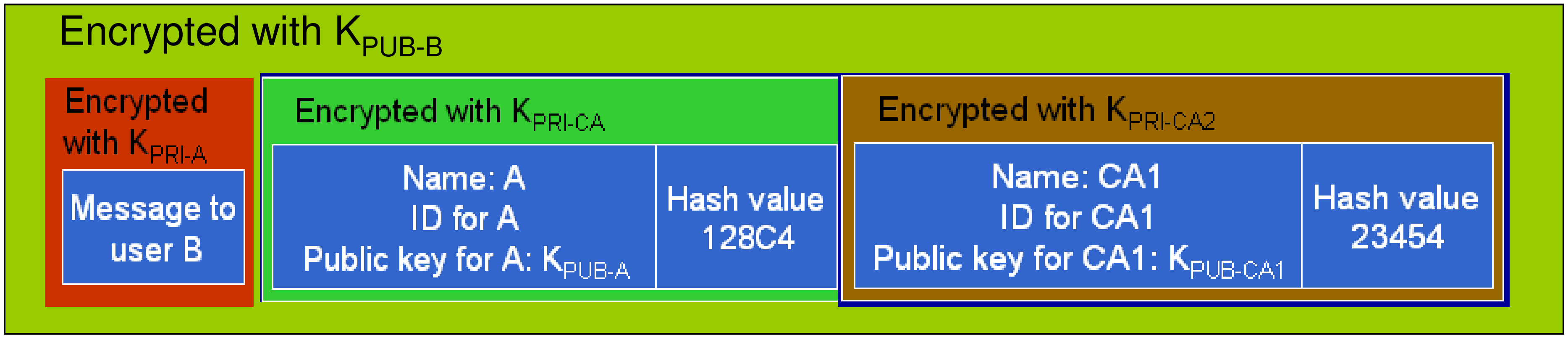
Encrypted with $K_{PRI-CA2}$
Name: CA1
ID for CA1
Public key for CA1: $K_{PUB-CA1}$
Hash value 23454

Digital Certificate for the Public Key of A



Encrypted with K_{PRI-CA}
Name: A
ID for A
Public key for A: K_{PUB-A}
Hash value 128C4
Encrypted with $K_{PRI-CA2}$
Name: CA1
ID for CA1
Public key for CA1: $K_{PUB-CA1}$
Hash value 23454

User A sending to user B

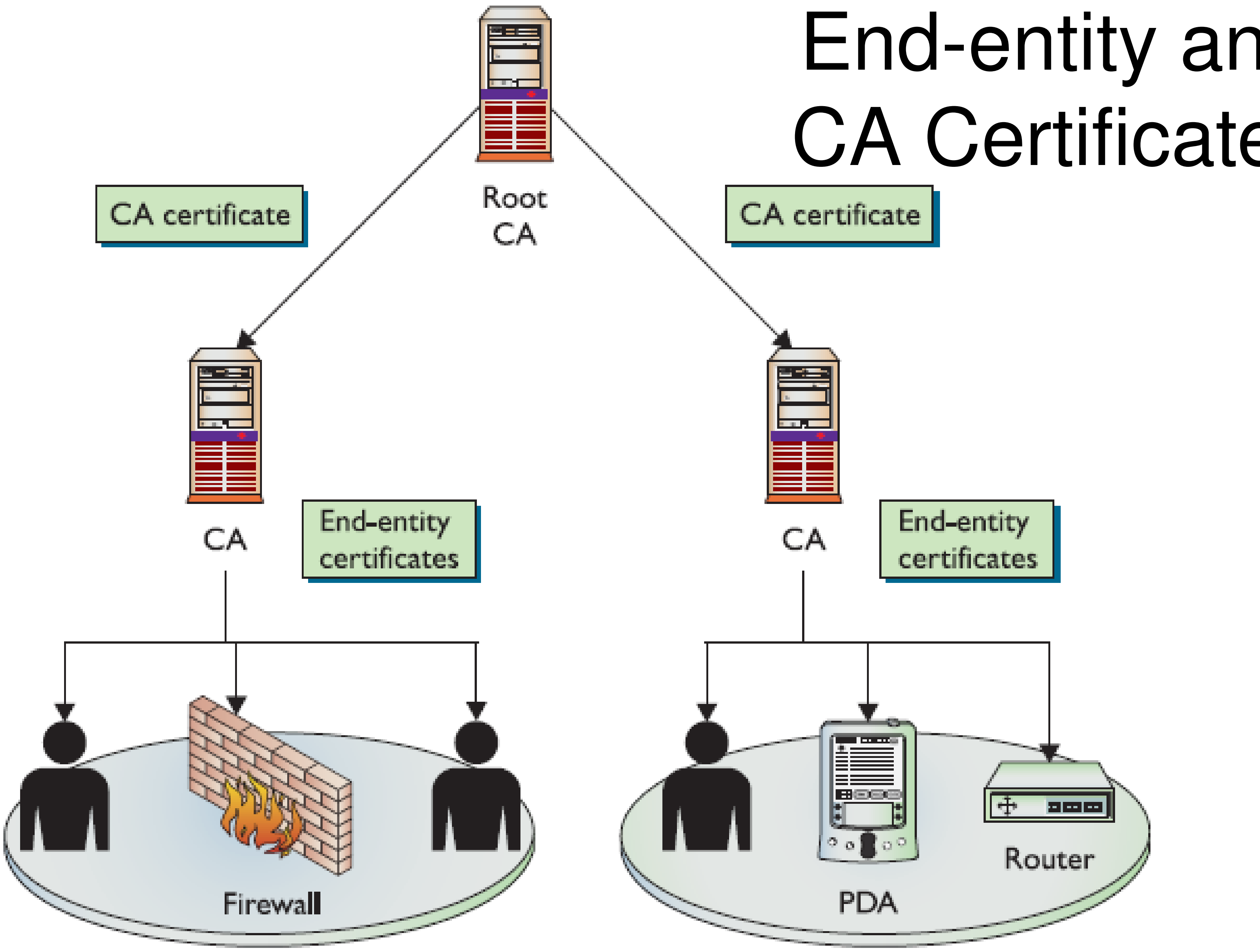


Encrypted with K_{PUB-B}
Encrypted with K_{PRI-A}
Message to user B
Encrypted with K_{PRI-CA}
Name: A
ID for A
Public key for A: K_{PUB-A}
Hash value 128C4
Encrypted with $K_{PRI-CA2}$
Name: CA1
ID for CA1
Public key for CA1: $K_{PUB-CA1}$
Hash value 23454

