Cryptographic Tools

- Cryptographic algorithms are an important element in many computer security services and applications
  - We overview the various types of algorithms
  - We discuss their applicability
  - We introduce the most important standardized algorithms in common use

Learning Objectives

- Explain the basic operation of **symmetric block encryption algorithms**
- Compare block encryption and **stream** encryption
- Discuss the use of secure **hash functions** for message authentication and other applications
- Explain the basic operation of **asymmetric block encryption algorithms**
- Present an overview of the **digital signature** mechanism and the concept of digital envelopes
- Explain the significance of **random and pseudorandom numbers** in cryptography

Index

- Confidentiality with Symmetric Encryption
- Message Authentication and Hash Functions
- Public-Key Encryption
- Digital Signatures and Key Management
- Random and Pseudorandom Numbers
- Practical Application: Encryption of Stored Data

Index

- Confidentiality with Symmetric Encryption
- Message Authentication and Hash Functions
- Public-Key Encryption
- Digital Signatures and Key Management
- Random and Pseudorandom Numbers
- Practical Application: Encryption of Stored Data

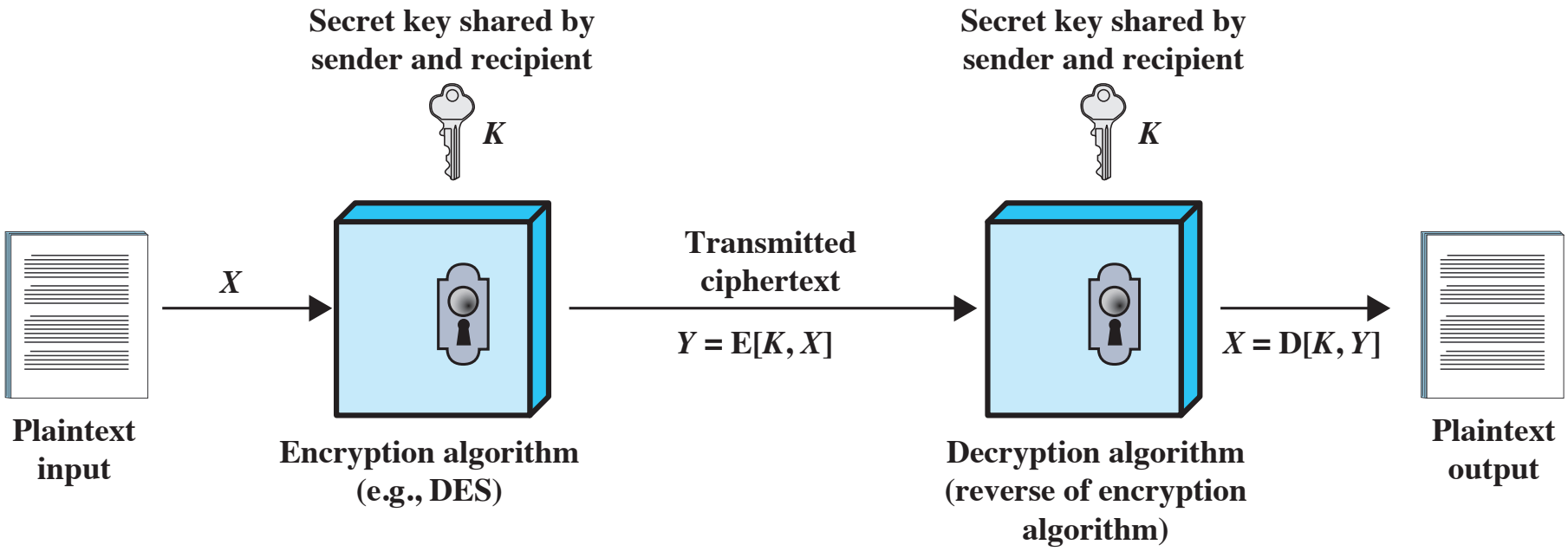
Symmetric Encryption

- Also referred to as *conventional encryption* or *single-key encryption*
- Universal technique for providing confidentiality for transmitted or stored data
- The only type of encryption in use prior to the intro of public-key encryption (late 1970s)
- It remains the more widely used of the two types of encryption

Five ingredients

- Plaintext: the original message or data that is fed into the algorithm as input
- Encryption algorithm: performs various substitutions and transformations on the plaintext
- Secret key: Another input to the encryption algorithm
  - The exact substitutions and transformations performed by the algorithm depend on the key
- Ciphertext: the scrambled message produced as output
  - It depends on the plaintext and the secret key
  - For a given message, two different keys will produce two different ciphertexts
- Decryption algorithm: essentially, the encryption algorithm run in reverse
  - It takes the ciphertext and the secret key and produces the original plaintext

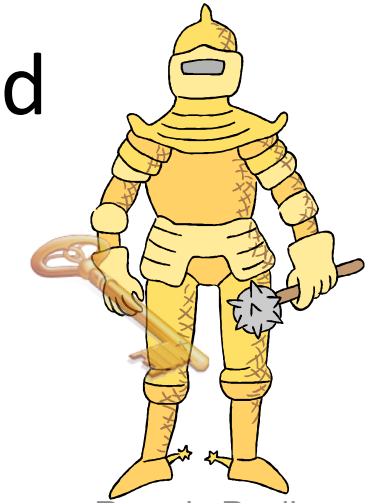
Model of Symmetric Encryption



Requirements for secure use

There are two *requirements for secure use* of a symmetric encryption scheme:

- Need a *strong* encryption algorithm
  - The opponent should be unable to decrypt the ciphertext or discover the key *even* if she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext
- Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure



Approaches to Attacking

There are two general *approaches to attacking* a symmetric encryption scheme

Cryptanalytic Attacks

- Rely on the *characteristics of the algorithm*, plus
  - some knowledge of the general characteristics of the plaintext, or
  - some sample plaintext-ciphertext pairs
- Attempt to deduce
  - a specific plaintext, or
  - the key being used, thus all future and past messages encrypted with that key are compromised

Brute-Force Attacks

- 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
  - Some degree of *knowledge about the expected plaintext* is needed, as well as some means of *automatically* distinguishing plaintext from garble

Symmetric 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
- The algorithm processes longer plaintext amounts as a series of fixed-size blocks
- The most important symmetric block encryption algorithms are
  - Data Encryption Standard (DES)
  - Triple DES
  - Advanced Encryption Standard (AES)

Comparison of Three Popular Symmetric Encryption Algorithms

	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

DES = Data Encryption Standard
AES = Advanced Encryption Standard

Data Encryption Standard (DES)

- Until recently, the *most widely used encryption scheme*
- Adopted in 1977 by the National Bureau of Standards, now the *National Institute of Standards and Technology* (NIST), as *Federal Information Processing Standard* 46 (FIPS PUB 46)
- The algorithm itself is referred to as the *Data Encryption Algorithm* (DEA)
  - Takes a plaintext block of 64 bits and a key of 56 bits, to produce a ciphertext block of 64 bits
  - The key is actually 64 bits, but every 8th bit is a parity check; so, only 56 or the 64 bits are meaningful
  - The algorithm originally had a 128-bit key, but the size of the key space was reduced by the NSA (for some reason)

Concerns about Strength of DES

- Weaknesses in the algorithm (**cryptanalysis**)
  - DEA is the *most-studied encryption algorithm* in existence
  - Despite numerous attempts, no one has so far reported a fatal weakness in the algorithm
- Key length (**brute force attacks**)
  - With a key length of 56 bits, there are 2^{56} possible keys, which is approximately $7.2 * 10^{16}$ keys
  - Electronic Frontier Foundation (EFF) announced in July 1998 that it had broken a DES encryption

If the only form of attack that could be made on an encryption algorithm is brute force, then the way to counter such attacks is obvious: *use longer keys*

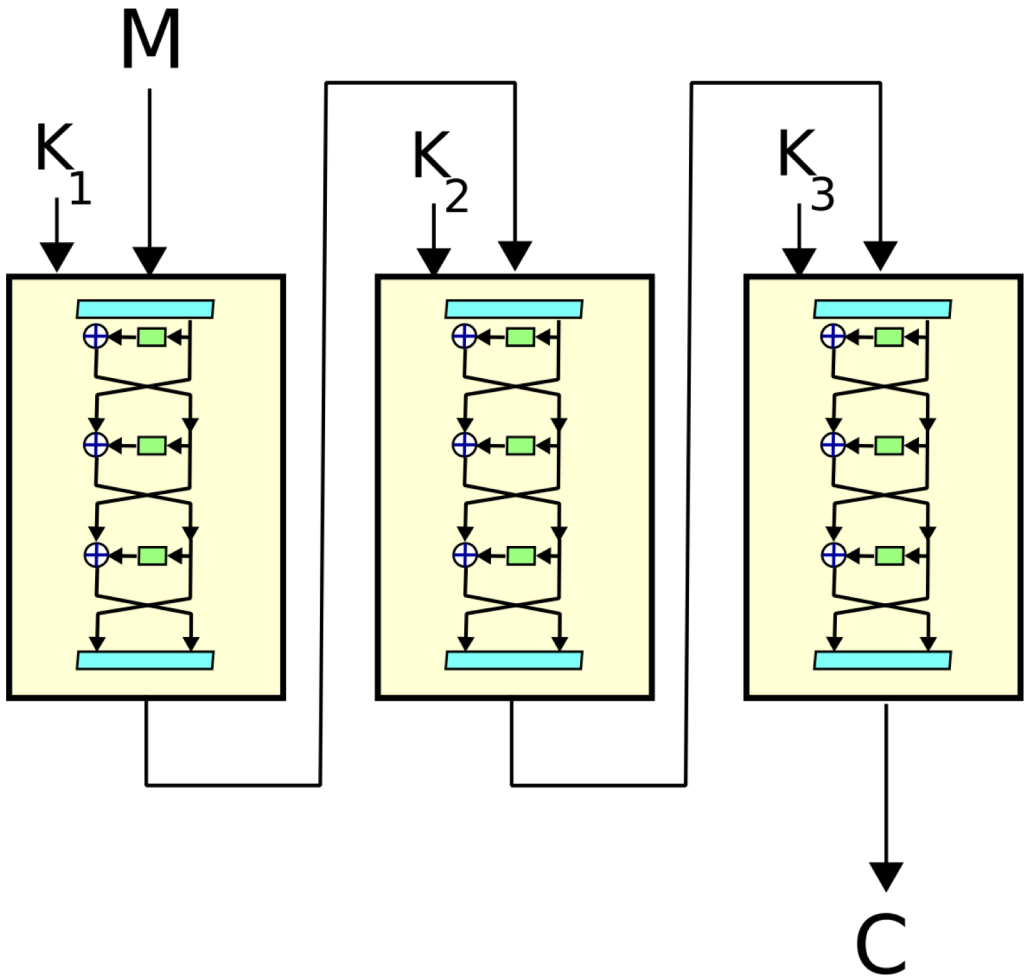
Average Time Required for Exhaustive Key Search

Key size (bits)	Cipher	Number of Alternative Keys	Time Required at 10^9 decryptions/s	Time Required at 10^{13} decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	2^{55} ns = 1.125 years	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	2^{127} ns = 5.3×10^{21} years	5.3×10^{17} years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	2^{167} ns = 5.8×10^{33} years	5.8×10^{29} years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	2^{191} ns = 9.8×10^{40} years	9.8×10^{36} years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	2^{255} ns = 1.8×10^{60} years	1.8×10^{56} years

