Chapter 3

Principles of Public-Key Cryptosystems

The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption. key distribution under symmetric encryption requires either (1) that two communicants already share a key, which somehow has been distributed to them; or (2) the use of a key distribution center. Whitfield Diffie, one of the discoverers of public-key encryption (along with Martin Hellman, both at Stanford University at the time), reasoned that this second requirement negated the very essence of cryptography: the ability to maintain total secrecy over your own communication.

The second problem that Diffie pondered, and one that was apparently unrelated to the first was that of "digital signatures." If the use of cryptography was to become widespread, not just in military situations but for commercial and private purposes, then electronic messages and documents would need the equivalent of signatures used in paper documents.

Public-Key Cryptosystems

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic:

It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristic:

Either of the two related keys can be used for encryption, with the other used for decryption.

A public-key encryption scheme has six ingredients

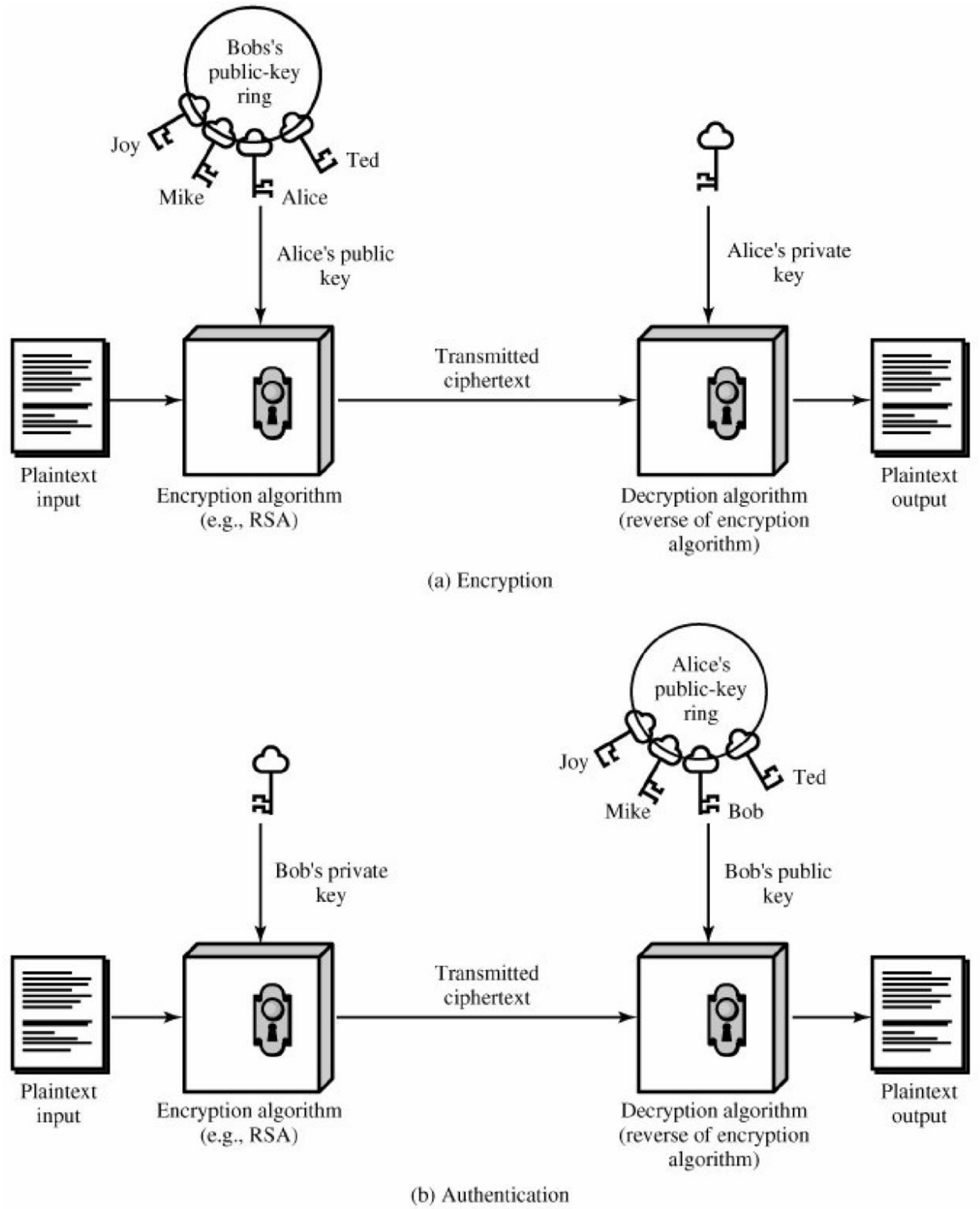
Plaintext: This is the readable message or data that is fed into the algorithm as input.