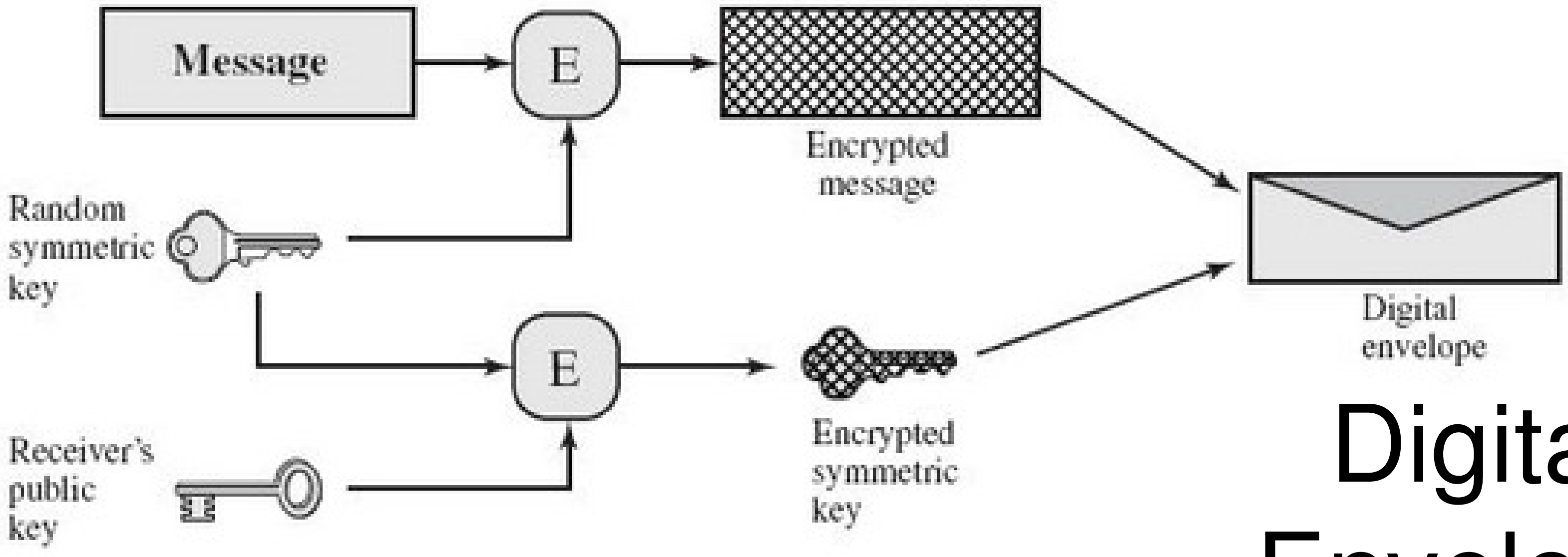
Classes of Digital Certificates

- The types of certificates available can vary between CAs; but, all CAs should at least support the following three classes of certificates:
- Class 1 – A Class 1 certificate is usually used to verify an individual's identity through e-mail. A person who receives a Class 1 certificate can use his public/ private key pair to digitally sign e-mail and encrypt message contents.
- Class 2 – A Class 2 certificate can be used for software signing. A software vendor would register for this type of certificate so that it could digitally sign the software. This provides integrity for the software after it is developed and released, and it allows the receiver software to verify where the software actually came from before installation.
- Class 3 – A Class 3 certificate can be used by a company to set up its own CA which will allow it to carry out its own identification verification and internally generate certificates.
- End-entity certificates (Class 1 and 2 are referred to as End-entity certificates)
- CA certificate – Class 3 can also be referred to as CA certificates.
- Note: An entity can have multiple public/ private key pairs and corresponding digital certificates, used for different purposes.

End-entity and CA Certificates



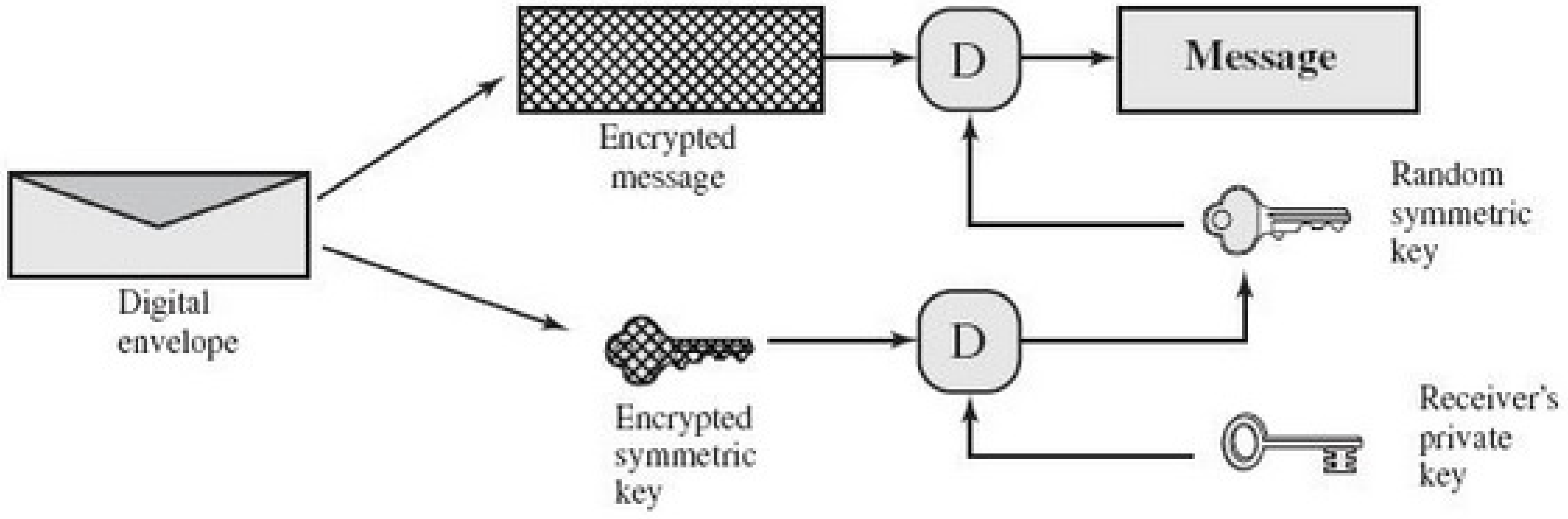
Source: Figure 6.8 from Conklin and White – Principles of Computer Security, 2nd Edition



(a) Creation of a digital envelope

Digital Envelope

<u>Idea:</u> Randomly generate a symmetric key on the fly; use it to encrypt the message and send it along with the encrypted symmetric key (encrypted with public key of receiver)