- A single PC can break DES in about a year and today's supercomputers should be able to find a key in about an hour
- Key sizes of 128 bits or greater are effectively unbreakable using simply a brute-force approach

Triple DES (3DES)

Repeats basic DES algorithm three times using one, two or three keys, for a key size of 168 bits



With three different keys the key length is of 168 bits

Triple DES (3DES)

- First standardized for use in financial applications in ANSI standard X9.17 in 1985
- **Advantages**:
  - 168-bit key length overcomes the vulnerability to brute-force attack of DES (actually, the effective security provided by 3DES corresponds to a 112-bit key length)
  - The underlying encryption algorithm is the same as in DES which has been longly scrutinized and no effective cryptanalytic attack based on the algorithm has been found
- **Drawbacks**:
  - Does not produce efficient software code
  - Uses a 64-bit block size (for efficiency and security, a larger block size is desirable)

Advanced Encryption Standard (AES)

Needed a replacement for 3DES

- 3DES was not reasonable for long term use

NIST called for proposals for a new AES in 1997

- Should have a security strength equal to or better
- Significantly improved efficiency
- Symmetric block cipher
- 128 bit data and 128/192/256 bit keys

Selected Rijndael in November 2001

- In a first round, 15 algorithms were accepted
- A second round narrowed the field to 5 algorithms
- Published as FIPS 197

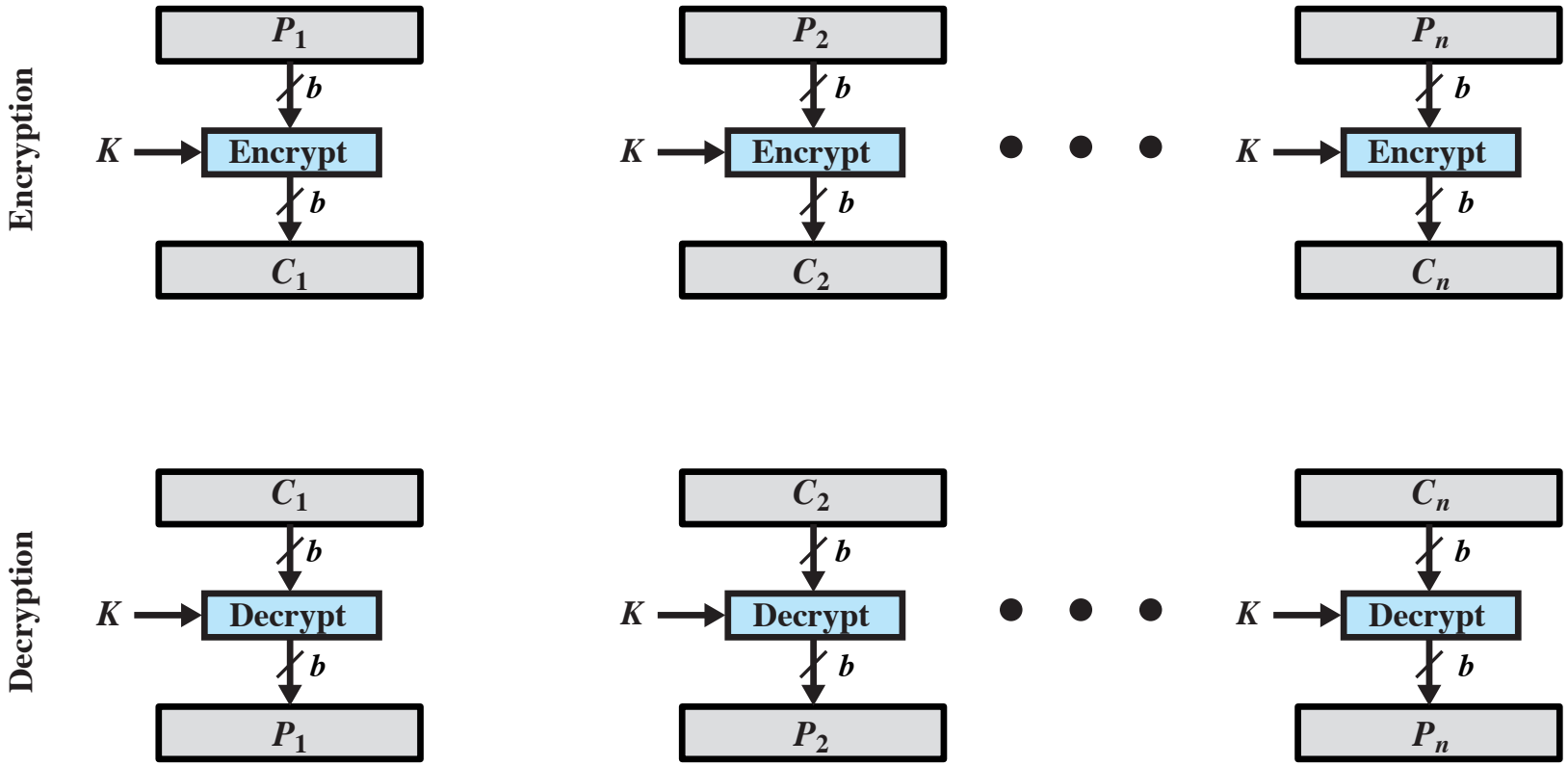
Different types of Symmetric Encryption

- Block Ciphers (es. DES, 3DES, AES)
- Stream Ciphers (es. RC4, A5/1)

Block Ciphers

- Typically symmetric encryption is applied to a unit of data larger than a single 64-bit or 128-bit block
  - Es. e-mail messages, network packets, database records, and other plaintext sources must be broken up into a series of fixed-length blocks for encryption by a symmetric block cipher
- Block ciphers basically operate as follows
  - A plaintext of length *n*b* is divided into *n b*-bit blocks (P_1, P_2,….,P_n) (the plaintext is possibly padded so that it is a multiple of the block size)
  - Each block is separately encrypted
  - A sequence of *n b*-bit blocks of ciphertext (C_1, C_2,….,C_n) is produced
- **Electronic CodeBook** (ECB) mode is the simplest approach to multiple-block encryption
  - Each block is encrypted using same algorithm and the same key
  - Cryptanalysts can exploit regularities in the plaintext
  - For lengthy messages, the ECB mode may not be secure
- *Modes of operation*
  - Alternative techniques developed to *increase* the security of symmetric block encryption for large sequences
  - Overcome the weaknesses of ECB

Block cipher encryption (Electronic Codebook Mode)

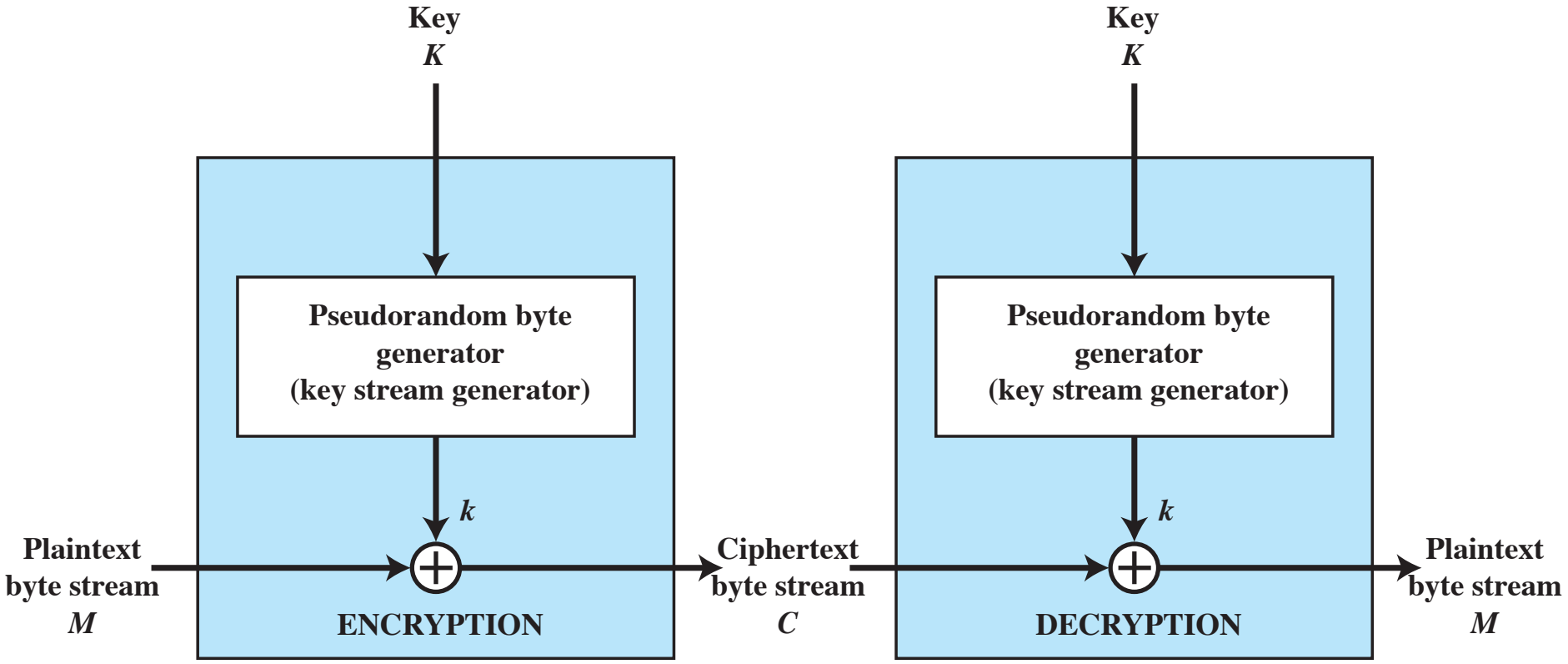


- A plaintext of length *n*b* is divided into *n b*-bit blocks (P_1, P_2,….,P_n)
- Each block is encrypted using the same algorithm and the same encryption key
- A sequence of *n b*-bit blocks of ciphertext (C_1, C_2,….,C_n) is produced

Stream cipher encryption

- A stream cipher *processes the input elements continuously, producing output one element at a time*, as it goes along
  - A typical stream cipher encrypts plaintext one byte at a time, although a stream cipher may be designed to operate on one bit at a time or on units longer than a byte at a time
- Stream ciphers basically operate as follows
  - A **key is input to a pseudorandom bit generator** that produces a stream of 8-bit numbers that are apparently random
    - A pseudorandom stream is one that is unpredictable without knowledge of the input key and which has an apparently random character
  - The output of the generator, called a keystream, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation

Stream cipher encryption



- A key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random
- The output of the generator, called a keystream, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation

Block & Stream Ciphers

- The advantage of a block cipher is that you can reuse keys
- The primary advantage of a stream cipher is that stream ciphers are almost always faster and use far less code than do block ciphers
- With a properly designed *pseudorandom number generator*, a stream cipher can be as secure as a block cipher of comparable key length

Block & Stream Ciphers

- Although block ciphers are far more common, there are certain applications in which a stream cipher is more appropriate
  - Block ciphers may be more appropriate for applications that deal with blocks of data, e.g. file transfer, e-mail, database
  - Stream ciphers might be a better alternative for applications that require encryption/decryption of a stream of data, e.g. over a communication channel or a browser/Web link
- However, *either type of cipher can be used in virtually any application*

Index

- Confidentiality with Symmetric Encryption
- Message Authentication and Hash Functions
- Public-Key Encryption
- Digital Signatures and Key Management
- Random and Pseudorandom Numbers
- Practical Application: Encryption of Stored Data

