OVER 40 YEARS OF ACADEMIC WISDOM

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

PRESIDENCY UNIVERSITY

CSE3078 – Cryptography and Network Security

School of Computer Science and Engineering

Module 3

Public Key Cryptography and its Applications

Private-Key Cryptography

- traditional **private/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed communications are compromised
- also is **symmetric**, parties are equal
- hence does not protect sender from receiver forging a message & claiming is sent by sender

Public-Key Cryptography

- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- uses clever application of number theoretic concepts to function
- complements **rather than** replaces private key crypto

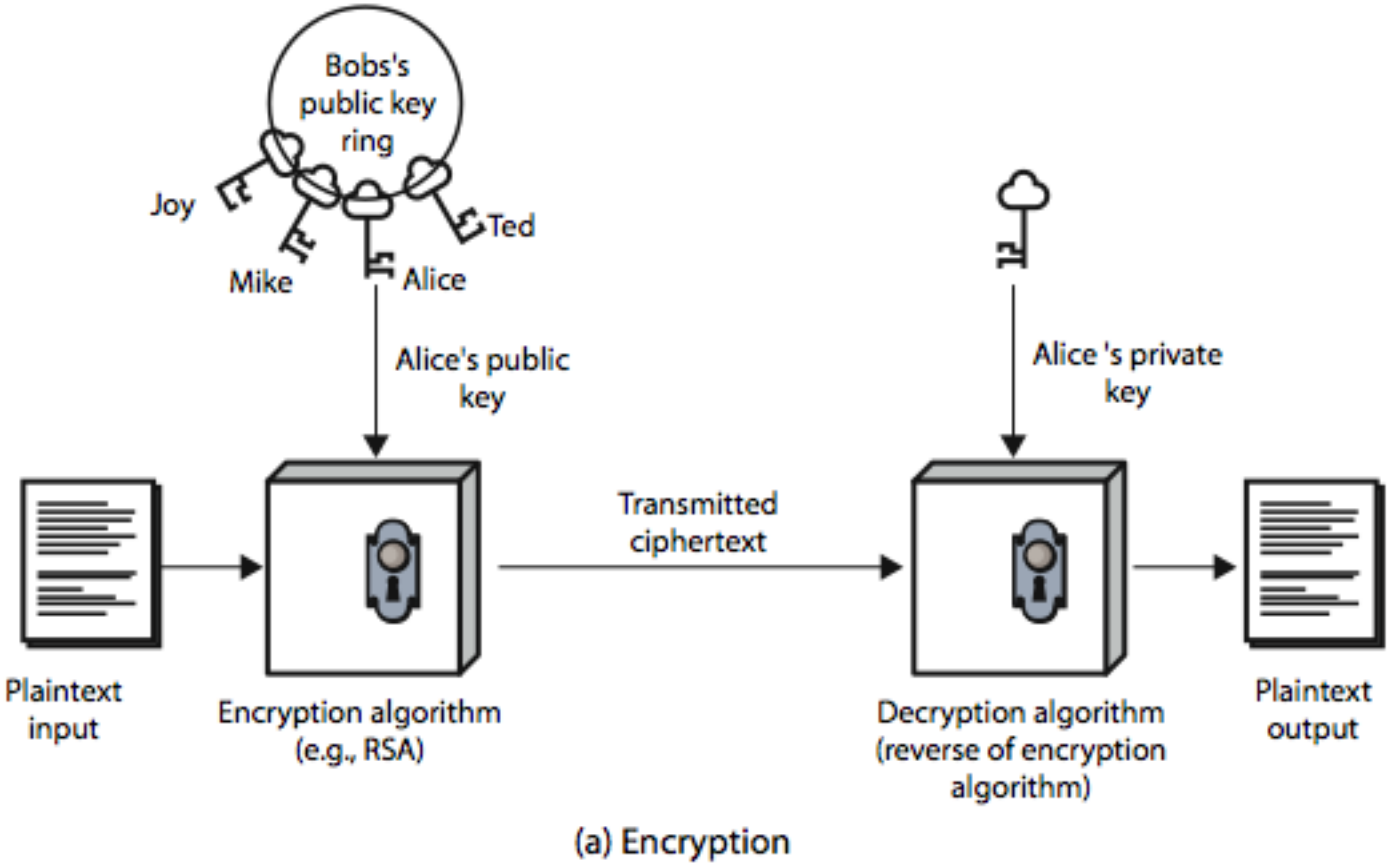
Why Public-Key Cryptography?

- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in classified community

Public-Key Cryptography

- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

Public-Key Cryptography

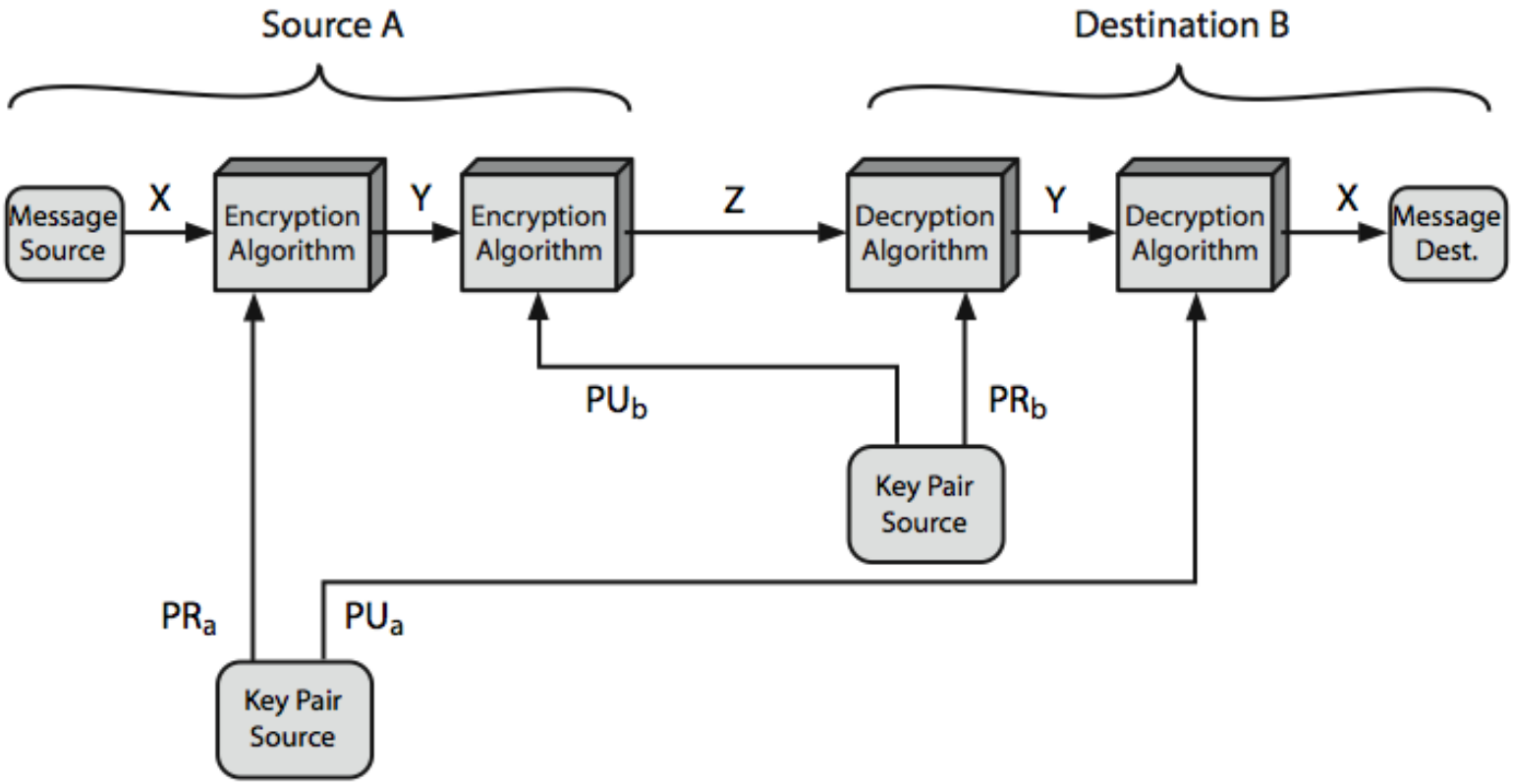


(a) Encryption

Public-Key Characteristics

- Public-Key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm & encryption key
  - it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)