Encryption algorithm: The encryption algorithm performs various transformations on the plaintext.

Public and private keys: This is 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: This is 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: This algorithm accepts the ciphertext and the matching key and produces the original plaintext.



The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.

2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As Figure 9.1a suggests, each user maintains a collection of public keys obtained from others.

3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.

4. 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.

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a user's private key remains protected and secret, incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

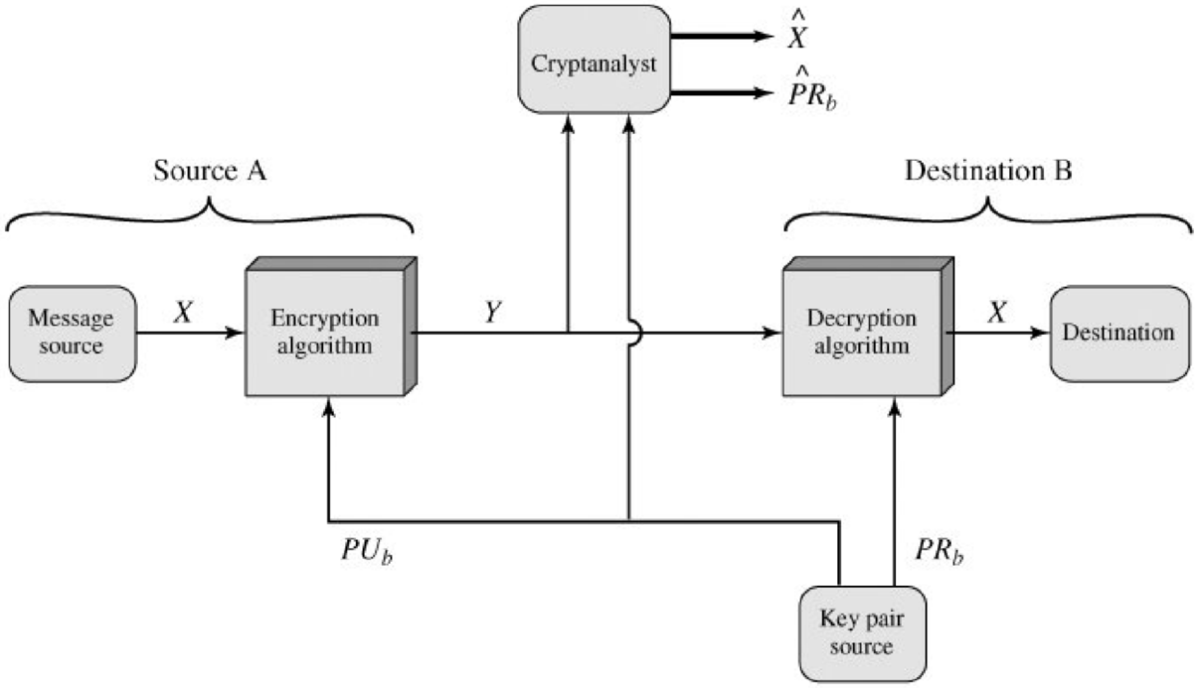
Table 9.1 summarizes some of the important aspects of symmetric and public-key encryption. To discriminate between the two, we refer to the key used in symmetric encryption as a **secret key**. The two keys used for asymmetric encryption are referred to as the **public key** and the **private key**.

[2] Invariably, the private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with symmetric encryption

Table 9.1. Conventional and Public-Key Encryption

Conventional Encryption	Public-Key Encryption
Needed to Work: The same algorithm with the same key is used for encryption and decryption. The sender and receiver must share the algorithm and the key.	**Needed to Work:** One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. The sender and receiver must each have one of the matched pair of keys (not the same one).
Needed for Security: Needed The key must be kept secret. It must be impossible or at least impractical to decipher a message if no other information is available. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	**Needed for Security: Needed** One of the two keys must be kept secret. It must be impossible or at least impractical to decipher a message if no other information is available. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Let us take a closer look at the essential elements of a public-key encryption scheme, using Figure 9.2 (compare with Figure 2.2). There is some source A that produces a message in plaintext, *X* =[*X*1, *X*2,..., *XM*,]. The *M* elements of *X* are letters in some finite alphabet. The message is intended for destination B. B generates a related pair of keys: a public key, *PUb*, and a private key, *PUb*. *PUb* is known only to B, whereas *PUb* is publicly available and therefore accessible by A.