Message (or Data) Authentication

- Protects against active attacks (falsification of data and transactions)
  - *Encryption* protects against passive attacks (eavesdropping)
- A procedure that allows communicating parties to verify that received or stored messages (files, documents, or other collections of data) are authentic, i.e.
  - are **genuine**, i.e. their content has not been altered
  - came from their **alleged source**
- One may also wish to verify a message's
  - *timeliness*, i.e. it has not been artificially delayed and replayed
  - *sequence relative* to other messages flowing between two parties
- All of these concerns come under the category of data integrity

Authentication using Symm. Encryption

- When performing authentication by simply using simmetric encryption, if only sender and receiver share a key then
  - *only the genuine sender* would be able to encrypt a message successfully for the other participant
  - if the message includes an error-detection code and a sequence number, the receiver is assured that *no alterations* have been made and that *sequencing is proper*
  - if the message also includes a *timestamp*, the receiver is assured that the message has not been delayed
- But *SE alone is not a suitable tool for data authentication*
  - E.g., in the ECB (Electronic CodeBook) mode of encryption, if *an attacker reorders* the blocks of the single ciphertext, then each block will still decrypt successfully
  - But the *reordering may alter the meaning* of the overall data sequence
  - Although sequence numbers may be used at some level (e.g., each IP packet), it is typically not the case that a separate sequence number will be associated with each b-bit block of plaintext
  - Thus, *block reordering is a threat*

Authentication without Encryption

- There are several approaches to message authentication that do *not* rely on message encryption
  - In all of these approaches, an authentication tag is generated and appended to each message for transmission
  - The *message itself is not encrypted* and can be read at the destination independent of the authentication function at the destination
- Because these approaches do not encrypt the message, message confidentiality is *not* provided
- Of course, it is possible to combine authentication and confidentiality in a single algorithm by encrypting a message plus its authentication tag

Authentication without Encryption

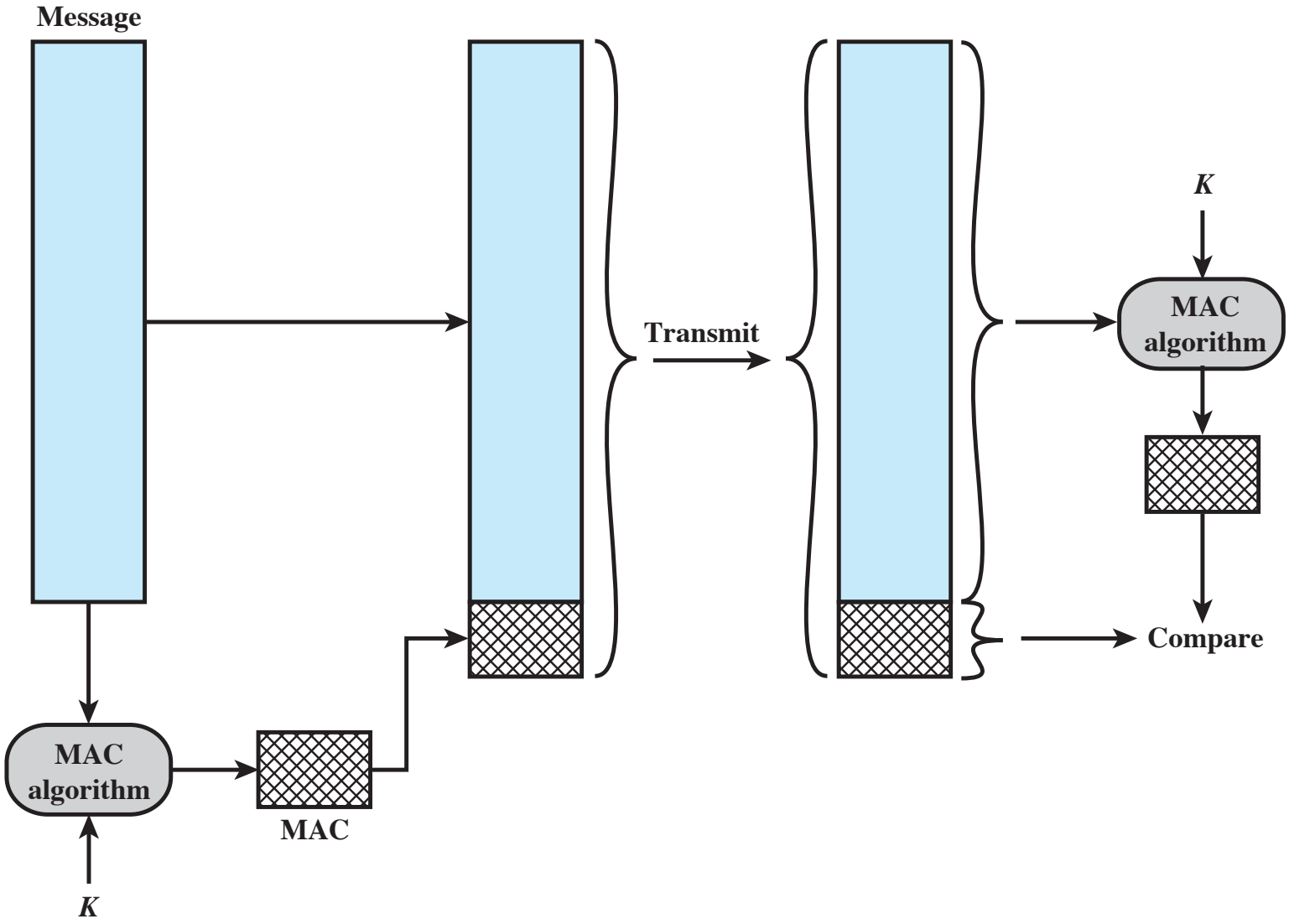
Message authentication is typically provided as a *separate function* from message encryption, as there are situations in which message authentication without confidentiality is preferable

- Applications in which *the same message is broadcast to a number of destinations*; e.g. notification to users that the network is now unavailable
  - It is cheaper and more reliable to have only one destination responsible for monitoring authenticity
  - Thus, the message must be broadcast in plaintext with an associated message authentication tag
  - The responsible system performs authentication; if a violation occurs, the other destination systems are alerted by a general alarm
- Another possible scenario is *an exchange in which one side has a heavy load* and cannot afford the time to decrypt all incoming messages
- Another attractive service is *authentication of a computer program* in plaintext
  - The computer program can be executed without having to decrypt it every time, which would be wasteful of processor resources
  - However, if a message authentication tag were attached to the program, it could be checked whenever assurance of the integrity of the program is required

Message Authentication Code (MAC)

- One authentication technique involves a small block of data, named Message Authentication Code (MAC), generated by using a secret key, that is appended to the message
- This technique assumes that two communicating parties, say A and B, share a **common secret key** K_{AB}
  - When A has a message to send to B, it calculates the MAC as a *complex function* of the message and the key: $MAC_M = F(K_{AB}, M)$
  - The message plus MAC code are transmitted to the intended recipient
  - The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC
  - The received MAC is compared to the calculated MAC
- Porperties: if the received MAC matches the calculated MAC , and if only the receiver and the sender know the identity of the secret key, then
  - the receiver is assured that the *message has not been altered*
  - the receiver is assured that the *message is from the alleged sender*
  - if the message includes a sequence number / timestamp, then the receiver can be assured of the *proper sequence / timeliness*

Message Authentication using a MAC



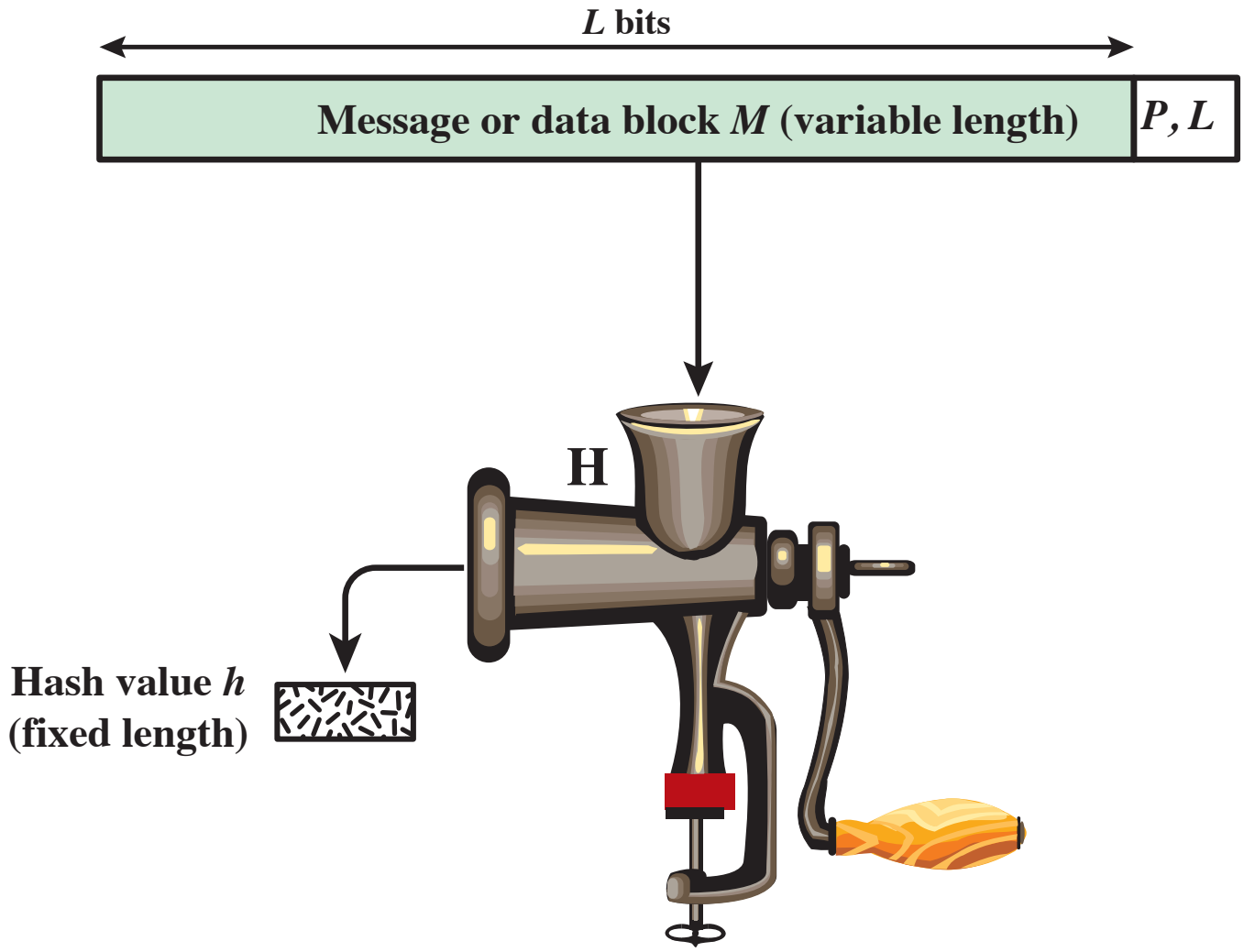
MAC: considerations

- A number of algorithms could be used to generate the MAC
  - DES or AES can be used to generate an encrypted version of the message, and some of the bits of ciphertext are used as the MAC
    - DES is applied to each 64-bit block of the message in sequence, with the input to the encryption algorithm being the XOR of the current plaintext block and the preceding ciphertext block (this is known as *Cipher Block Chaining* (CBC) mode)
    - The MAC is derived from the final block encryption
  - A 16- or 32-bit code used to be typical, but would now be much too small to provide sufficient collision resistance!
    - A function H is collision resistant if it is hard to find two inputs that are mapped to the same output
    - Every function with more inputs than outputs will necessarily have collisions
    - Collision resistance does not mean that no collisions exist; simply that they are *hard to find*
- One difference with encryption is that the authentication algorithm does not need to be reversible, as it must for decryption