Public-Key Cryptosystems



Public-Key Applications

- can classify uses into 3 categories:
  - **encryption/decryption** (provide secrecy)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)
- some algorithms are suitable for all uses, others are specific to one

Security of Public Key Schemes

- like private key schemes brute force **exhaustive search** attack is always theoretically possible
- but keys used are too large (>512bits)
- security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalyse) problems
- more generally the **hard** problem is known, but is made hard enough to be impractical to break
- requires the use of **very large numbers**
- hence is **slow** compared to private key schemes

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - nb. exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random - p, q
- computing their system modulus n=p.q
  - note ø(n)=(p-1)(q-1)
- selecting at random the encryption key e
    - where 1<e<ø(n), gcd(e,ø(n))=1
- solve following equation to find decryption key d
  - e.d=1 mod ø(n) and 0≤d≤n
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}

RSA Use

- to encrypt a message M the sender:
  - obtains **public key** of recipient PU={e,n}
  - computes: $C = M^e \bmod n$, where $0 \leq M < n$
- to decrypt the ciphertext C the owner:
  - uses their private key PR={d,n}
  - computes: $M = C^d \bmod n$
- note that the message M must be smaller than the modulus n (block if needed)

Why RSA Works

- because of Euler's Theorem:
  - $a^{\phi(n)} \bmod n = 1$ where $\gcd(a,n)=1$
- in RSA have:
  - $n=p.q$
  - $\phi(n)=(p-1)(q-1)$
  - carefully chose e & d to be inverses $\bmod \phi(n)$
  - hence $e.d=1+k.\phi(n)$ for some k
- hence :

$$C^d = M^{e.d} = M^{1+k.\phi(n)} = M^1.(M^{\phi(n)})^k$$
$$= M^1.(1)^k = M^1 = M \bmod n$$

RSA Example - Key Setup

1. Select primes: p=17 & q=11
2. Compute $n = pq$ =17 x 11=187
3. Compute ø(n)=(p–1)(q-1)=16 x 10=160
4. Select e: gcd(e,160)=1; choose e=7
5. Determine d: de=1 mod 160 and d < 160
   Value is d=23 since 23x7=161= 10x160+1
6. Publish public key PU={7,187}
7. Keep secret private key PR={23,187}

RSA Example - En/Decryption

- sample RSA encryption/decryption is:
- given message $M = 88$ (nb. $88<187$)
- encryption:

  $C = 88^7 \bmod 187 = 11$

- decryption:

  $M = 11^{23} \bmod 187 = 88$

Exponentiation

- can use the Square and Multiply Algorithm
- a fast, efficient algorithm for exponentiation
- concept is based on repeatedly squaring base
- and multiplying in the ones that are needed to compute the result
- look at binary representation of exponent
- only takes $O(\log_2 n)$ multiples for number n
  - eg. $7^5 = 7^4.7^1 = 3.7 = 10 \bmod 11$
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \bmod 11$

Exponentiation

```
c = 0; f = 1
for i = k downto 0
    do c = 2 x c
       f = (f x f) mod n
    if b_i == 1 then
       c = c + 1
       f = (f x a) mod n
return f
```

Efficient Encryption

- encryption uses exponentiation to power e
- hence if e small, this will be faster
  - often choose e=65537 (2^{16}-1)
  - also see choices of e=3 or e=17
- but if e too small (eg e=3) can attack
  - using Chinese remainder theorem & 3 messages with different modulii
- if e fixed must ensure gcd(e,ø(n))=1
  - ie reject any p or q not relatively prime to e

Efficient Decryption

- decryption uses exponentiation to power d
  - this is likely large, insecure if not
- can use the Chinese Remainder Theorem (CRT) to compute mod p & q separately. then combine to get desired answer
  - approx 4 times faster than doing directly
- only owner of private key who knows values of p & q can use this technique

RSA Key Generation

- users of RSA must:
  - determine two primes at random - `p, q`
  - select either `e` or `d` and compute the other
- primes `p,q` must not be easily derived from modulus `n=p.q`
  - means must be sufficiently large
  - typically guess and use probabilistic test
- exponents `e, d` are inverses, so use Inverse algorithm to compute the other

RSA Security

- possible approaches to attacking RSA are:
  - **Brute force**: This involves trying all possible private keys.
  - **Mathematical attacks**: There are several approaches, all equivalent in effort to factoring the product of two primes.
  - **Timing attacks**: These depend on the running time of the decryption algorithm.
  - **Chosen ciphertext attacks**: This type of attack exploits properties of the RSA algorithm.

Factoring Problem

- mathematical approach takes 3 forms:
  - factor n=p.q, hence compute ø(n) and then d
  - determine ø(n) directly and compute d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663) bit with LS
  - biggest improvement comes from improved algorithm
    - cf QS to GHFS to LS
  - currently assume 1024-2048 bit RSA is secure
    - ensure p, q of similar size and matching other constraints

Timing Attacks

- developed by Paul Kocher in mid-1990's
- exploit timing variations in operations
  - eg. multiplying by small vs large number
  - or IF's varying which instructions executed
- infer operand size based on time taken
- RSA exploits time taken in exponentiation
- countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations

Chosen Ciphertext Attacks

- RSA is vulnerable to a Chosen Ciphertext Attack (CCA)
- Attackers chooses ciphertexts & gets decrypted plaintext back
- Choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis
- To counter with random pad of plaintext
or use Optimal Asymmetric Encryption Padding (OASP)

Diffie-Hellman Key Exchange

- The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages.
- The algorithm itself is limited to the exchange of secret values.

Diffie-Hellman key exchange algorithm

- For this scheme, there are two publicly known numbers: a prime number and an integer α that is a primitive root of .
- Suppose the users A and B wish to exchange a key.
- User A selects a random integer $X_A < q$ and computes

$$Y_A = \alpha^{X_A} \bmod q$$

- Similarly, user B independently selects a random integer $X_B < q$ and computes

$$Y_B = \alpha^{X_B} \bmod q$$

- Each side keeps the X value private and makes the Y value available publicly to the other side.
- User A computes the key as $K = (Y_B)^{X_A} \bmod q$

and user B computes the key as $K = (Y_A)^{X_B} \bmod q$

- These two calculations produce identical results

Example

Here is an example. Key exchange is based on the use of the prime number $q = 353$ and a primitive root of 353, in this case $\alpha = 3$. A and B select secret keys $X_A = 97$ and $X_B = 233$, respectively. Each computes its public key:

A computes $Y_A = 3^{97} \bmod 353 = 40$.

B computes $Y_B = 3^{233} \bmod 353 = 248$.

After they exchange public keys, each can compute the common secret key:

A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.

B computes $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160$.