(b) Opening a digital envelope

Key Distribution

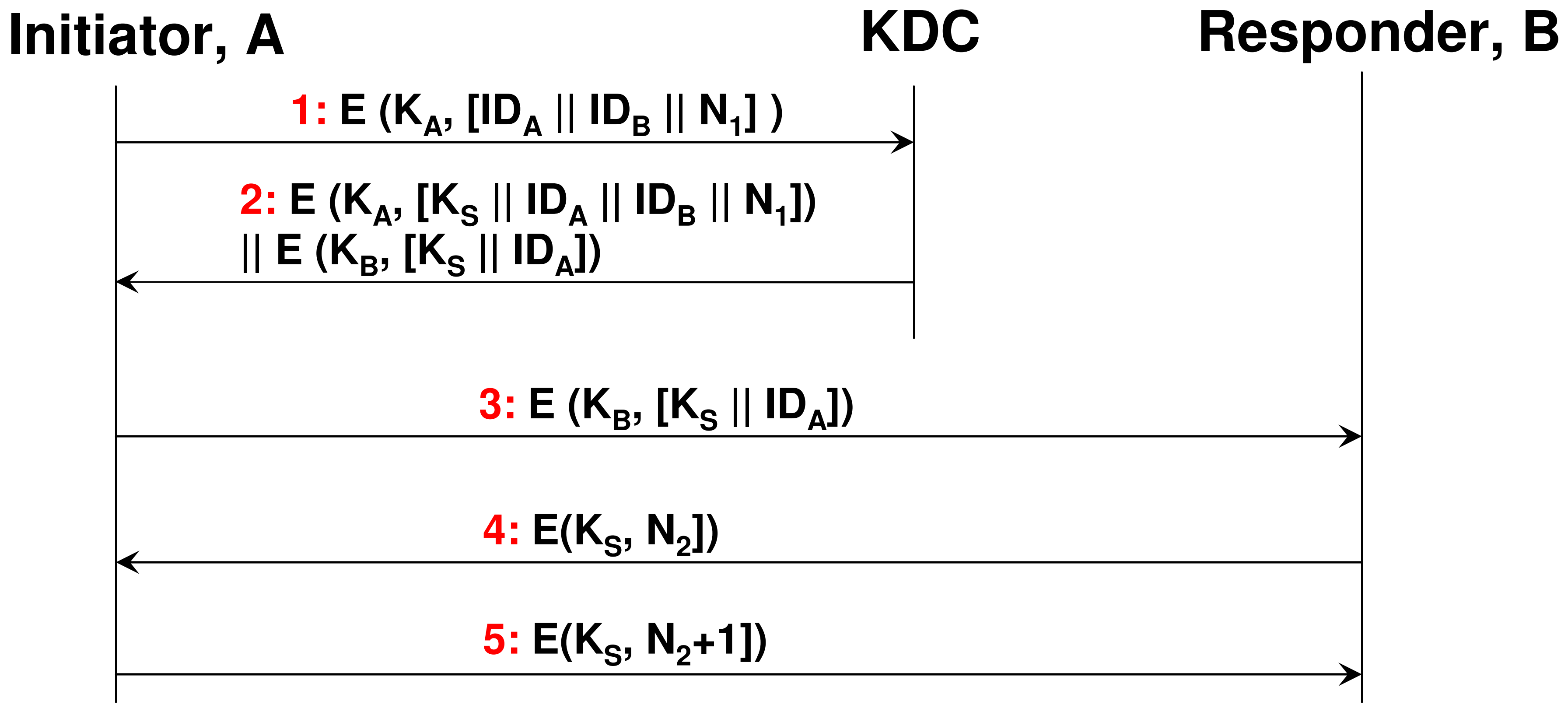
- Both symmetric key schemes and public key schemes require both parties to acquire valid keys.
- In symmetric key schemes, the shared key should be securely distributed between the source and destination, while protecting it from others.

- Frequent key changes are usually desirable to limit the amount of data compromised if an attacker learns the key.
- A new session key should be used for each new connection-oriented session. For a connectionless protocol, a new session key is used for a certain fixed period only or for a certain number of transactions.

- On many occasions systems have been broken, not because of a poor encryption algorithm, but because of poor key selection or management.
- Preferred Approach, especially for scalability - A third party, whom all parties trust, can be used as a trusted intermediary to mediate the establishment of secure communications between users.

Needham-Schroeder Protocol for Secure Key Distribution and Authentication

- We assume a Key Distribution Center (KDC) shares a unique key with each party/ user.



Initiator, A
KDC
Responder, B
1: $E(K_A, [ID_A \| ID_B \| N_1])$
2: $E(K_A, [K_S \| ID_A \| ID_B \| N_1])$
$\| E(K_B, [K_S \| ID_A])$
3: $E(K_B, [K_S \| ID_A])$
4: $E(K_S, N_2])$
5: $E(K_S, N_2+1])$

Note: Steps 1, 2 and 3 are related to "Key Distribution," while steps 3, 4 and 5 are related to providing "Authentication" for the initiator A at the responder B

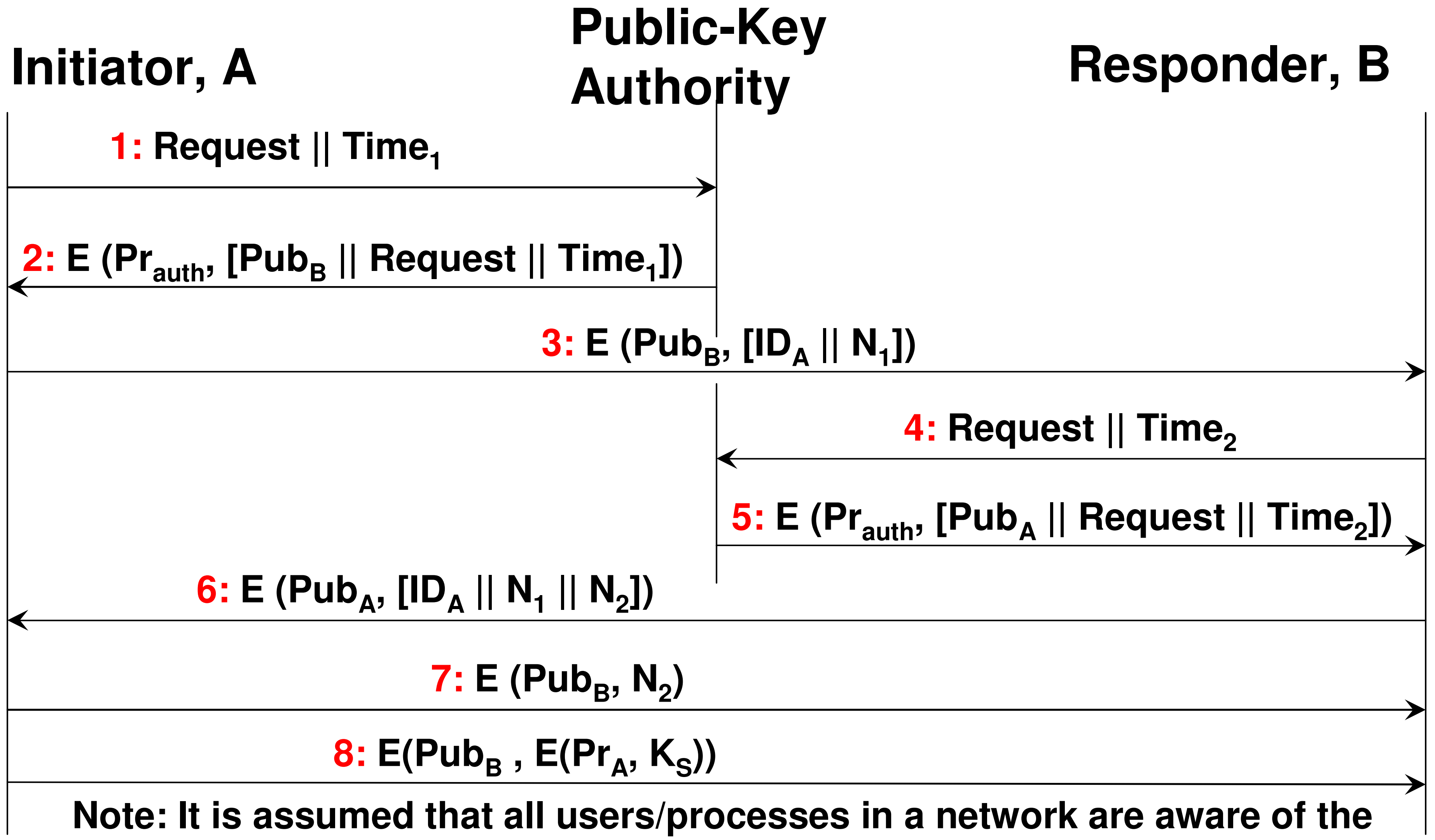
Key Distribution Issues

- For very large networks, a hierarchy of KDCs can be established.
- For communication among entities within the same local domain, the local KDC is responsible for key distribution. If two entities in different domains desire a shared key, then the corresponding local KDCs can communicate through a (hierarchy of) global KDC(s)
- The use of a key distribution center imposes the requirement that the KDC be trusted and be protected from subversion. This requirement can be avoided if key distribution is fully decentralized.