Cryptographic Hash Function

- Another authentication technique, *alternative* to generation of the MAC, relies on one-way hash functions
- As with the MAC, a hash function accepts a variable-size message M as input and produces a fixed-size message digest H(M) as output
  - Typically, the message 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
- Unlike the MAC, a hash function does not take a secret key as input
- In addition to providing authentication, a message digest also provides data integrity
  - If any bits in the message are accidentally altered in transit, the message digest will be in error

Cryptographic Hash Function: h = H(M)

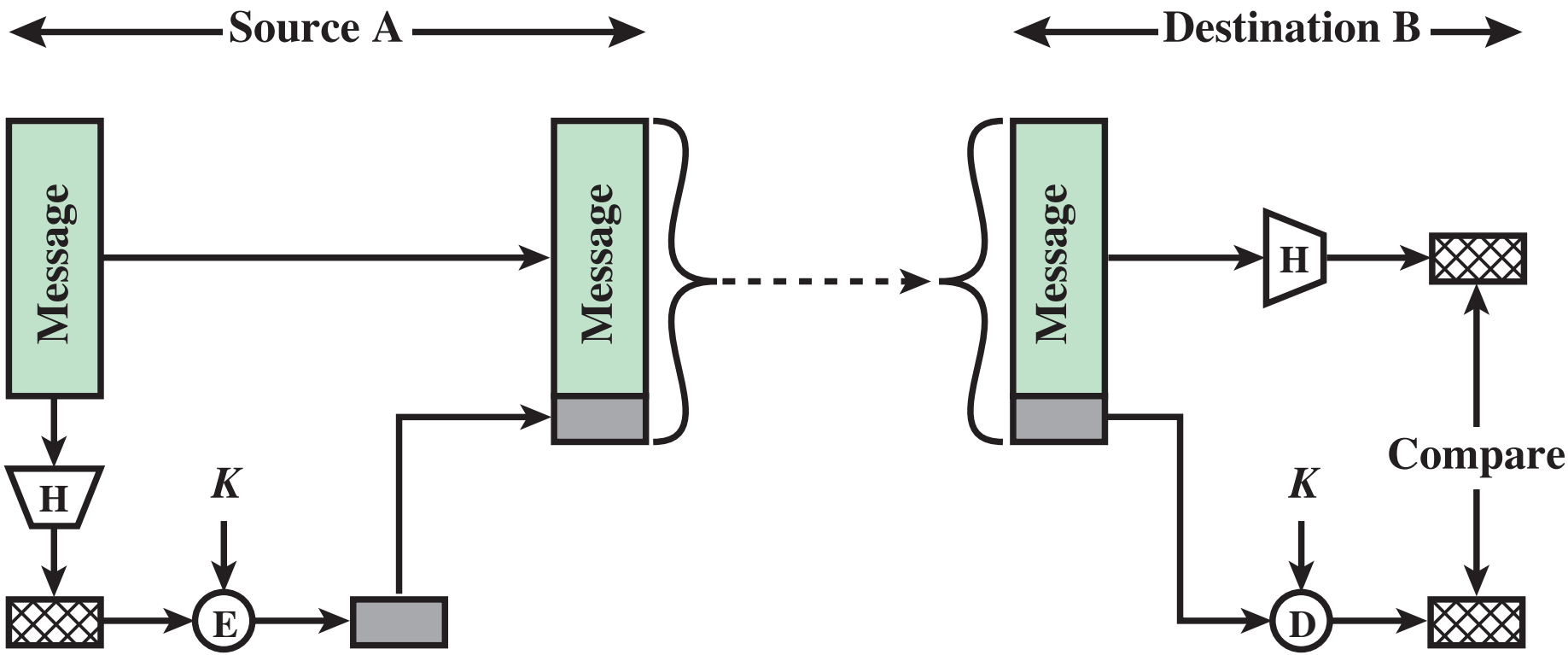


P, L = padding plus length field

Three ways of using a Hash Function for message authentication

- The message digest can be encrypted using **symmetric encryption**
  - If it is assumed that only the sender and receiver share the encryption key, then authenticity is assured
- The message digest can be encrypted using **public-key encryption**; *the public-key approach has two advantages*:
  - It does not require the distribution of keys to communicating parties
  - It also provides a digital signature (in addition to message authentication)
- The message digest uses a secret key but no encryption (**keyed hash MAC**)
  - Assumes that two communicating parties, say A and B, share a common secret key K
  - This secret key is incorporated into the process of generating a hash code
  - When A has a message to send to B, it calculates the hash function over the concatenation of the secret key and the message: $MD_M = H(K||M||K)$. It then sends $[M||MD_M]$ to B
  - Because B possesses K, it can recompute $H(K||M||K)$ and verify MD_M
  - As long as the secret key remains secret, an attacker should not be able to
    - modify an intercepted message
    - generate a false message

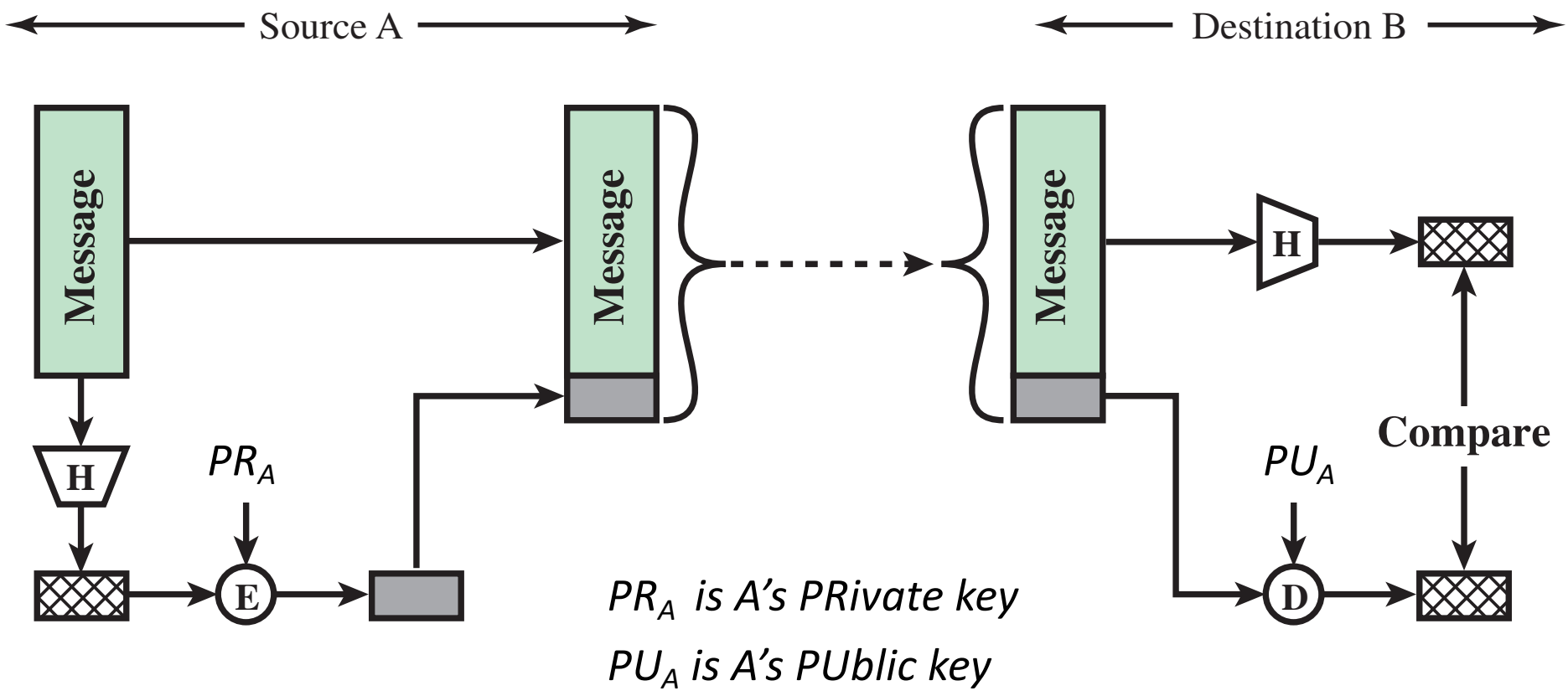
Message Authentication using a Hash Function and Symmetric Encryption



The message digest can be encrypted using symmetric encryption

- If it is assumed that only the sender and receiver share the encryption key, then authenticity is assured

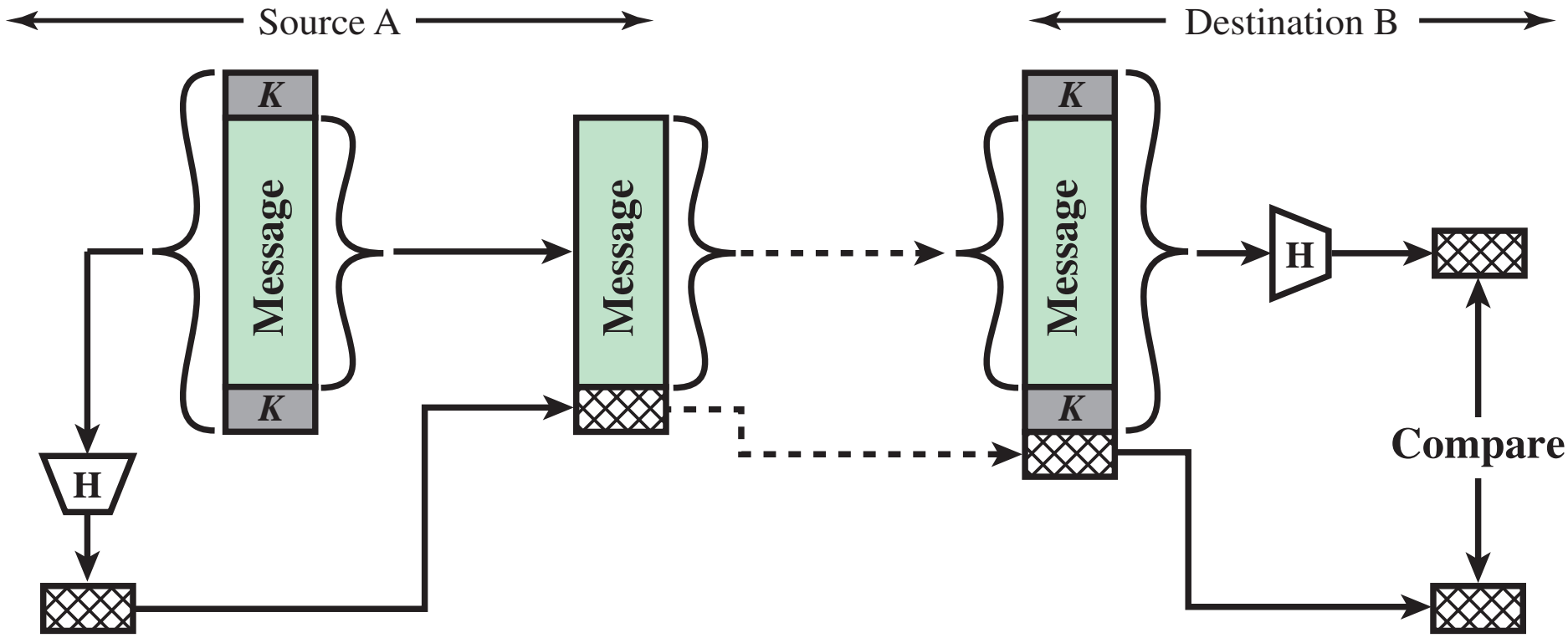
Message Authentication using a Hash Function and Public-key Encryption