- To assume an attacker would have available the following information:

  $q = 353$; $a = 3$; $Y_A = 40$; $Y_B = 248$

- In this simple example, it would be possible by brute force to determine the secret key 160.
- In particular, an attacker E can determine the common key by discovering a solution to the equation

$$3^a \bmod 353 = 40 \text{ or the equation } 3^b \bmod 353 = 248$$

- The brute-force approach is to calculate powers of 3 modulo 353, stopping when the result equals either 40 or 248.
- The desired answer is reached with the exponent value of 97, which provides

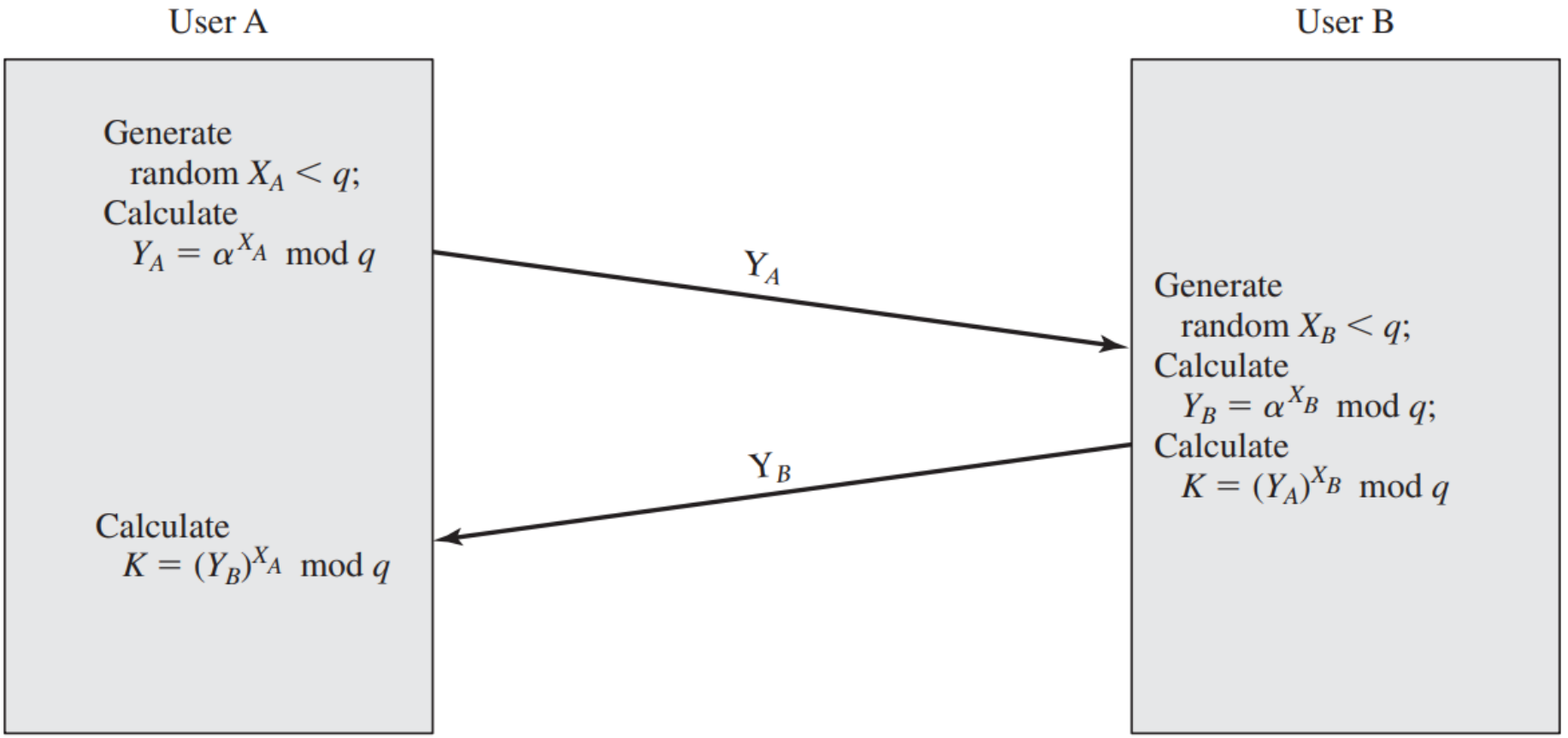
$$3^{97} \bmod 353 = 40$$

- With larger numbers, the problem becomes impractical.

Key Exchange Protocols

- Suppose that user A wishes to set up a connection with user B and use a secret key to encrypt messages on that connection.
- User A can generate a one-time private key X_A , calculate Y_A, and send that to user B. User B responds by generating a private value X_B, calculating Y_B, and sending Y_B to user A.
- Both users can now calculate the key. The necessary public values q and α would need to be known ahead of time. Alternatively, user A could pick values for q and α and include those in the first message.

Key Exchange Protocols



Man-in-the-Middle Attack

- The protocol represent in previous slide is insecure against a man-in-the-middle attack.
- Suppose Alice and Bob wish to exchange keys, and Darth is the adversary.

The attack proceeds as follows.

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_B} \bmod q$.
5. Bob transmits Y_B to Alice.
6. Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)^{X_{D1}} \bmod q$.
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$.

- At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key and Alice and Darth share secret key .

All future communication between Bob and Alice is compromised in the following way.

1. Alice sends an encrypted message .

2. Darth intercepts the encrypted message and decrypts it to recover .

3. Darth sends Bob , where is any message.

- In the first case, Darth simply wants to eavesdrop on the communication without altering it.
- In the second case, Darth wants to modify the message going to Bob.
- The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants.
- This vulnerability can be overcome with the use of digital signatures and public-key certificates;

Cryptographic Hash Function

- A hash function H accepts a variable-length block of data as input and produces a fixed-size hash value

  h = H(M)

- A "good" hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random.
- In general terms, the principal object of a hash function is data integrity.
- A change to any bit or bits in results, with high probability, in a change to the hash code.

- The kind of hash function needed for security applications is referred to as a cryptographic hash function
- A cryptographic hash function is an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either
  (a) a data object that maps to a pre-specified hash result (the one-way property) or
  (b) two data objects that map to the same hash result (the collision-free property).

- Because of these characteristics, hash functions are often used to determine whether or not data has changed.
- The general operation of a cryptographic hash function.
- Typically, the input is padded out to an integer multiple of some fixed length (e.g., 1024 bits), and the padding includes the value of the length of the original message in bits.

- The length field is a security measure to increase the difficulty for an attacker to produce an alternative message with the same hash value.