- <u>Hybrid Key Distribution Scheme:</u> This scheme retains the use of a key distribution center (KDC) that shares a secret master key with each user and distributes secret session keys encrypted with the master key. A public key scheme is used to distribute the master keys.
- The addition of a public-key layer provides a secure, efficient means of distributing master keys.

- <u>Distribution of Public Keys:</u> Using Public-Key Authority (centralized approach, needs real-time access to the authority) and Public-Key Certificates (no need for real-time access to the certificate authority).

Distributing Public Keys and Secret Keys using Public-Key Authority


Initiator, A
Public-Key Authority
Responder, B
1: Request || $Time_1$
2: E (Pr_{auth}, [Pub_B || Request || $Time_1$])
3: E (Pub_B, [ID_A || N_1])
4: Request || $Time_2$
5: E (Pr_{auth}, [Pub_A || Request || $Time_2$])
6: E (Pub_A, [ID_A || N_1 || N_2])
7: E (Pub_B, N_2)
8: E(Pub_B , E(Pr_A, K_S))

Note: It is assumed that all users/processes in a network are aware of the public key of the Public-Key Authority.

Modular Arithmetic

- Modular Exponentiation
  - The Right-to-Left Binary Algorithm

To compute b^e mod n

First, write the exponent e in binary notation.

$$e = \sum_{i=0}^{m-1} a_i 2^i$$

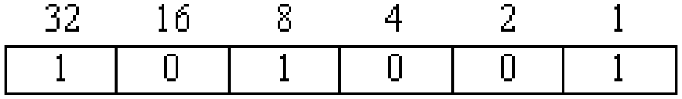
In this notation, the length of e is m bits. For any i, such that $0 \le i < m-1$, the a_i take the value of 0 or 1. By definition, $a_{m-1} = 1$.

$$b^e = b^{\left(\sum_{i=0}^{m-1} a_i 2^i\right)} = \prod_{i=0}^{m-1} \left(b^{2^i}\right)^{a_i}$$

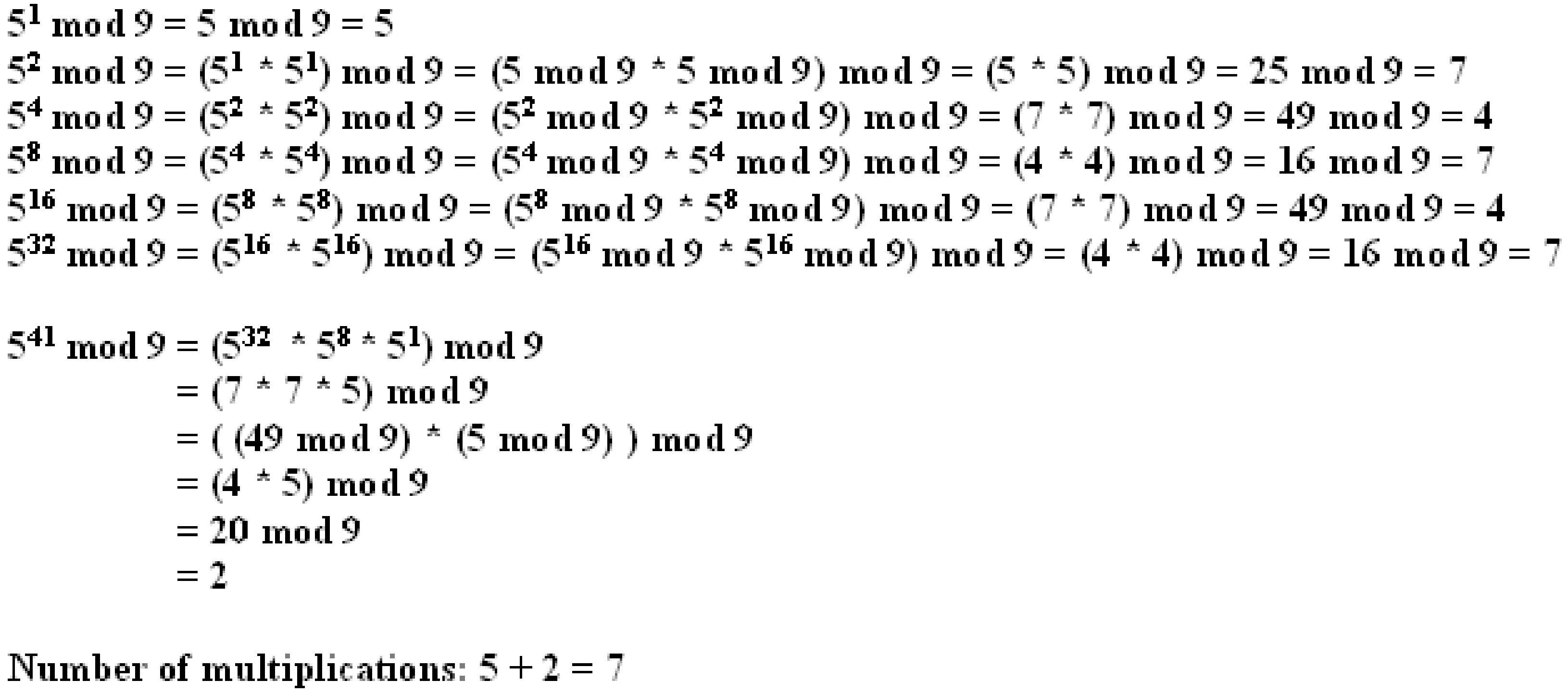
Solution for b^e mod n = $\prod_{i=0}^{m-1} \left(b^{2^i}\right)^{a_i} \bmod n$

Example for Modular Exponentiation

- To compute 5^{41} mod 9
  - Straightforward approach:
    - 5^{41} mod 9 = (45474735088646411895751953125) mod 9 = 2
    - Number of multiplications - 40
  - Using the Right-to-Left Binary Algorithm
    - Write 41 in binary: 101001
    - $5^{41} = 5^{32} * 5^{8} * 5^{1}$



32	16	8	4	2	1
1	0	1	0	0	1



5^{1} mod 9 = 5 mod 9 = 5

5^{2} mod 9 = (5^{1} * 5^{1}) mod 9 = (5 mod 9 * 5 mod 9) mod 9 = (5 * 5) mod 9 = 25 mod 9 = 7

5^{4} mod 9 = (5^{2} * 5^{2}) mod 9 = (5^{2} mod 9 * 5^{2} mod 9) mod 9 = (7 * 7) mod 9 = 49 mod 9 = 4