Public-key encryption has two advantages

- It does not require the distribution of keys to communicating parties
- It also provides a digital signature (in addition to message authentication)

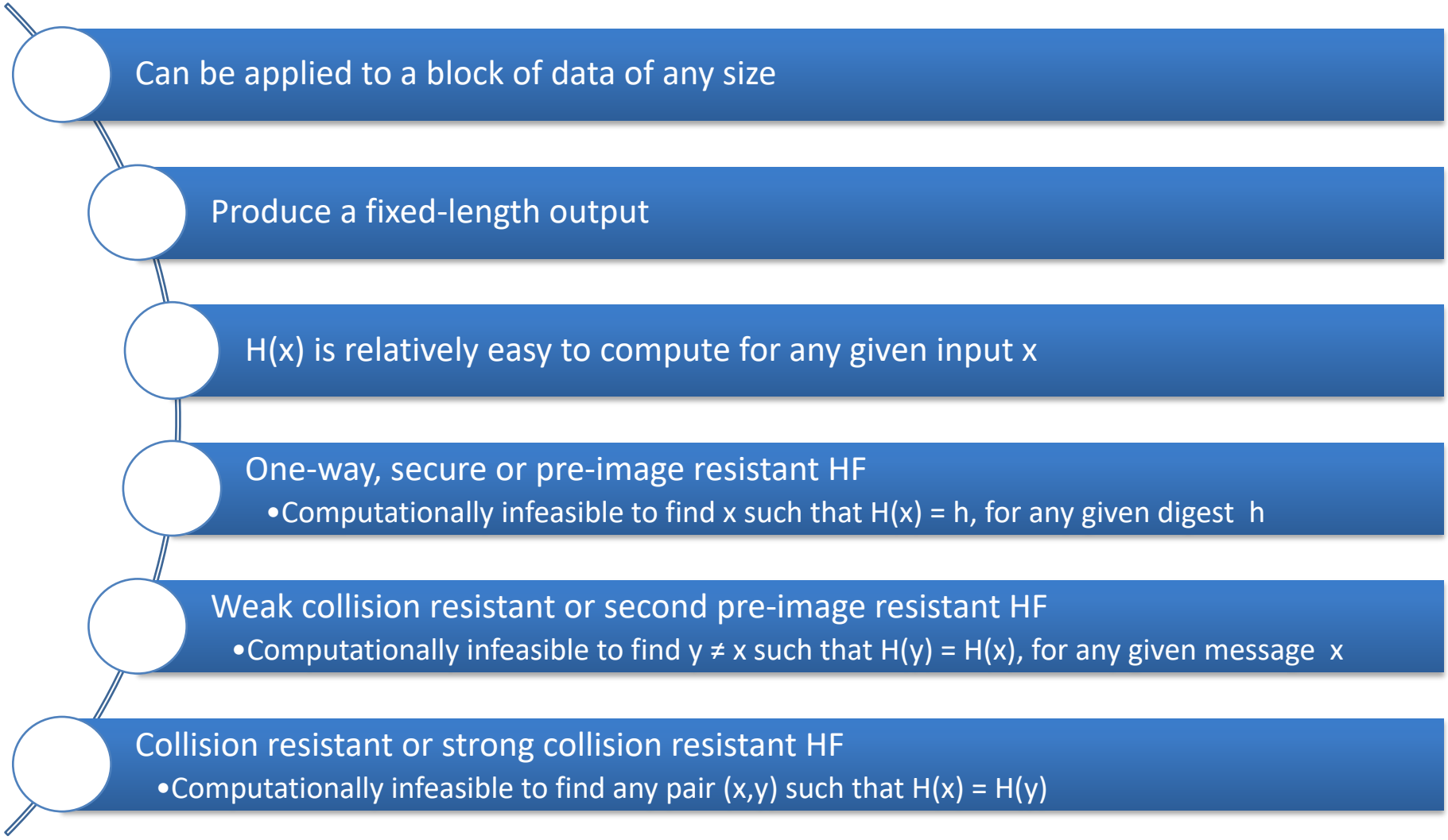
Message Authentication using a Hash Function and a Secret Value (no Encryption)



- The shared secret key K is incorporated into the process of generating a hash code
- When A has a message to send to B, it calculates the hash function over the concatenation of the secret key and the message: $MD_M = H(K||M||K)$. It then sends $[M||MD_M]$ to B
- Because B possesses K, it can recompute $H(K||M||K)$ and verify MD_M

Cryptographic HF Requirements

The purpose of a hash function is to produce a fingerprint of a file/message/data block
To be useful for message authentication, a HF H must have the following properties



1-4: one-way/secure HF 1-5: weak HF 1-6: strong HF

Cryptographic HF Requirements

- The **first three properties** are requirements for the *practical application of a hash function to message authentication*
- The **fourth property** (pre-image resistant property) guarantees that it is virtually impossible to generate a message given a hash code
  - This property is *important if the authentication technique involves the use of a secret value* S_{AB} : if the hash function is not one-way, an attacker can easily discover the secret value
    - If the attacker can observe or intercept a transmission, the attacker obtains the message *M* and the hash code $MD_M = H(S_{AB} || M)$
    - The attacker then inverts the hash function to obtain $S_{AB} || M = H^{-1}(MD_M)$
    - Because the attacker now has both *M* and $S_{AB} || M$, it is a trivial matter to recover S_{AB}
- The **fifth property** (second pre-image resistant property) guarantees that it is impossible to find an alternative message with the same hash value as a given message
  - This *prevents forgery when an encrypted hash code is used*, otherwise an attacker could:
    - observe or intercept a message plus its encrypted hash code
    - generate an unencrypted hash code from the message
    - generate an alternate message with the same hash code
- The **sixth property** (collision resistant property) protects against attacks in which one party generates a message for another party to sign
  - For example, suppose Alice agrees to sign an IOU (I owe You) for a small amount that is sent to her by Bob
  - Suppose also that Bob can find two messages with the same hash value, one of which requires Alice to pay the small amount and one that requires a large payment
  - Alice signs the first message and Bob is then able to claim that the second message is authentic

Security of Hash Function

There are two approaches to attacking a secure HF

Cryptanalysis

- Exploit logical weaknesses in the algorithm

Brute-force attack

- Strength of HF is proportional to the length *n* of the hash code
  -- 2^n: Pre-image resistant
  -- 2^n: Second pre-image resistant
  -- $2^{n/2}$: Collision resistant
- With today's technology, 160 bit length appears suspect

Most widely used HF: Secure Hash Algorithm (SHA) by NIST (1993)

SHA-1 (1995) produces hash values of 160 bits

SHA-2 (2002) produces hash values of 256, 384, and 512 bits

SHA-3 (2015) produces hash values of 256, 384, and 512 bits, but is very different from SHA-2 and SHA-1

Other applications of secure HF

Passwords

- Hash of a password is stored by an OS rather than the password itself
- Requires preimage resistance and perhaps second preim. resistance

Intrusion detection

- Store H(F) for each file on a system and secure the hash values
- An intruder would need to change F without changing H(F)
- Requires second preimage resistance

Index

- Confidentiality with Symmetric Encryption
- Message Authentication and Hash Functions
- Public-Key Encryption
- Digital Signatures and Key Management
- Random and Pseudorandom Numbers
- Practical Application: Encryption of Stored Data

Public-Key Encryption

- Of equal importance to symmetric encryption
- Finds use both in message authentication and key distribution
- The first truly *revolutionary* advance in encryption in literally thousands of years!

Public-Key Encryption Structure

Publicly proposed by Diffie and Hellman in 1976

Based on mathematical functions

Asymmetric

- Uses two separate keys: public key and private key
- Public key is made public for others to use

The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication

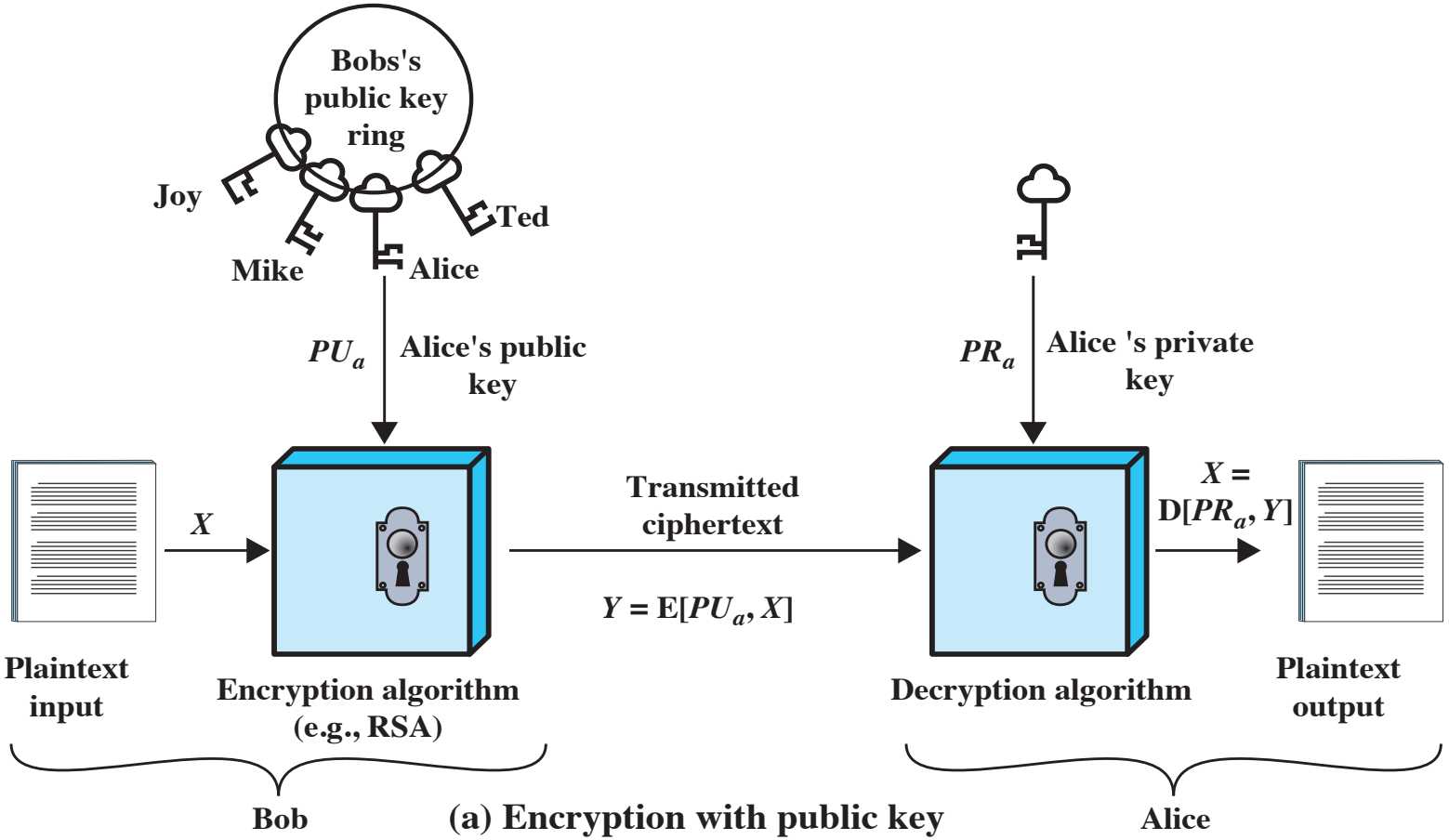
Misconceptions

- Public-key encryption is *more secure* from cryptanalysis than symmetric encryption
  - There is nothing in principle about either symmetric or public-key encryption that makes one superior to another from the point of view of resisting cryptanalysis
- Public-key encryption is a general-purpose technique that has made *symmetric encryption obsolete*
  - Because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that symmetric encryption will be abandoned
- *Key distribution is trivial* when using public-key encryption, compared to the rather cumbersome handshaking involved with key distribution centers for symmetric encryption
  - Some form of protocol is needed, often involving a central agent, and the procedures involved are no simpler or any more efficient than those required for symmetric encryption