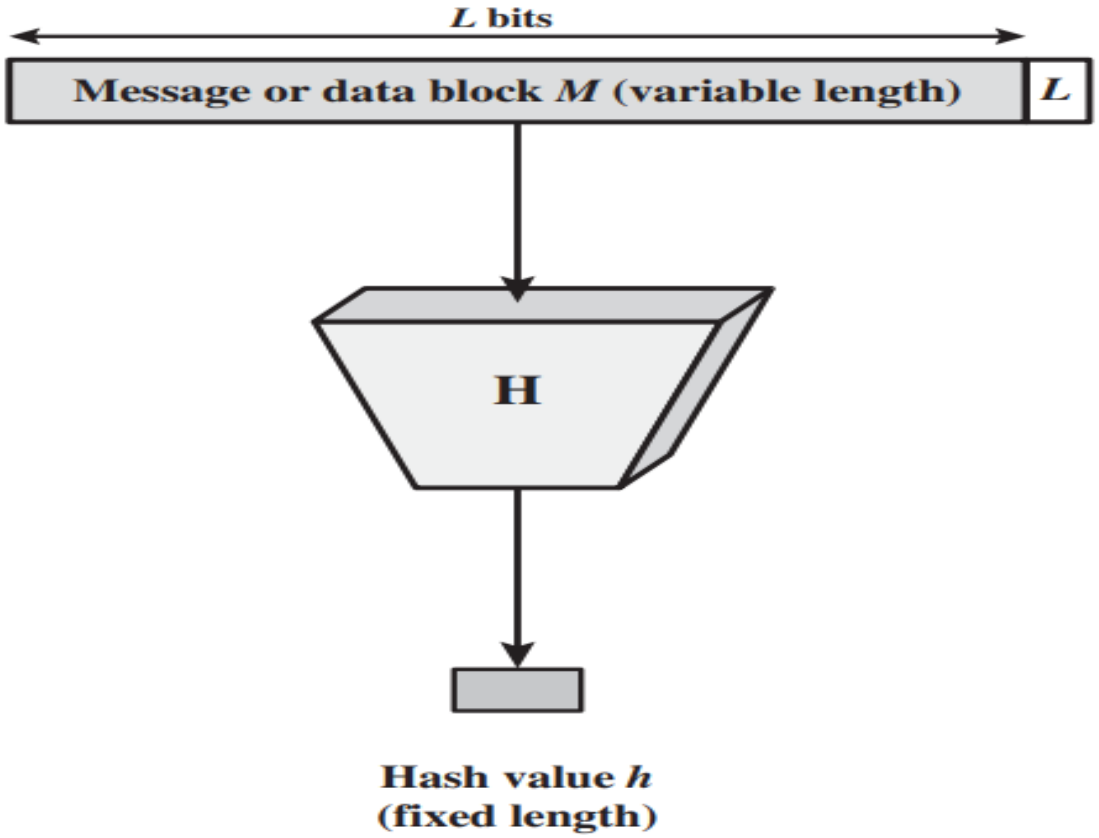


Figure 11.1 Black Diagram of Cryptographic Hash Function; $h = H(M)$

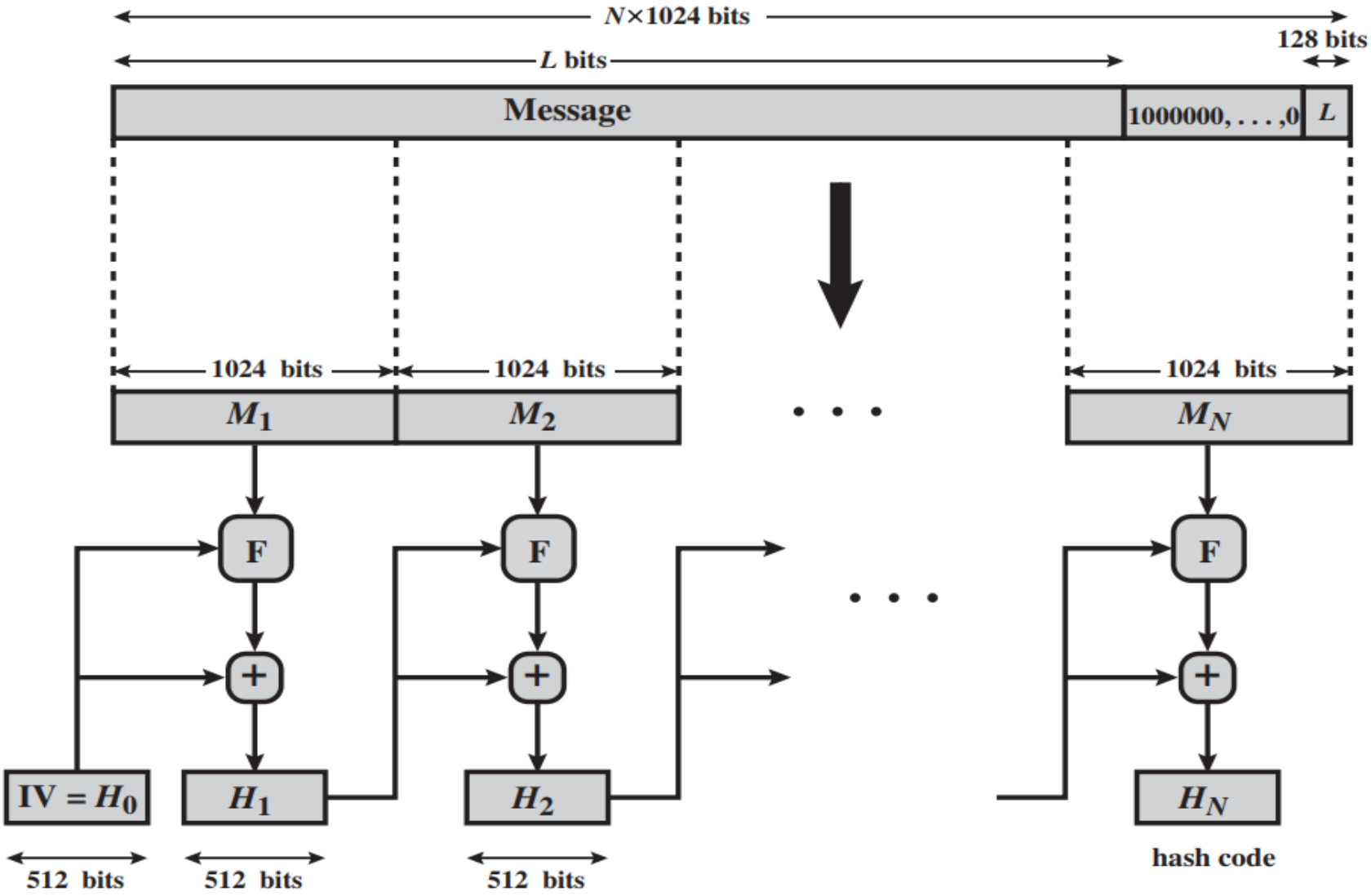
Secure Hash Algorithm (SHA)

- SHA was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993.
- The actual standards document is entitled "Secure Hash Standard." SHA is based on the hash function MD4, and its design closely models MD4.
- SHA-1 produces a hash value of 160 bits

SHA-512 Logic

- The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest.
- The input is processed in 1024-bit blocks

Message Digest Generation Using SHA-512



The processing consists of the following steps.

Step 1 Append padding bits.

- The message is padded so that its length is congruent to 896 modulo 1024 $[\text{length} \equiv 896 (\text{mod } 1024)]$
- Padding is always added, even if the message is already of the desired length.
- Thus, the number of padding bits is in the range of 1 to 1024.
- The padding consists of a single 1 bit followed by the necessary number of 0 bits.

Step 2 Append length.

- A block of 128 bits is appended to the message.
- This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the length of the original message (before the padding).
- The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length.
- The expanded message is represented as the sequence of 1024-bit blocks M_1, M_2,….M_N, so that the total length of the expanded message is N X 1024bits .

Step 3 Initialize hash buffer.