5^{8} mod 9 = (5^{4} * 5^{4}) mod 9 = (5^{4} mod 9 * 5^{4} mod 9) mod 9 = (4 * 4) mod 9 = 16 mod 9 = 7

5^{16} mod 9 = (5^{8} * 5^{8}) mod 9 = (5^{8} mod 9 * 5^{8} mod 9) mod 9 = (7 * 7) mod 9 = 49 mod 9 = 4

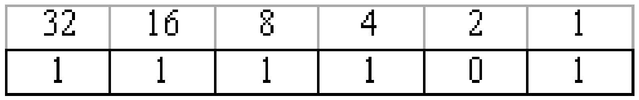
5^{32} mod 9 = (5^{16} * 5^{16}) mod 9 = (5^{16} mod 9 * 5^{16} mod 9) mod 9 = (4 * 4) mod 9 = 16 mod 9 = 7

5^{41} mod 9 = (5^{32} * 5^{8} * 5^{1}) mod 9
= (7 * 7 * 5) mod 9
= ((49 mod 9) * (5 mod 9)) mod 9
= (4 * 5) mod 9
= 20 mod 9
= 2

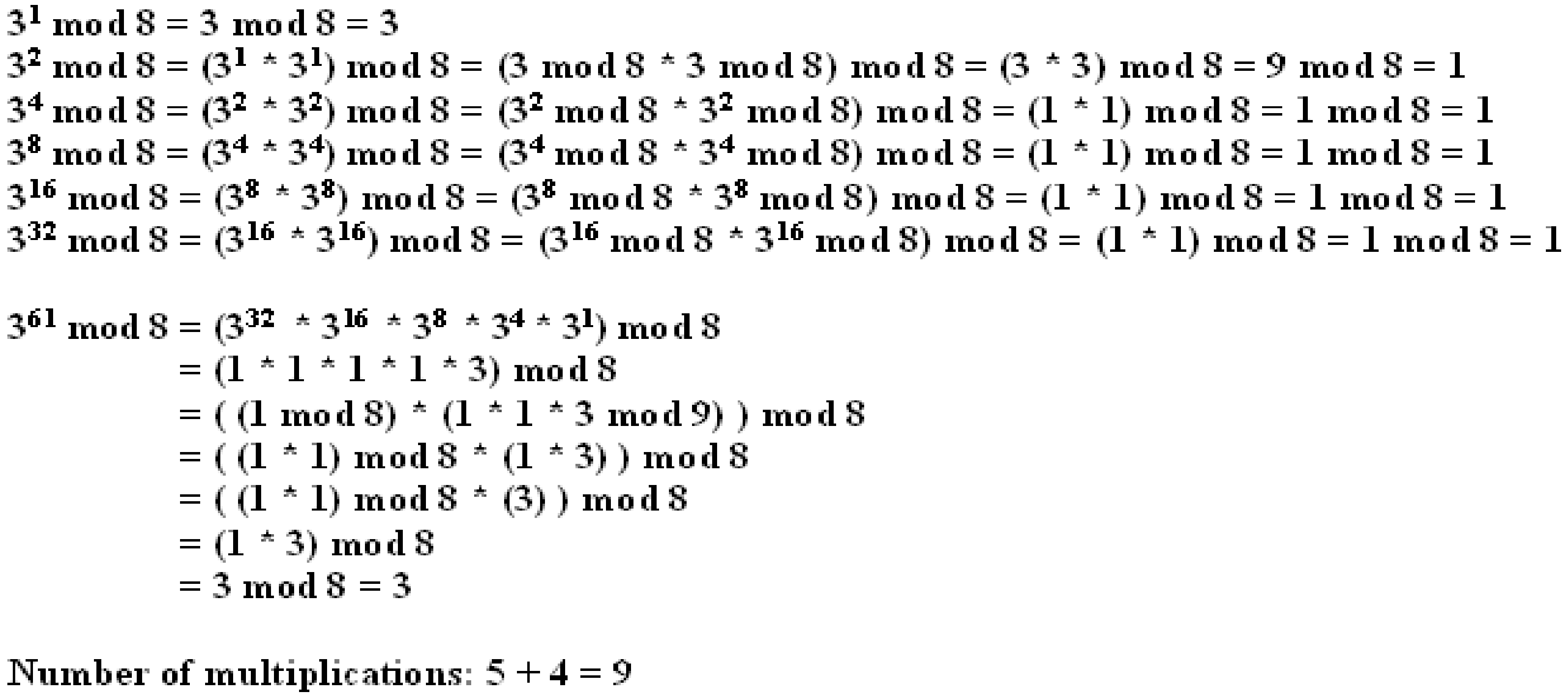
Number of multiplications: 5 + 2 = 7

Example for Modular Exponentiation

- To compute 3^{61} mod 8
  - Straightforward approach:
    - 3^{61} mod 8 = (127173478256486195428832996O3) mod 8 = 3
    - Number of multiplications - 60
  - Using the Right-to-Left Binary Algorithm
    - Write 61 in binary: 111101
    - $3^{41} = 3^{32} * 3^{16} * 3^{8} * 3^{4} * 3^{1}$



32	16	8	4	2	1
1	1	1	1	0	1



3^1 mod 8 = 3 mod 8 = 3
3^2 mod 8 = (3^1 * 3^1) mod 8 = (3 mod 8 * 3 mod 8) mod 8 = (3 * 3) mod 8 = 9 mod 8 = 1
3^4 mod 8 = (3^2 * 3^2) mod 8 = (3^2 mod 8 * 3^2 mod 8) mod 8 = (1 * 1) mod 8 = 1 mod 8 = 1
3^8 mod 8 = (3^4 * 3^4) mod 8 = (3^4 mod 8 * 3^4 mod 8) mod 8 = (1 * 1) mod 8 = 1 mod 8 = 1
3^{16} mod 8 = (3^8 * 3^8) mod 8 = (3^8 mod 8 * 3^8 mod 8) mod 8 = (1 * 1) mod 8 = 1 mod 8 = 1
3^{32} mod 8 = (3^{16} * 3^{16}) mod 8 = (3^{16} mod 8 * 3^{16} mod 8) mod 8 = (1 * 1) mod 8 = 1 mod 8 = 1

3^{61} mod 8 = (3^{32} * 3^{16} * 3^8 * 3^4 * 3^1) mod 8
= (1 * 1 * 1 * 1 * 3) mod 8
= ((1 mod 8) * (1 * 1 * 3 mod 9)) mod 8
= ((1 * 1) mod 8 * (1 * 3)) mod 8
= ((1 * 1) mod 8 * (3)) mod 8
= (1 * 3) mod 8
= 3 mod 8 = 3

Number of multiplications: 5 + 4 = 9

Applications of Encryption

- Diffie-Hellman Key Exchange
  - Used to allow two parties that have to establish a shared secret key over an insecure communication channel.
  - Alice and Bob agree on a field size n and a starting number g.
  - Alice generates a secret integer a and sends g^a mod n to Bob. Alice sends this encrypted using its private key, so that Bob can decrypt it using Alice's public key, thereby authenticating that the message came from Alice. E($K_{PRI-ALICE}$, g^a mod n)
  - At the same time, Bob generates a secret integer b and sends g^b mod n to Alice. Bob sends this encrypted using its private key, thereby authenticating to Alice that the message came from Bob. E($K_{PRI-Bob}$, g^b mod n)
  - When Bob gets Alice's message, it computes $(g^a)^b$ mod n and uses it as the secret key.
  - Similarly, when Alice gets Bob's message, it computes $(g^b)^a$ mod n and uses it as the secret key.
  - According to Modular arithmetic, $(g^a)^b$ mod n = $(g^b)^a$ mod n. Hence, both Alice and Bob have agreed on a shared secret key.