Ingredients

- Plaintext: the readable message or data that is fed into the algorithm as input
- Encryption algorithm: performs various transformations on the plaintext
- Public and private key: a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption
  - The exact transformations performed by the algorithm depend on the public or private key that is provided as input
- Ciphertext: the scrambled message produced as output
  - It depends on the plaintext and the key
  - For a given message, two different keys will produce two different ciphertexts
- Decryption algorithm: accepts the ciphertext and the matching key and produces the original plaintext

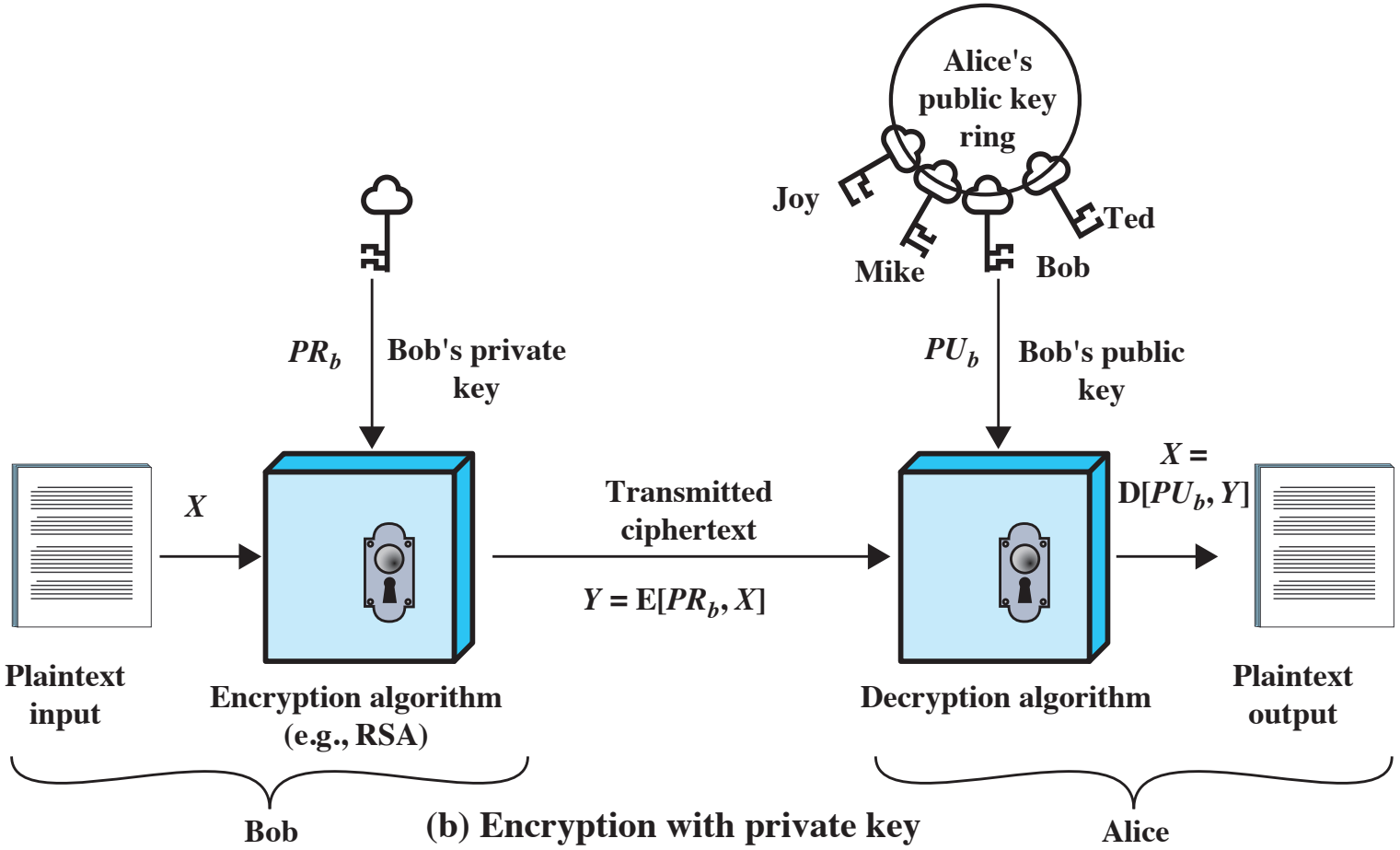
Encryption with public key



(a) Encryption with public key

1. Each user generates a pair of keys to be used for the encryption and decryption of messages
   - The public key is placed in a public register or other accessible file
   - The companion key is kept private. Each user maintains a collection of public keys from others
2. If Bob wishes to send a private message to Alice, Bob encrypts the message using Alice's public key
3. When Alice receives the message, she decrypts it using her private key
   - No other recipient can decrypt the message because only Alice knows Alice's private key

Encryption with private key



- Bob encrypts data using his own private key
- Alice and anyone who knows the corresponding public key will then be able to decrypt the message

Considerations

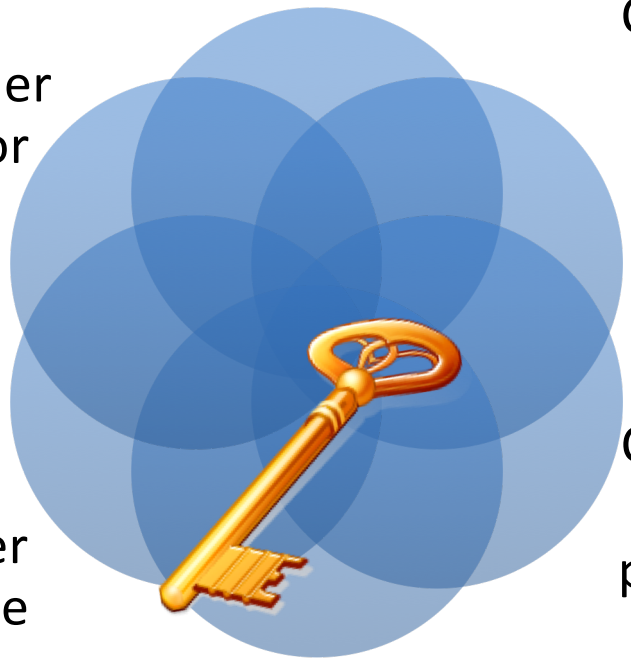
- The scheme of encryption with public key is directed toward providing confidentiality
  - Only the intended recipient should be able to decrypt the ciphertext because only the intended recipient is in possession of the required private key
  - Whether in fact confidentiality is provided depends on a number of factors, including the security of the algorithm, whether the private key is kept secure, and the security of any protocol of which the encryption function is a part
- The scheme of encryption with private key is directed toward providing authentication and/or data integrity
  - If a user is able to successfully recover the plaintext from Bob's ciphertext using Bob's public key, this indicates that only Bob could have encrypted the plaintext, thus providing authentication
  - Further, no one but Bob would be able to modify the plaintext because only Bob could encrypt the plaintext with Bob's private key
  - Once again, the actual provision of authentication or data integrity depends on a variety of factors

Requirements for Public-Key Cryptosystems

Computationally easy to create key pairs

Useful (but not necessary for all applications) if either key can be used for encryption or decryption

Computationally easy for sender knowing public key to encrypt messages



Computationally infeasible for opponent to recover the original message without private key

Computationally easy for receiver knowing private key to decrypt ciphertexts

Computationally infeasible for opponent to determine private key from public key

Public-Key Encryption Algorithms

- RSA (Rivest, Shamir, Adleman)
  - Developed in 1977
  - Most widely accepted and implemented approach to public-key encryption
  - Block cipher in which the plaintext and ciphertext are integers between 0 and *n-1,* for some *n*
- Diffie-Hellman key exchange algorithm
  - The first published public-key algorithm (1976)
  - Enables two users to securely reach agreement about a shared secret to be used as a secret key for subsequent symmetric encryption of messages
- Digital Signature Standard (DSS)
  - Originally proposed in 1991 by NIST
  - Most recent revision amounts to 2013 (FIPS PUB 186-4)
  - Uses an algorithm that is designed to provide only the digital signature function
  - Cannot be used for encryption or key exchange
- Elliptic curve cryptography (ECC)
  - A competitor of RSA
  - As secure as RSA, but with much smaller keys, thereby reducing processing overhead
  - The confidence level in ECC is not yet as high as that in RSA: although the theory of ECC has been around for some time, it is only recently that products have begun to appear and that there has been sustained cryptanalytic interest in probing for weaknesses

Applications for Public-Key Cryptosystems

- Three main categories of applications for public-key cryptosystems
  - digital signature
  - symmetric key distribution
  - encryption of secret keys
- Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications
- Depending on the application, the sender uses either her private key or the receiver's public key, or both, to perform some type of cryptographic function

Algorithm	Digital Signature	Symmetric Key Distribution	Encryption of Secret Keys
Rivest, Shamir, Adleman (RSA)	Yes	Yes	Yes
Diffie-Hellman	No	Yes	No
Digital Signature Standard (DSS)	Yes	No	No
Elliptic Curve Cryptography (ECC)	Yes	Yes	Yes

Index

- Confidentiality with Symmetric Encryption
- Message Authentication and Hash Functions
- Public-Key Encryption
- Digital Signatures and Key Management
- Random and Pseudorandom Numbers
- Practical Application: Encryption of Stored Data

Digital Signatures and Key Management

- Public-key algorithms are used in a variety of applications falling into two categories
  - Digital signatures
  - Key management and distribution
- With respect to key management and distribution, there are at least three distinct aspects
  - The secure distribution of public keys (*Public-Key Certificates*)
  - The distribution of symmetric keys
  - The protection of one-time symmetric keys for message encryption (*Digital Envelopes*)

A definition of digital signature

Digital signature

The result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for verifying origin authentication, data integrity and signatory non-repudiation

NIST FIPS PUB 186-4 [Digital Signature Standard (DSS), July 2013]
(NIST = U.S. National Institute of Standards and Technology)