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

```
a = 6A09E667F3BCC908   e = 510E527FADE682D1

b = BB67AE8584CAA73B   f = 9B05688C2B3E6C1F

c = 3C6EF372FE94F82B   g = 1F83D9ABFB41BD6B

d = A54FF53A5F1D36F1   h = 5BE0CD19137E2179
```

- These values are stored in big-endian format, which is the most significant byte of a word in the low-address (leftmost) byte position.
- These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

- **Step 4 Process message in 1024-bit (128-word) blocks.** The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F

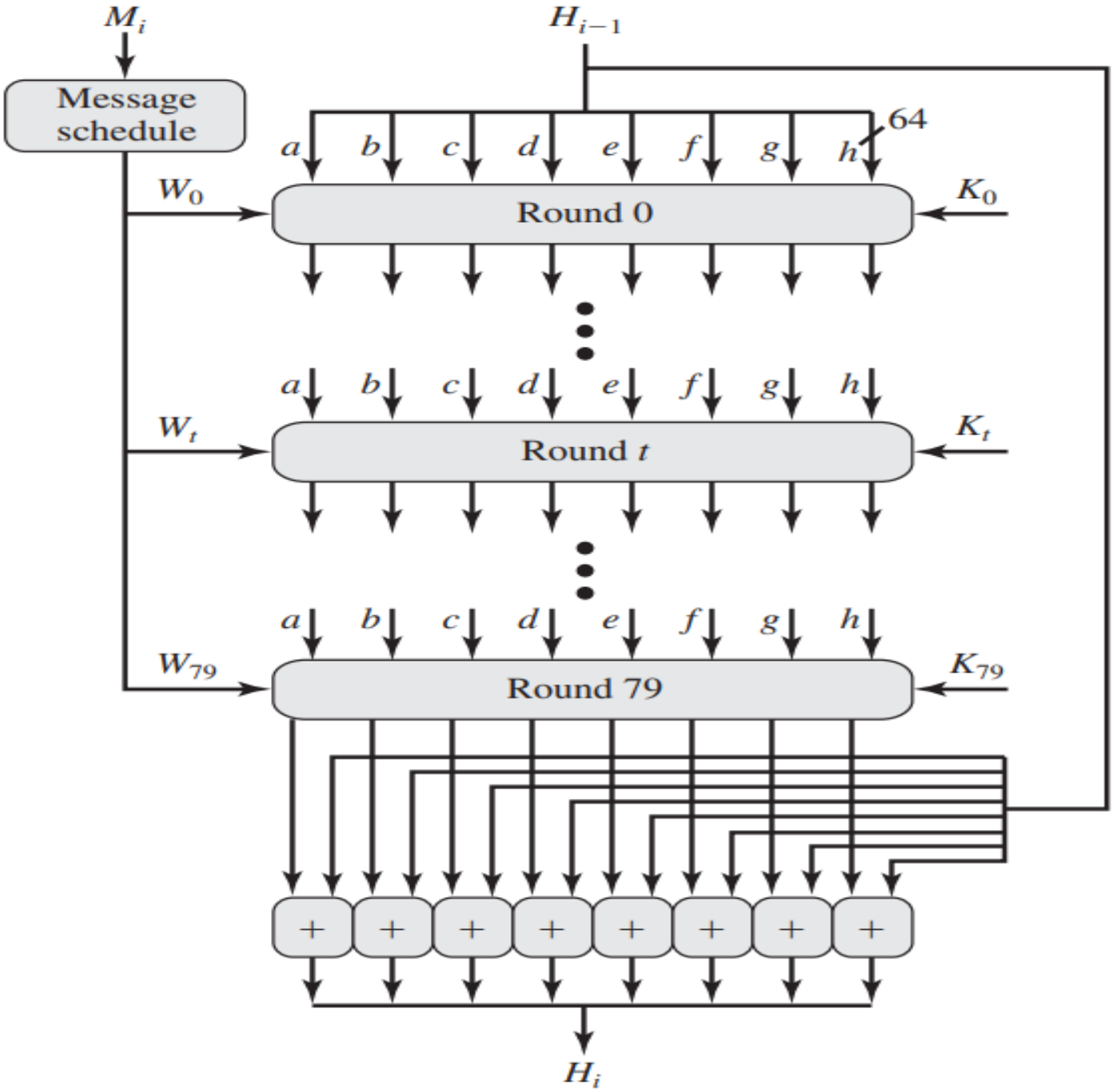


Figure 11.9 SHA-512 Processing of a Single 1024-Bit Block

- Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer.
- At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1}.
- Each round makes use of a 64-bit value W_t, derived from the current 1024-bit block being processed(M_i).
- These values are derived using a message schedule described subsequently.
- Each round also makes use of an additive constant K_t, where $0 <= t <= 79$ indicates one of the 80 rounds.

- These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers.
- The constants provide a “randomized” set of 64-bit patterns, which should eliminate any regularities in the input data

Step 5 Output.

- After all 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.

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

$$H_0 = \text{IV}$$

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

$$MD = H_N$$

where

IV	=	initial value of the abcdefgh buffer, defined in step 3
abcdefgh_i	=	the output of the last round of processing of the ith message block
N	=	the number of blocks in the message (including padding and length fields)
SUM_{64}	=	addition modulo 2^{64} performed separately on each word of the pair of inputs
MD	=	final message digest value

Message Authentication Code

- Message authentication is a mechanism or service used to verify the integrity of a message.
- Symmetric encryption provides authentication among those who share the secret key.
- A **message authentication code** (MAC) is an algorithm that requires the use of a secret key. A MAC takes a variable-length message and a secret key as input and produces an authentication code. A recipient in possession of the secret key can generate an authentication code to verify the integrity of the message.

Message Authentication Requirement

In the context of communications across a network, the following attacks can be identified.

1. **Disclosure**: Release of message contents to any person or process not possessing the appropriate cryptographic key.

2. **Traffic analysis**: Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.

3. **Masquerade**: Insertion of messages into the network from a fraudulent source.

- ✓ This includes the creation of messages by an opponent that are purported to come from an authorized entity.
- ✓ Also included are fraudulent acknowledgments of message receipt or nonreceipt by someone other than the message recipient.

4. Content modification: Changes to the contents of a message, including insertion, deletion, transposition, and modification.

5. Sequence modification: Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.

6. Timing modification: Delay or replay of messages.

- ✓ In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed.
- ✓ In a connectionless application, an individual message (e.g., datagram) could be delayed or replayed.

7. Source repudiation: Denial of transmission of message by source.

8. Destination repudiation: Denial of receipt of message by destination.

Message Authentication Functions

- **Hash function**: A function that maps a message of any length into a fixed length hash value, which serves as the authenticator
- **Message encryption**: The ciphertext of the entire message serves as its authenticator
- **Message authentication code (MAC):** A function of the message and a secret key that produces a fixed-length value that serves as the authenticator

Message Authentication Code

- An authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a **cryptographic checksum** or MAC, that is appended to the message.
- This technique assumes that two communicating parties, say A and B, share a common secret key.
- When A has a message to send to B, it calculates the MAC as a function of the message and the key:

$$\mathbf{MAC = MAC(K, M)}$$

- Where

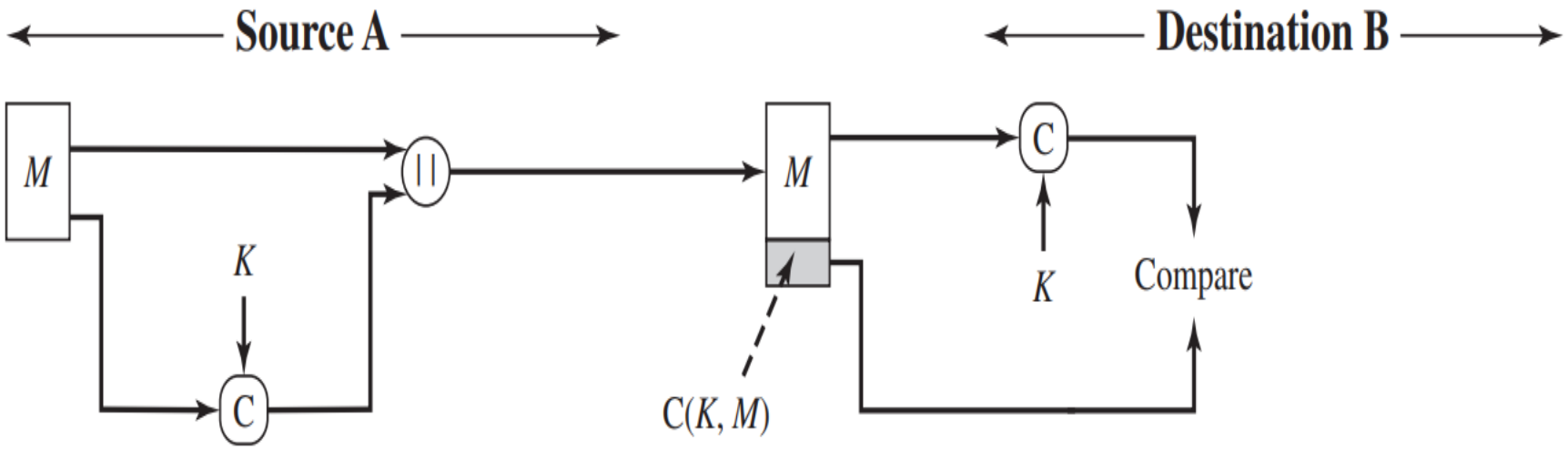
  M = Input message

  C = MAC function

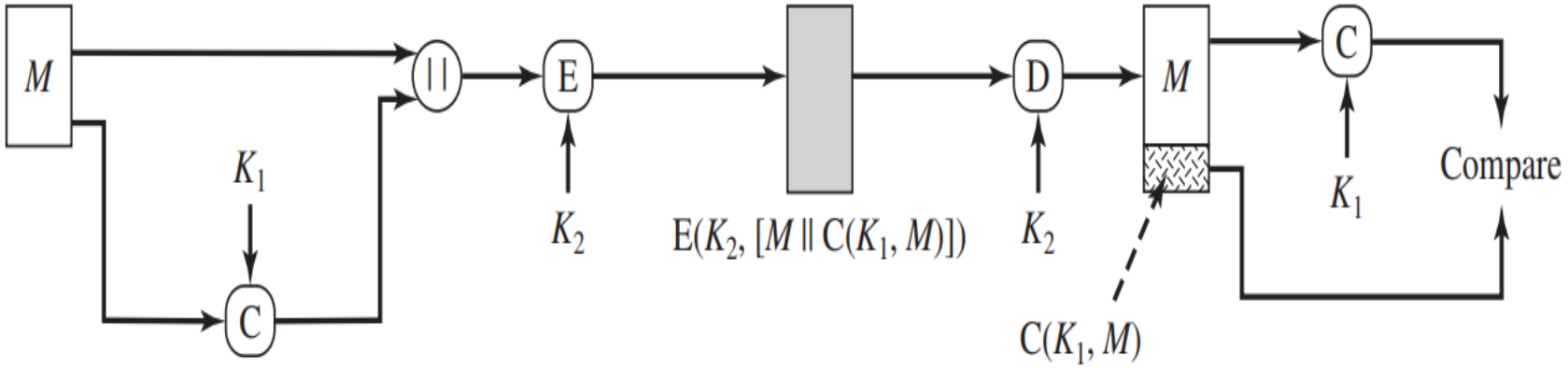
  K = Shared secret key

  MAC = Message Authentication Code

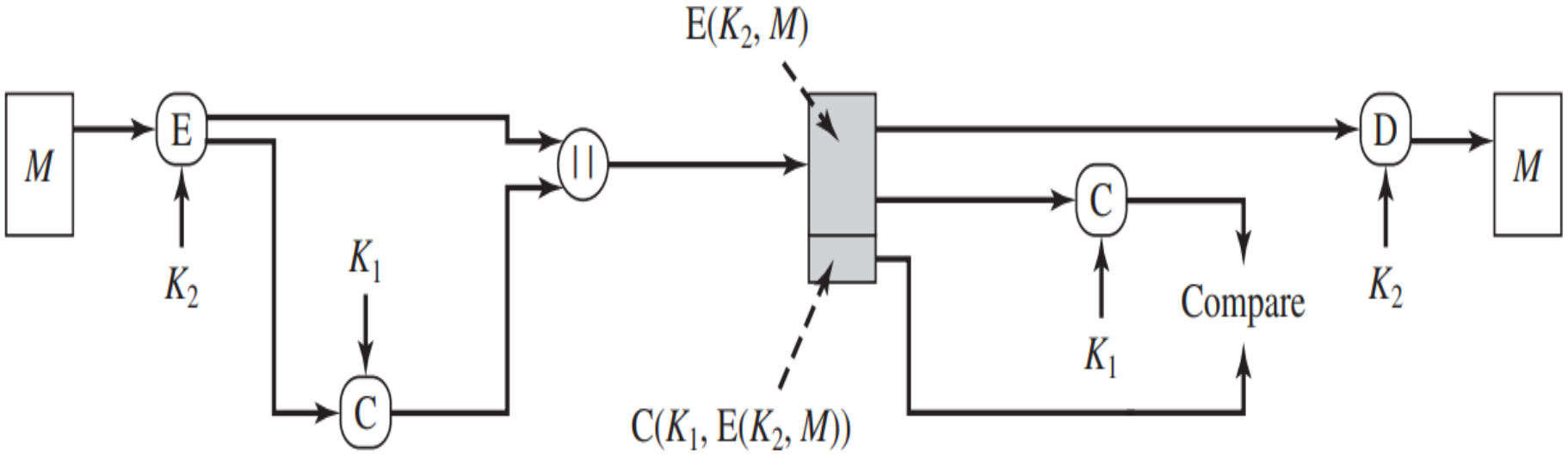
- The message plus MAC are transmitted to the intended recipient.
- The recipient performs the same calculate generate a new MAC.
- The received MAC is compared to the calculated MAC.
- To assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC



(a) Message authentication



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



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

Figure 12.4 Basic Uses of Message Authentication code (MAC)

- The receiver is assured that the message has not been altered.
- If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC.
- Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.

- The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
- If the message includes a sequence number (such as is used with HDLC, X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

Requirement for MAC

- The MAC function should satisfy the following requirements

1. If an opponent observes M and $\text{MAC}(K, M)$, it should be computationally infeasible for the opponent to construct a message M' such that

$$\text{MAC}(K, M') = \text{MAC}(K, M)$$

2. $\text{MAC}(K, M)$ should be uniformly distributed in the sense that for randomly chosen messages, M and M', the probability that $\text{MAC}(K, M) = \text{MAC}(K, M')$ is 2^{-n}, where n is the number of bits in the tag.
3. Let M' be equal to some known transformation on M. That is, $M' = f(M)$. For example, f may involve inverting one or more specific bits. In that case,

$$\Pr[\text{MAC}(K, M) = \text{MAC}(K, M')] = 2^{-n}$$

Security of MACS

Brute-Force Attacks

- A brute-force attack on a MAC is a more difficult undertaking than a brute-force attack on a hash function because it requires known message-tag pairs

Cryptanalysis

- Cryptanalytic attacks on MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search.

MACs Based on Hash Function: HMAC

- In recent years, there has been increased interest in developing a MAC derived from a cryptographic hash function.

The motivations for this interest are

1. Cryptographic hash functions such as MD5 and SHA generally execute faster in software than symmetric block ciphers such as DES.