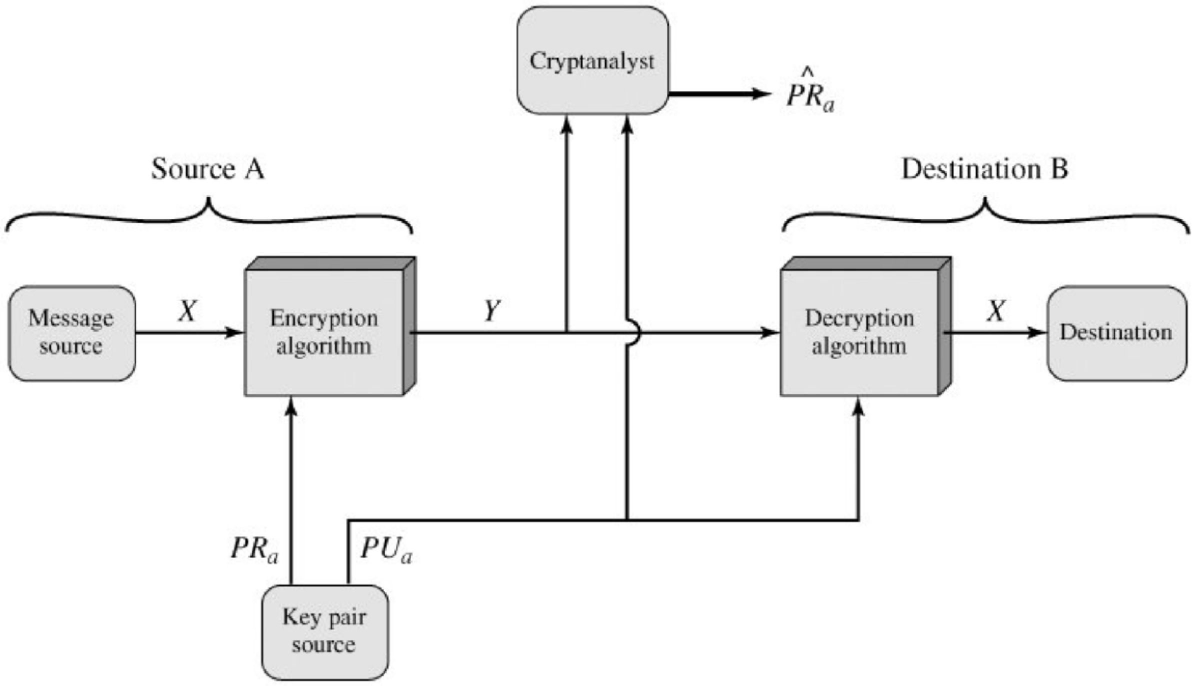
With the message *X* and the encryption key *PUb* as input, A forms the ciphertext *Y* = [*Y*1, *Y*2,..., *YN*]:

Y = E(*PUb*, *X*)

The intended receiver, in possession of the matching private key, is able to invert the transformation:

X = D(*PRb*, *Y*)

An adversary, observing *Y* and having access to *PUb* but not having access to *PRb* or *X*, must attempt to recover *X* and/or *PRb*. It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms. If the adversary is interested only in this particular message, then the focus of effort is to recover *X*, by generating a plaintext estimate ˆX



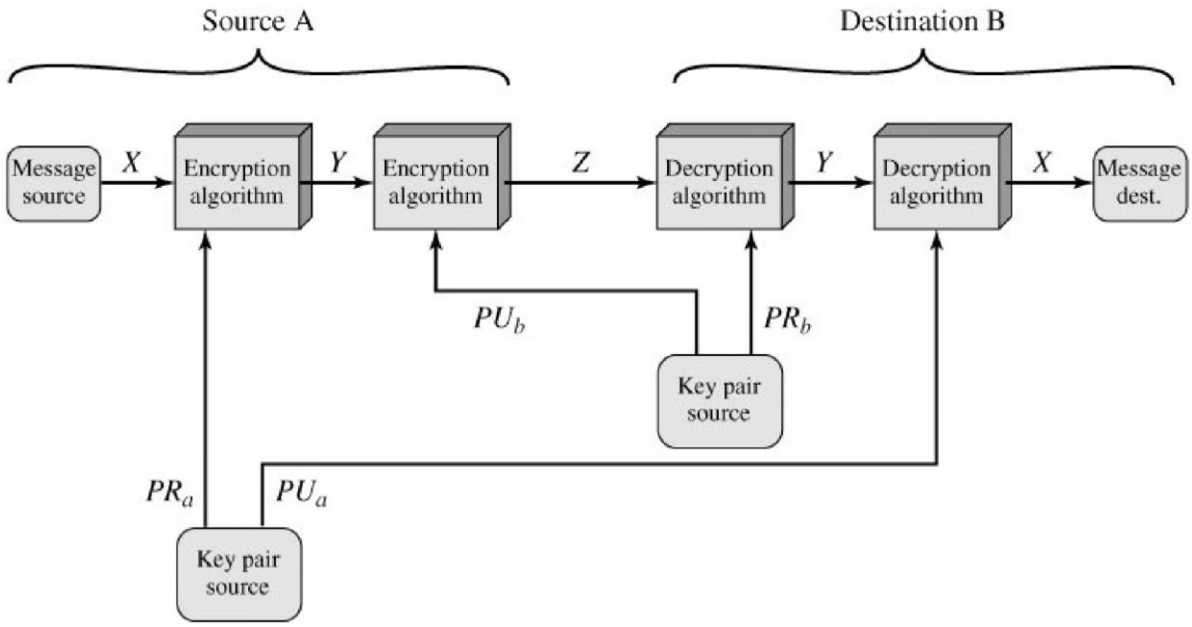
In this case, A prepares a message to B and encrypts it using A's private key before transmitting it. B can decrypt the message using A's public key. Because the message was encrypted using A's private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a *digital signature*. In addition, it is impossible to alter the message without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity.