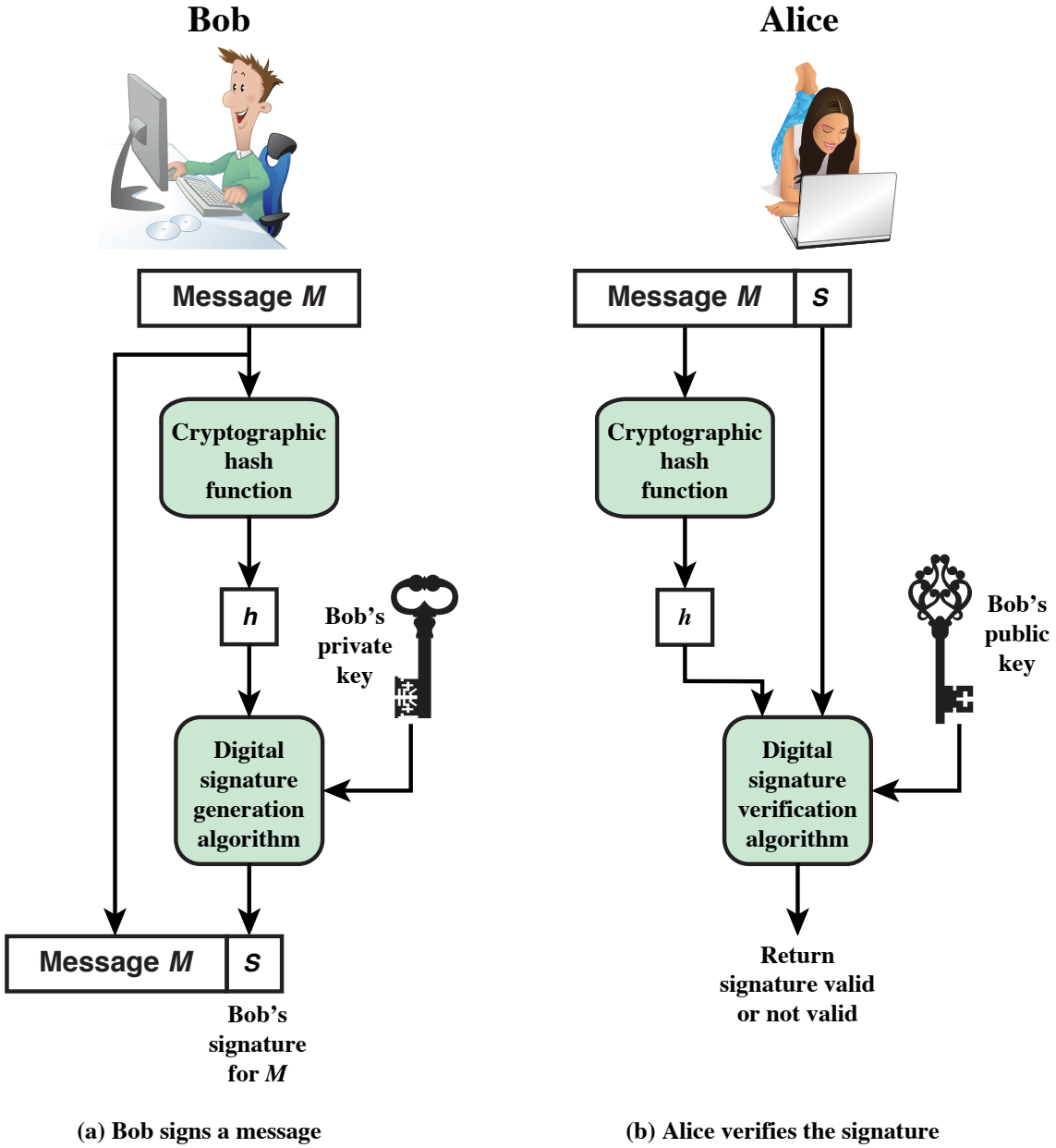
Digital signature

- Thus, a digital signature is a *data-dependent bit pattern*, generated by an agent as a function of a file, message, or other form of data block
- Another agent can access the data block and its associated signature and verify that
  - the data block has been signed by the *alleged signer*
  - the data block has *not been altered* since the signing
- Further, the signer *cannot repudiate* the signature
- Therefore, digital signature is a technique enabling the use of public-key encryption for authentication
- FIPS 186-4 specifies the use of one of three digital signature **algorithms**
  - Digital Signature Algorithm (DSA) (based on the difficulty of computing discrete logarithms)
  - RSA Digital Signature Algorithm (based on RSA)
  - Elliptic Curve Digital Signature Algorithm (ECDSA) (based on ECC)

Digital Signatures

- Suppose that *Bob wants to send a message to Alice* so that she can be certain that the message is indeed from him
  - Bob uses a secure hash function to generate a hash value for the message
  - That hash value, together with Bob's private key, serve as input to a digital signature generation algorithm that produces a short block that functions as a digital signature
  - Bob sends the message with the signature attached
- When Alice receives the message plus signature, she
  - calculates a hash value for the message
  - provides the hash value and Bob's public key as inputs to a digital signature verification algorithm
- If the algorithm returns the result that the signature is valid, Alice is assured that *the message must have been signed by Bob*
  - No one else has Bob's private key and therefore no one else could have created a signature that could be verified for this message with Bob's public key
  - It is impossible to alter the message without access to Bob's private key, so the message is authenticated both in terms of source and in terms of data integrity

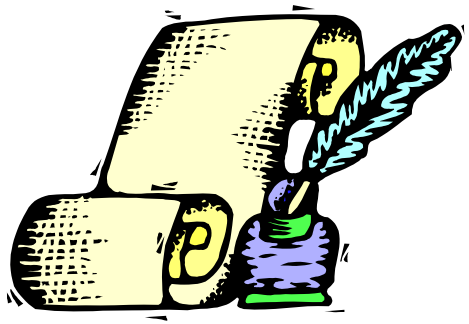
Simplified Depiction of Essential Elements of a Digital Signature Process



(a) Bob signs a message

(b) Alice verifies the signature

Considerations



- Public-key encryption (encrypting with the sender's private key) can be used for
  - sender authentication
  - data integrity
  - signatory non-repudiation
- A digital signature is created by encrypting a hash code with a private key
- Does not provide confidentiality
  - The message being sent is safe from alteration but not from eavesdropping
  - Even in the case the message is completely encrypted, any observer can decrypt the message by using the sender's public key

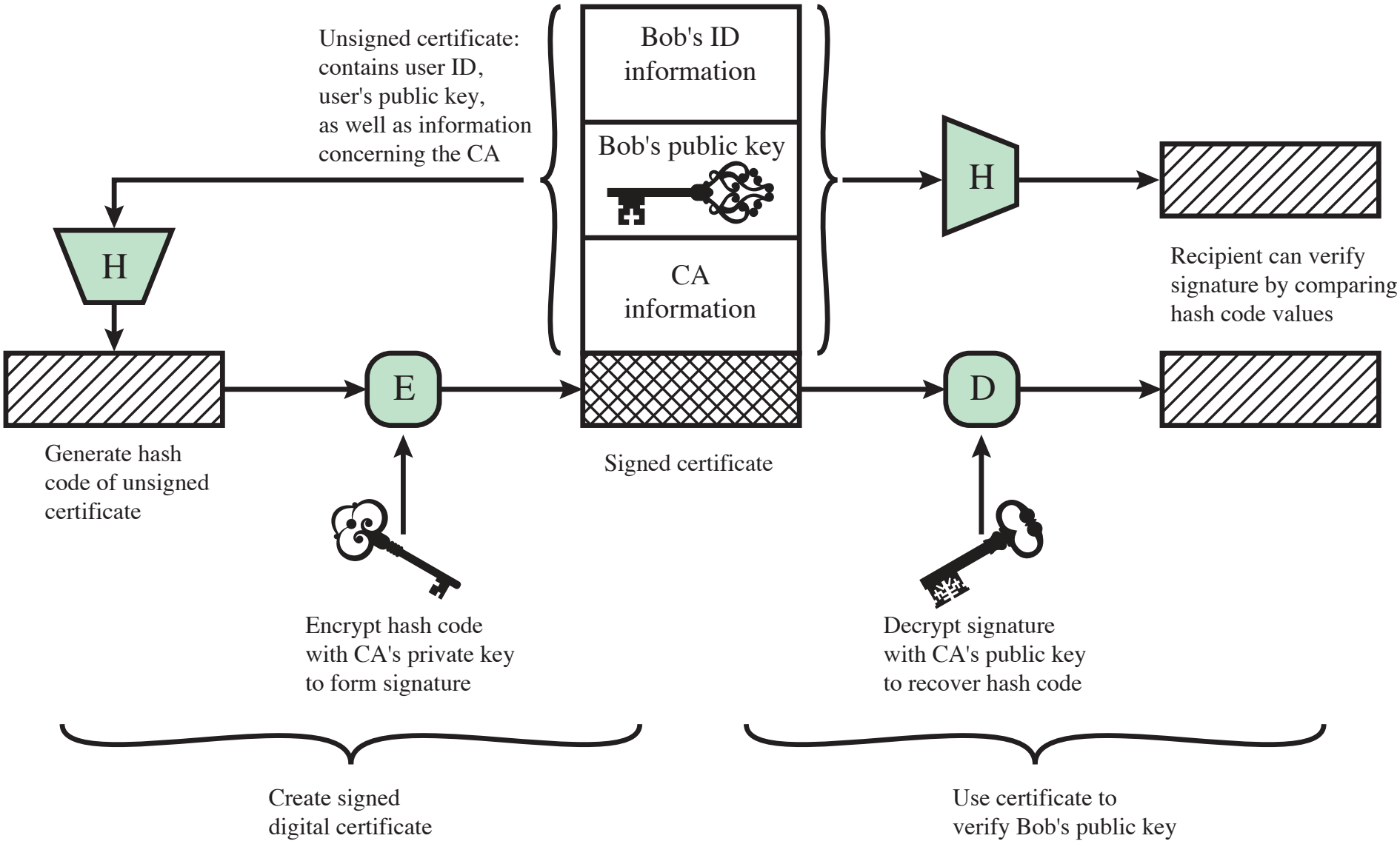
Key management and distribution

- Other than for digital signature applications, public-key algorithms are used also for key management and distribution
- There are at least **three distinct aspects** to the use of public-key encryption in this regard:
  - The secure distribution of public keys (*Public-Key Certificates*)
  - The distribution of symmetric keys
  - The protection of one-time symmetric keys for message encryption (*Digital Envelopes*)

Public-Key Certificates

- Public-key encryption has a major weakness: public keys can be forged
  - Some user could pretend to be Bob and send a public key to another participant or broadcast such a public key
  - Until such time as Bob discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for Bob and to use the forged keys for authentication
- *Solution:* use of public-key certificates
  - A **certificate** consists of a public key plus a user ID of the key owner, with the whole block signed by a trusted third party
  - The certificate also includes some information about the third party plus an indication of the period of validity of the certificate
  - Typically, the third party is a **certificate authority** (CA) that is trusted by the user community, such as a *government agency* or a *financial institution*
  - In practice, a user can present his or her public key to the authority in a secure manner and obtain a signed certificate
  - *The user can then publish the certificate*
  - Anyone needing this user's public key can obtain the certificate and verify that it is valid by means of the attached trusted CA signature

Public-Key Certificate: key steps



The X.509 standard

- One scheme universally accepted for formatting public-key certificates
- Used in most network security applications, including
  - IP Security (IPsec)
  - Transport Layer Security (TLS)
  - Secure Shell (SSH)
  - Secure/Multipurpose Internet Mail Extension (S/MIME)