2. Library code for cryptographic hash functions is widely available.

HMAC Design Objectives

- To use, without modifications, available hash functions. In particular, to use hash functions that perform well in software and for which code is freely and widely available.
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

HMAC Algorithm

- **The overall operation of HMAC. Define the following terms**

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)

IV = initial value input to hash function

M = message input to HMAC (including the padding specified in the embedded hash function)

Y_i = i th block of M, $0 \leq i \leq (L - 1)$

L = number of blocks in M

b = number of bits in a block

n = length of hash code produced by embedded hash function

K = secret key; recommended length is $\geq n$; if key length is greater than b, the key is input to the hash function to produce an n-bit key

HMAC Structure

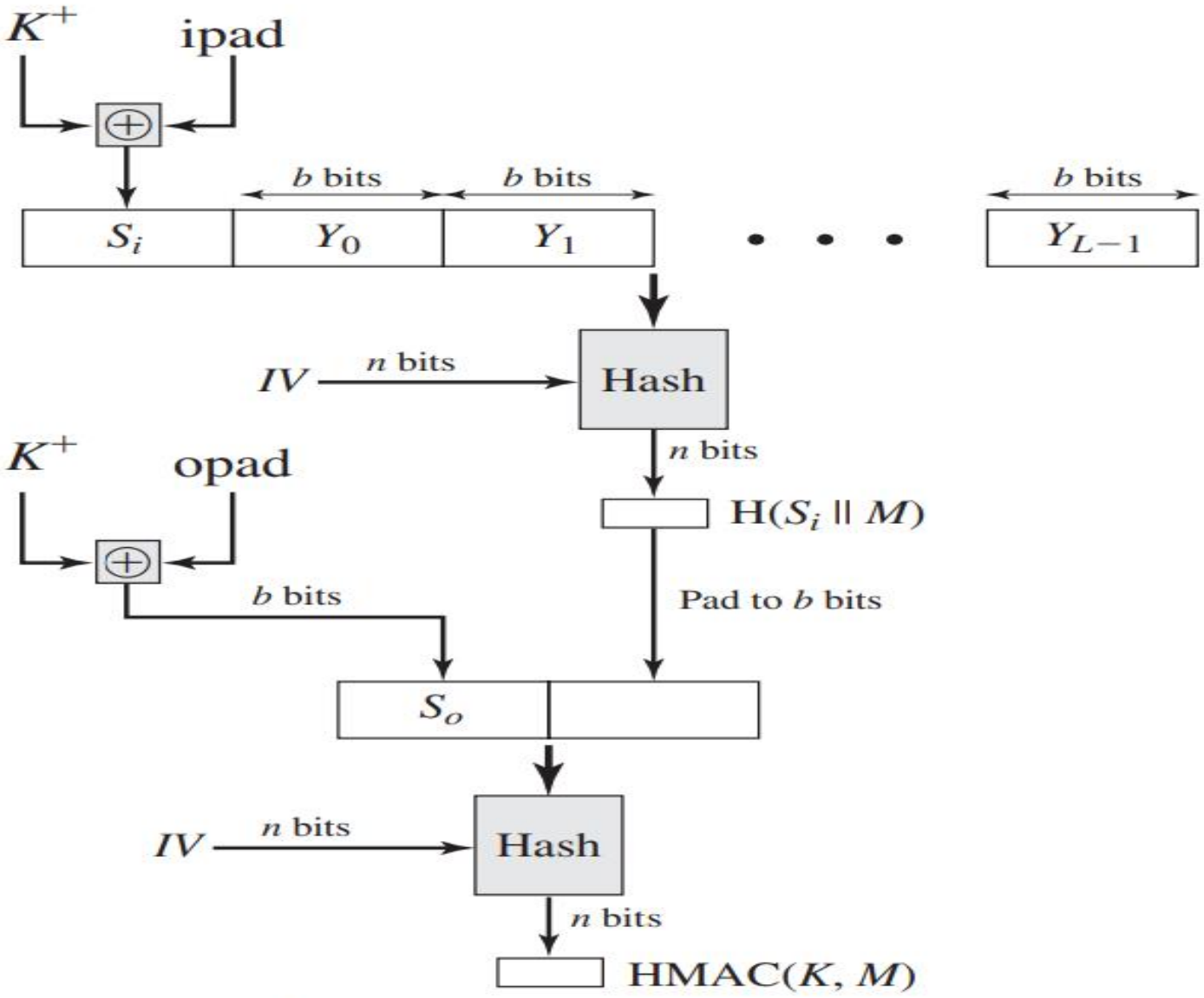


Figure 12.5 HMAC Structure

K^+ = K padded with zeros on the left so that the result is b bits in length

ipad = 00110110 (36 in hexadecimal) repeated $b/8$ times

opad = 01011100 (5C in hexadecimal) repeated $b/8$ times

Then HMAC can be expressed as

$$\text{HMAC}(K, M) = \text{H}[(K^+ \oplus \text{opad}) \| \text{H}[(K^+ \oplus \text{ipad}) \| M]]$$

We can describe the algorithm as follows.

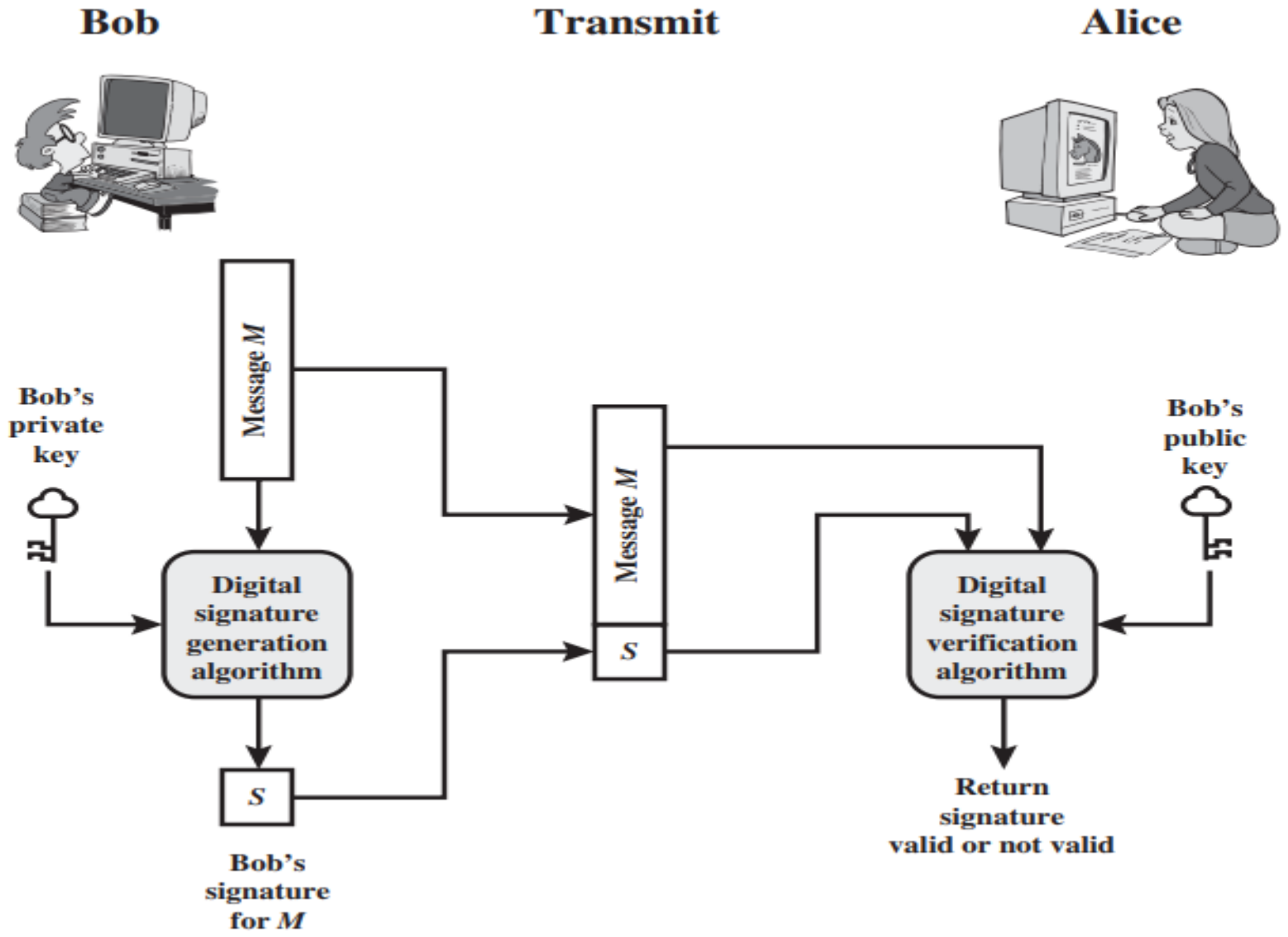
1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and $b = 512$, then K will be appended with 44 zeroes).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_o.
6. Append the hash result from step 4 to S_o.
7. Apply H to the stream generated in step 6 and output the result.