Example for Diffie-Hellman Key Exchange

- Assume the secret integers used by Alice and Bob to be 15 and 29 respectively. The values of *g* and *n* are 13 and 45 respectively. What would be the secret key they will be agreeing with?

$\boxed{g = 13;\ n = 45;\ a = 15;\ b = 29}$

Alice Side

	8	4	2	1
15 is:	1	1	1	1

Compute $g^a \bmod n = 13^{15} \bmod 45$

$13^1 \bmod 45 = 13$
$13^2 \bmod 45 = (13^1 \bmod 45 * 13^1 \bmod 45) = 169 \bmod 45 = 34$
$13^4 \bmod 45 = (13^2 \bmod 45 * 13^2 \bmod 45) = (34*34) \bmod 45 = 31$
$13^8 \bmod 45 = (13^4 \bmod 45 * 13^4 \bmod 45) = (31*31) \bmod 45 = 16$

$$\begin{aligned} 13^{15} \bmod 45 &= (13^8 * 13^4 * 13^2 * 13^1) \bmod 45 = (16 * 31 * 34 * 13) \bmod 45 \\ &= (1 * 34 * 13) \bmod 45 \\ &= 37 \end{aligned}$$

Example for Diffie-Hellman Key Exchange (continued…)

$\boxed{g = 13;\ n = 45;\ a = 15;\ b = 29}$

	16	8	4	2	1
29 is:	1	1	1	0	1

Alice sends 37 to Bob

Bob computes $(g^{a} \bmod n)^{b} \bmod n = 37^{29} \bmod 45$

$37^{1} \bmod 45 = 37$
$37^{2} \bmod 45 = (37^{1} \bmod 45 * 37^{1} \bmod 45) = 19$
$37^{4} \bmod 45 = (37^{2} \bmod 45 * 37^{2} \bmod 45) = (19*19) \bmod 45 = 1$
$37^{8} \bmod 45 = (37^{4} \bmod 45 * 37^{4} \bmod 45) = (1*1) \bmod 45 =1$
$37^{16} \bmod 45 = (37^{8} \bmod 45 * 37^{8} \bmod 45) = (1*1) \bmod 45 =1$

$37^{29} \bmod 45 = (37^{16} * 37^{8} * 37^{4} * 37^{1}) \bmod 45 = (1 * 1 * 1 * 37) \bmod 45$
$= 37$

Example for Diffie-Hellman Key Exchange (continued…)

- Assume the secret integers used by Alice and Bob to be 15 and 29 respectively. The values of *g* and *n* are 13 and 45 respectively. What would be the secret key they will be agreeing with?

$\boxed{g = 13; n = 45; a = 15; b = 29}$

Bob Side

	16	8	4	2	1
29 is:	1	1	1	0	1

Compute $g^a \bmod n = 13^{29} \bmod 45$

$13^1 \bmod 45 = 13$
$13^2 \bmod 45 = (13^1 \bmod 45 * 13^1 \bmod 45) = 19$
$13^4 \bmod 45 = (13^2 \bmod 45 * 13^2 \bmod 45) = (19*19) \bmod 45 = 1$
$13^8 \bmod 45 = (13^4 \bmod 45 * 13^4 \bmod 45) = (1*1) \bmod 45 =1$
$13^{16} \bmod 45 = (13^8 \bmod 45 * 13^8 \bmod 45) = (1*1) \bmod 45 =1$

$13^{29} \bmod 45 = (13^{16} * 13^8 * 13^4 * 13^1) \bmod 45 = (1 * 1 * 1 * 13) \bmod 45$
$= 13$

Example for Diffie-Hellman Key Exchange (continued…)

$\boxed{g = 13;\ n = 45;\ a = 15;\ b = 29}$

	8	4	2	1
15 is:	1	1	1	1

Bob sends 13 to Alice

Alice computes $(g^b \bmod n)^a \bmod n = 13^{15} \bmod 45$

$13^1 \bmod 45 = 13$
$13^2 \bmod 45 = (13^1 \bmod 45 * 13^1 \bmod 45) = 169 \bmod 45 = 34$
$13^4 \bmod 45 = (13^2 \bmod 45 * 13^2 \bmod 45) = (34*34) \bmod 45 = 31$
$13^8 \bmod 45 = (13^4 \bmod 45 * 13^4 \bmod 45) = (31*31) \bmod 45 = 16$

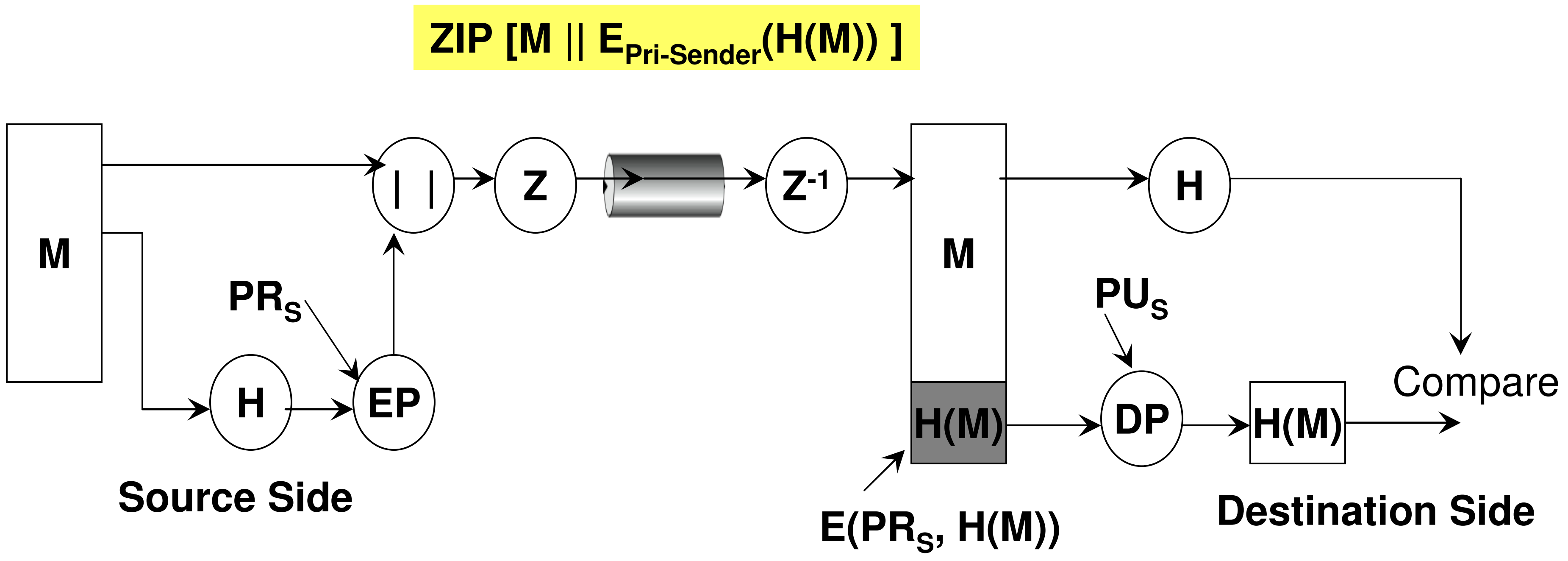
$$\begin{aligned} 13^{15} \bmod 45 &= (13^8 * 13^4 * 13^2 * 13^1) \bmod 45 = (16 * 31 * 34 * 13) \bmod 45 \\ &= (1 * 34 * 13) \bmod 45 \\ &= 37 \end{aligned}$$