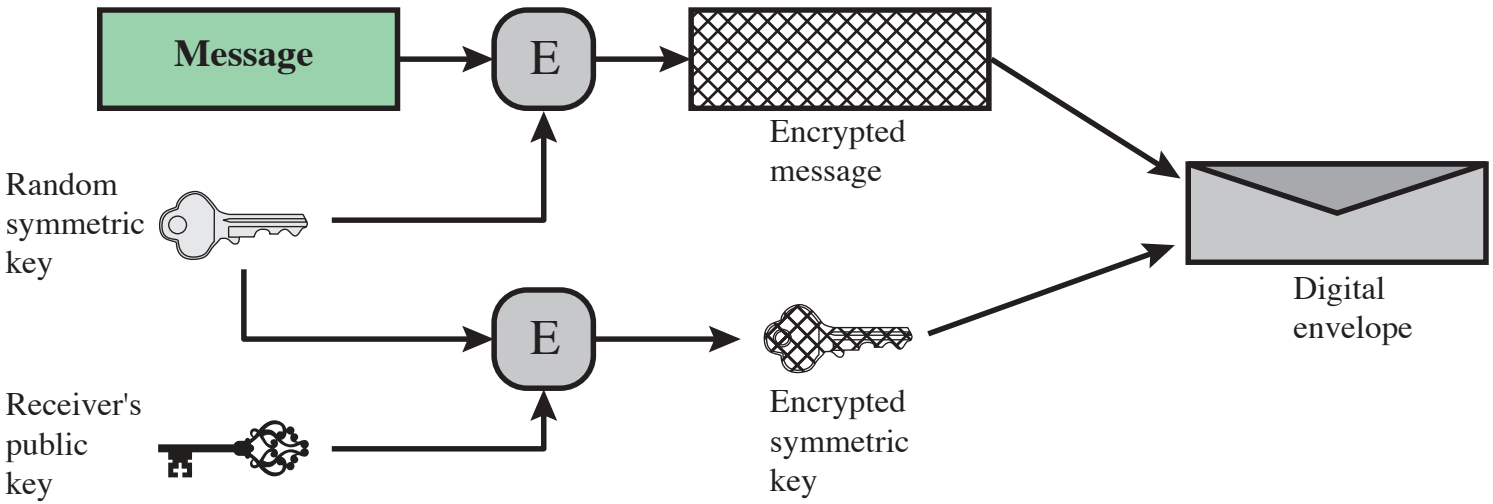
Symmetric Key Exchange Using Public-Key Encryption

- With symmetric encryption, a *fundamental requirement* for two parties to communicate securely is that they share a secret key
- Diffie-Hellman key exchange algorithm can be used to exchange keys among each pair of users that need to share a symmetric key
  - Widely used but, in its simplest form, provides no authentication of the two interacting partners
  - Variations to Diffie-Hellman exist that also ensure authentication
- There are protocols using other public-key algorithms that achieve the same objective

Digital Envelopes

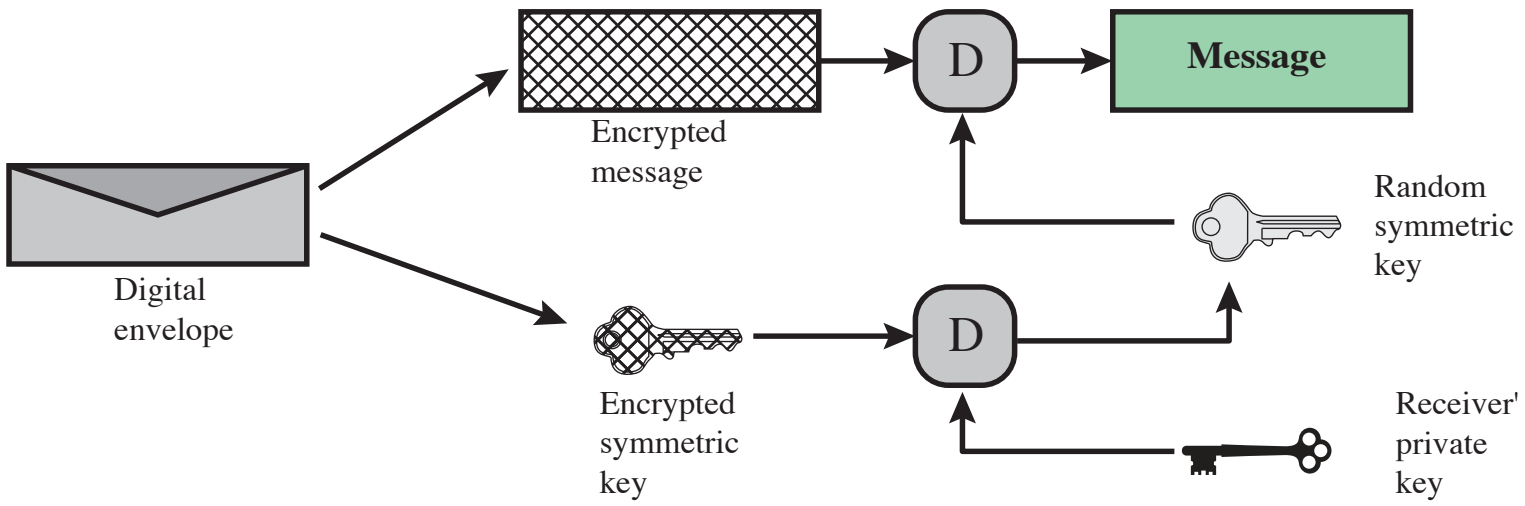
- This technique can be used to protect confidentiality of a message without needing to first arrange for sender and receiver to have the same **secret key**
- A digital envelope is the equivalent of a sealed envelope containing an unsigned letter
- Suppose *Bob wishes to send a confidential message to Alice, but they do not share a symmetric secret key*

Digital Envelopes: Creation



- Bob does the following steps:
  1. Prepare a message
  2. Generate a random symmetric key that will be used this one time only
  3. Encrypt that message using symmetric encryption and the one-time key
  4. Encrypt the one-time key using public-key encryption with Alice's public key
  5. Attach the encrypted one-time key to the encrypted message and send it to Alice

Digital Envelopes: Opening



- Alice makes the reverse steps
  1. Receive the message and separate the encrypted message from the encrypted symmetric key
  2. Dencrypt the encrypted one-time key by using her private key
  3. Dencrypt the message using symmetric encryption and the one-time key
  4. Remove the one-time key
- Properties
  - Only Alice is capable of decrypting the one-time key and therefore of recovering the original message
  - If Bob obtained Alice's public key by means of Alice's public-key certificate, then Bob is assured that it is a valid key

Index

- Confidentiality with Symmetric Encryption
- Message Authentication and Hash Functions
- Public-Key Encryption
- Digital Signatures and Key Management
- Random and Pseudorandom Numbers
- Practical Application: Encryption of Stored Data

(Some) Uses of Random Numbers

- Random numbers play an **important role** in the use of encryption for various network security applications
  - Generation of keys for the RSA and other public-key algorithms
  - Generation of a keystream for symmetric stream ciphers
  - Generation of a symmetric key for use as a temporary session key or in creating a digital envelope
  - Generation of *nonces* to prevent replay attacks in many key distribution scenarios, such as Kerberos
  - Generation of session keys, whether done by a key distribution center or by one of the principals
- These applications give rise to two distinct **requirements** for a *sequence of random numbers*:
  - randomness
  - unpredictability

Random Number Requirements

Randomness: the sequence of numbers must be random in some well-defined statistical sense; two criteria to validate that a sequence is random

- **Uniform distribution**

  Frequency of occurrence of each of the numbers should be approximately the same

- **Independence**

  No one value in the sequence can be inferred from the others

Unpredictability

- Opponent should not be able to predict future elements of the sequence on the basis of earlier elements

- "True" randomness cannot be obtained as there are no tests to "prove" independence (one can instead prove the contrary!)
- Unpredictability poses less stringent requirements and *often suffices*
  - *Unpredicatability may hold, even if independence does not*

Random versus Pseudorandom

- Cryptographic applications typically make use of algorithmic techniques for random number generation
  - *Algorithms are deterministic* and therefore produce sequences of numbers that are not statistically random
  - If the algorithm is good, the resulting sequences will pass many reasonable tests of randomness
- Such numbers are referred to as **pseudorandom numbers**, they are
  - Sequences produced that satisfy statistical randomness tests
  - Likely to be unpredictable
- Under most circumstances, *pseudorandom numbers will perform as well as if they were random*
  - The same principle applies in statistical applications, in which a statistician takes a sample of a population and assumes that the results will be approximately the same as if the whole population were measured
- True random number generator (TRNG):
  - HW devices using a nondeterministic source to produce randomness
  - Most operate by measuring unpredictable natural processes
    - E.g. pulse detectors of ionizing radiation events
  - Increasingly provided on modern processors
    - E.g. Intel digital random number generator offered on new multicore chips since May 2012

Index

- Confidentiality with Symmetric Encryption
- Message Authentication and Hash Functions
- Public-Key Encryption
- Digital Signatures and Key Management
- Random and Pseudorandom Numbers
- Practical Application: Encryption of Stored Data

Practical Application: Encryption of Stored Data

- Common to encrypt transmitted data
- Much less common for stored data (*data at rest*)
  - There is often little protection beyond *domain authentication* and operating system *access controls*
  - Data at rest are backed up to secondary storage, such as hard disk, CD-ROM or tape, archived for indefinite periods
  - Even though erased, until disk sectors are reused data are recoverable
  - Thus it becomes attractive, and indeed should be mandatory, to encrypt data at rest and combine this with an effective encryption key management scheme

Approaches to encrypt stored data

- A simple approach for use on PC is to use a commercially available encryption package, e.g. **Pretty Good Privacy** (PGP)
  - PGP enables a user to generate a key from a password and then use that key to encrypt selected files on the hard disk
  - To recover a file, the user enters the password, PGP generates the key and decrypts the file
  - The files are fully protected while at rest, so long as the user protects his or her password (PGP does not store the password) and does not use an easily guessable password
  - Impacts on performance, as any other SW encryption

Some more recent approaches

- **Back-end appliance**: HW device sitting between servers and storage systems
  - Encrypts all data going from the server to the storage system and decrypts data going in the opposite direction
  - These devices encrypt data at close to wire speed, with very little latency
  - In contrast, encryption SW on servers and storage systems slows backups
- **Library-based tape encryption**: co-processor board embedded in the tape drive HW and tape library
  - The co-processor encrypts data using a nonreadable key configured into the board
  - The tapes can then be sent off-site to a facility that has the same tape drive HW
  - If the matching tape drive HW co-processor is not available at the other site, the target facility can use the key in a SW decryption package to recover the data
  - The key can be exported via secure e-mail or a small flash drive that is transported securely
- **Background laptop/PC data encryption**: many vendors offer SW products that provide encryption that is transparent to the application and the user
  - Some products encrypt all or designated files and folders; other products create a virtual disk, which can be maintained locally on the user's hard drive or maintained on a network storage device, with all data on the virtual disk encrypted
  - Various key management solutions are offered to restrict access to the owner of the data

Summary

- Confidentiality with Symmetric Encryption
  - Symmetric Encryption
  - Symmetric Block Encryption Algorithms Stream Ciphers
- Message Authentication and Hash Functions
  - Authentication Using Symmetric Encryption
  - Message Authentication without Message Encryption
  - Secure Hash Functions
  - Other Applications of Hash Functions
- Public-Key Encryption
  - Public-Key Encryption
  - Structure Applications for Public-Key Cryptosystems
  - Requirements for Public-Key Cryptography
  - Asymmetric Encryption Algorithms
- Digital Signatures and Key Management
  - Digital Signature
  - Public-Key Certificates
  - Symmetric Key Exchange Using Public-Key Encryption
  - Digital Envelopes
- Random and Pseudorandom Numbers
  - The Use of Random Numbers
  - Random versus Pseudorandom
- Practical Application: Encryption of Stored Data