DIGITAL SIGNATURES

Properties

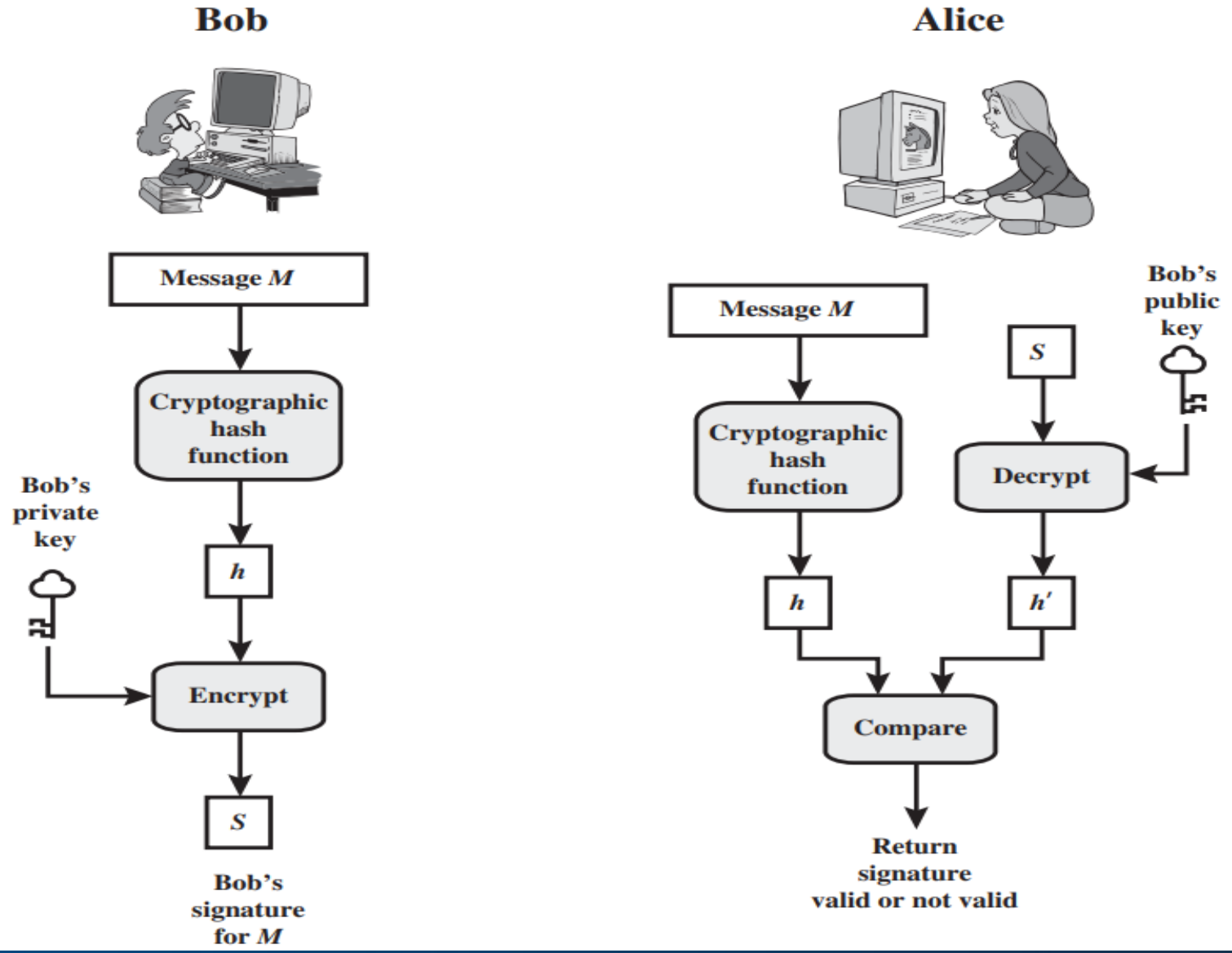
- Message authentication protects two parties who exchange messages from any third party.
- However, it does not protect the two parties against each other.
- Several forms of dispute between the two are possible

Generic Model of Digital Signature Process



- ✓ A digital signature is an authentication mechanism that enables the creator of a message to attach a code that acts as a signature.
- ✓ Typically the signature is formed by taking the hash of the message and encrypting the message with the creator's private key.
- ✓ The signature guarantees the source and integrity of the message.
- ✓ The digital signature standard (DSS) is an NIST standard that uses the secure hash algorithm (SHA).

Simplified Depiction of Essential Elements of Digital Signature Process



The digital signature must have the following properties:

- It must verify the author and the date and time of the signature.
- It must authenticate the contents at the time of the signature.
- It must be verifiable by third parties, to resolve disputes.

Attacks and Forgeries

- **Key-only attack:** C only knows A's public key.
- **Known message attack:** C is given access to a set of messages and their signatures.
- **Generic chosen message attack:** C chooses a list of messages before attempting to breaks A's signature scheme, independent of A's public key. C then obtains from A valid signatures for the chosen messages. The attack is generic, because it does not depend on A's public key; the same attack is used against everyone.
- **Directed chosen message attack:** Similar to the generic attack, except that the list of messages to be signed is chosen after C knows A's public key but before any signatures are seen.
- **Adaptive chosen message attack:** C is allowed to use A as an "oracle." This means the A may request signatures of messages that depend on previously obtained message–signature pairs.

Total break: C determines A's private key.

Universal forgery: C finds an efficient signing algorithm that provides an equivalent way of constructing signatures on arbitrary messages.

Selective forgery: C forges a signature for a particular message chosen by C.

Existential forgery: C forges a signature for at least one message. C has no control over the message. Consequently, this forgery may only be a minor nuisance to A.

Digital Signature Requirements

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information unique to the sender to prevent both forgery and denial.
- It must be relatively easy to produce the digital signature.
- It must be relatively easy to recognize and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to retain a copy of the digital signature in storage

Direct Digital Signature

- The term direct digital signature refers to a digital signature scheme that involves only the communicating parties (source, destination).
- It is assumed that the destination knows the public key of the source


THANK YOU