37 is the **session key** agreed upon by both sides

Motivation for E-mail Security

- Electronic mail (E-mail) is one of the widely used and regarded network –based application in virtually all distributed environments.
- Currently, message contents are not secure.
  - May be inspected either in transit or by suitably privileged users on destination systems
- Requirements for E-mail Security
  - Confidentiality: protection from disclosure
  - Authentication of sender of message
  - Message integrity: protection from modification
  - Non-repudiation of origin: protection from denial by sender
- PGP (Pretty Good Privacy) and S/MIME (Secure Multi-purpose Internet Mail Extensions) are the two commonly used E-mail Security Standards

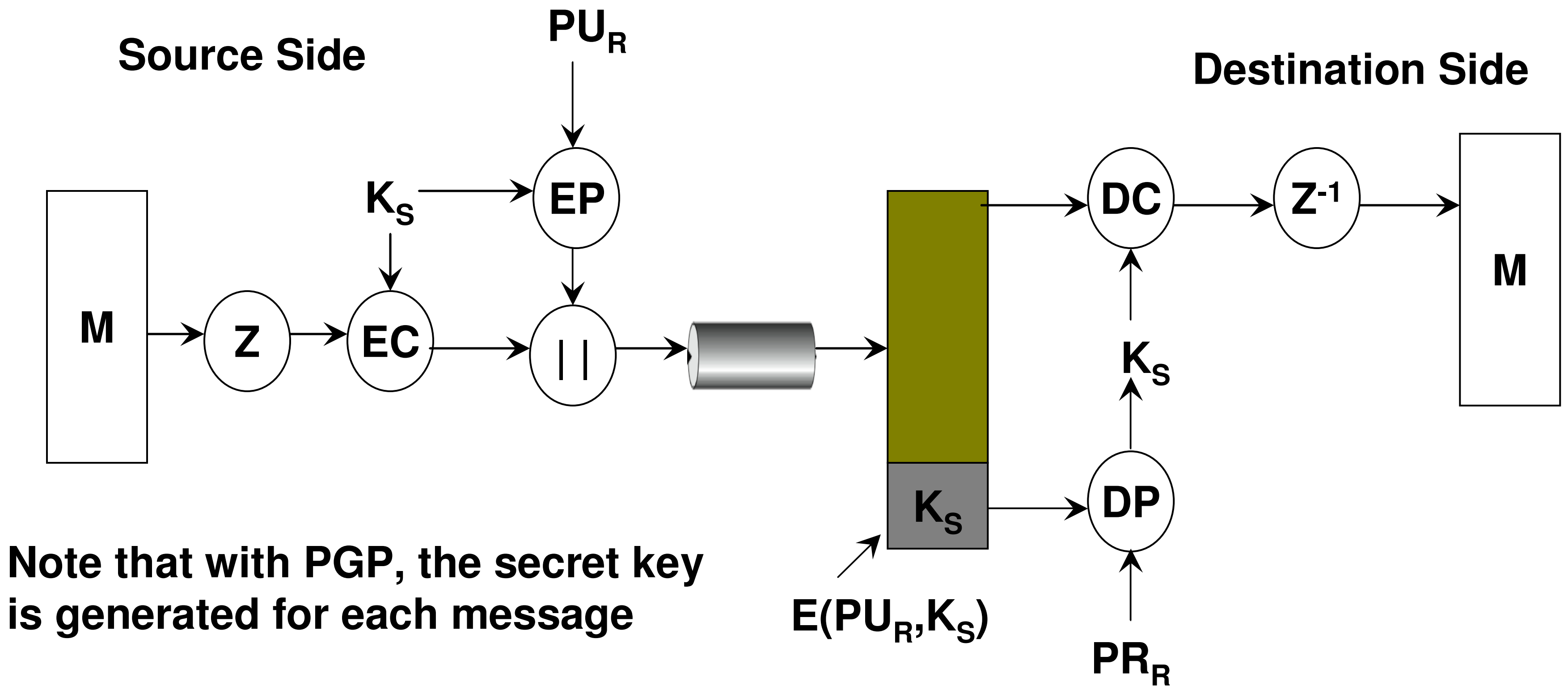
PGP for Authentication



Hashing Algorithm, H - SHA-1160 bit hash
Encryption for digital signature (public-key encryption), EP – RSA
Compression, Z – ZIP algorithm

Source (adapted from): Figure 18.1 (a), from William Stallings – Cryptography and Network Security, 5th Edition

PGP for Confidentiality



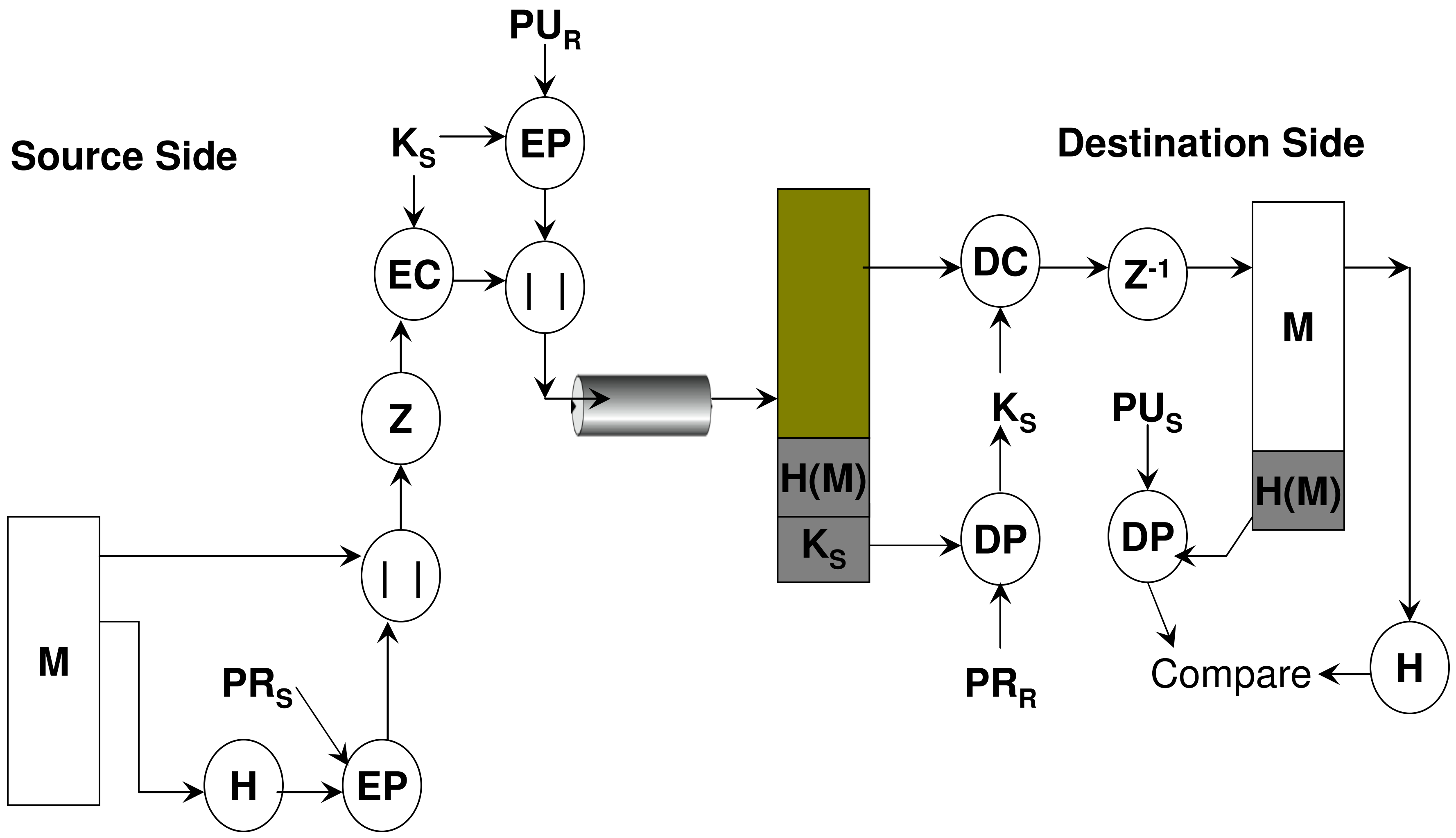
Note that with PGP, the secret key is generated for each message

$E_{Secret\text{-}key}[ZIP\ (M)]$ || $E_{Pub\text{-}Receiver}(Secret\text{-}key)$

Encryption for confidentiality, EC – IDEA (International Data Encryption Algorithm)

Source (adapted from): Figure 18.1 (b), from William Stallings – Cryptography and Network Security, 5th Edition

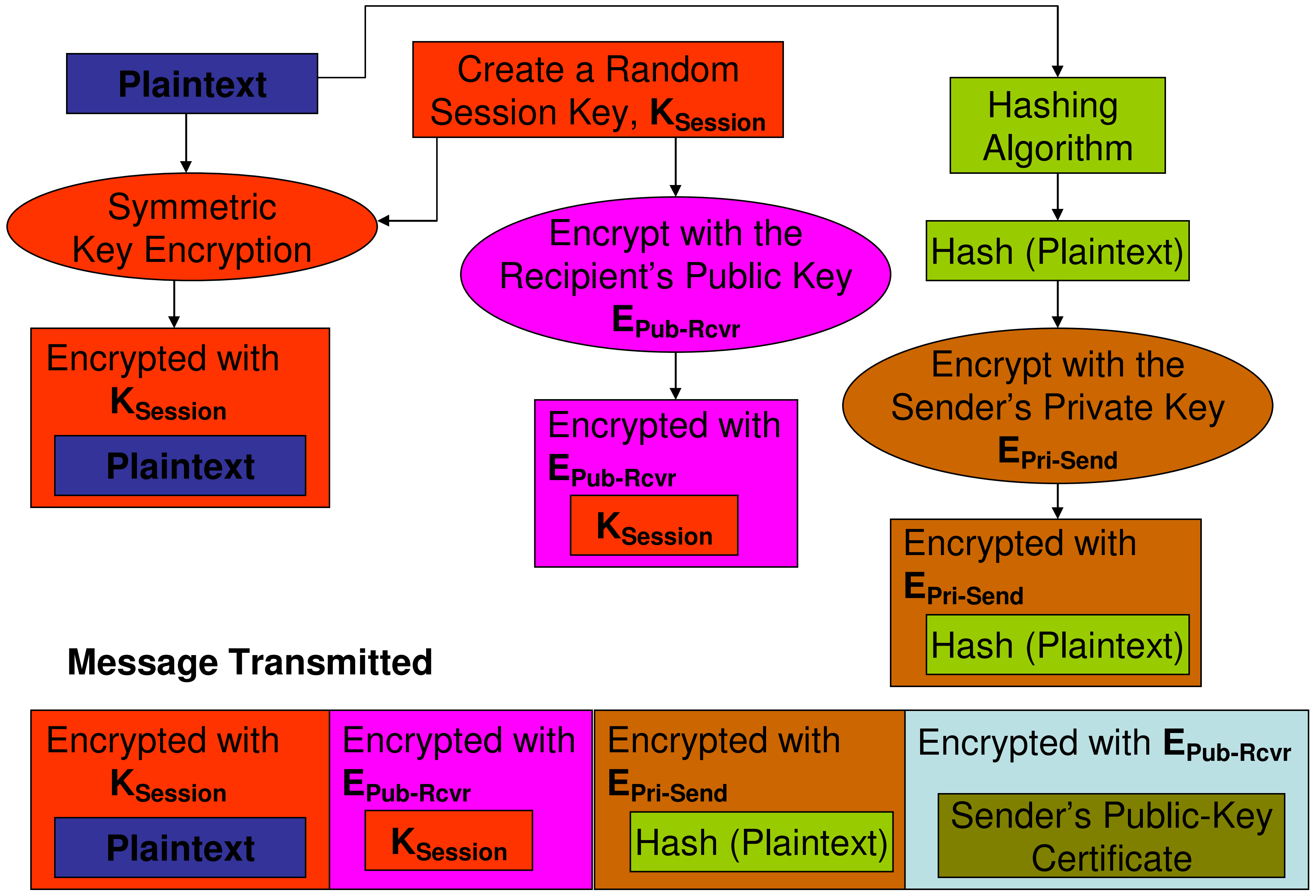
PGP for Authentication and Confidentiality



$E_{\text{Secret-key}}[\text{ZIP }(M \,||\, E_{\text{Pri-Sender}}(H(M))\,)] \;||\; E_{\text{Pub-Receiver}}(\text{Secret-key})$

Source (adapted from): Figure 18.1 (c), from William Stallings – Cryptography and Network Security, 5th Edition

S/MIME



Plaintext
Create a Random Session Key, $K_{Session}$
Hashing Algorithm
Symmetric Key Encryption
Encrypt with the Recipient's Public Key $E_{Pub-Rcvr}$
Hash (Plaintext)
Encrypted with $K_{Session}$
Plaintext
Encrypt with the Sender's Private Key $E_{Pri-Send}$
Encrypted with $E_{Pub-Rcvr}$
$K_{Session}$
Encrypted with $E_{Pri-Send}$
Hash (Plaintext)
Message Transmitted
Encrypted with $K_{Session}$
Plaintext
Encrypted with $E_{Pub-Rcvr}$
$K_{Session}$
Encrypted with $E_{Pri-Send}$
Hash (Plaintext)
Encrypted with $E_{Pub-Rcvr}$
Sender's Public-Key Certificate