In the preceding scheme, the entire message is encrypted, which, although validating both author and contents, requires a great deal of storage. Each document must be kept in plaintext to be used for practical purposes. A copy also must be stored in ciphertext so that the origin and contents can be verified in case of a dispute. A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted with the sender's private key, it serves as a signature that verifies origin, content, and sequencingt is important to emphasize that the encryption process depicted in Figures 9.1b and 9.3 does not provide confidentiality. That is, the message being sent is safe from alteration but not from eavesdropping. This is obvious in the case of a signature based on a portion of the message, because the rest of the message is transmitted in the clear. Even in the case of complete encryption, as shown in Figure 9.3, there is no protection of confidentiality because any observer can decrypt the message by using the sender's public key.

It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme (Figure 9.4):

Z = E(PUb, E(PRa, X))

X = D(PUa, E(PRb, Z))



In this case, we begin as before by encrypting a message, using the sender's private key. This provides the digital signature. Next, we encrypt again, using the receiver's public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided. The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

Applications for Public-Key Cryptosystems

Before proceeding, we need to clarify one aspect of public-key cryptosystems that is otherwise likely to lead to confusion. Public-key systems are characterized by the use of a cryptographic algorithm with two keys, one held private and one available publicly. Depending on the application, the sender uses either the sender's private key or the receiver's public key, or both, to perform some type of cryptographic function. In broad terms, we can classify the use of public-key cryptosystems into three categories:

Encryption/decryption: The sender encrypts a message with the recipient's public key.

Digital signature: The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.

Key exchange: Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Table 9.2. Applications for Public-Key Cryptosystems

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

The RSA Algorithm

The pioneering paper by Diffie and Hellman [DIFF76b] introduced a new approach to cryptography and, in effect, challenged cryptologists to come up with a cryptographic algorithm that met the requirements for public-key systems. One of the first of the responses to the challenge was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978. The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption.

The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and *n* 1 for some*n*. A typical size for *n* is 1024 bits, or 309 decimal digits. That is, *n* is less than 2 1024 . We examine RSA in this section in some detail, beginning with an explanation of the algorithm. Then we examine some of the computational and cryptanalytical implications of RSA.

Description of the Algorithm

The scheme developed by Rivest, Shamir, and Adleman makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number *n*. That is, the block size must be less than or equal to $\log_2(n)$; in practice, the block size is *i* bits, where $2^i < n < 2^{i+1}$

i+1

. Encryption and decryption are of the following form, for some plaintext block *M* and ciphertext block *C*:

$C = M^e \bmod n$

$M = C^d \bmod n = (Me)^d \bmod n = M^{ed} \bmod n$

Both sender and receiver must know the value of *n*. The sender knows the value of *e*, and only the receiver knows the value of *d*. Thus, this is a public-key encryption algorithm with a public key of *PU* = {*e*, *n*} and a private key of *PU* = {*d*, *n*}. For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:

1. It is possible to find values of *e*, *d*, *n* such that $M^{ed} \bmod n = M$ for all $M < n$.

2. It is relatively easy to calculate mod $M^e \bmod n$ and C^d for all values of $M < n$.

3. It is infeasible to determine *d* given *e* and *n*.

For now, we focus on the first requirement and consider the other questions later. We need to find a relationship of the form

$M^{ed} \bmod n = M$

The preceding relationship holds if *e* and *d* are multiplicative inverses modulo f(*n*), where f(*n*) is the Euler totient function. It is shown that for *p*, *q* prime, f(*pq*) = (*p* 1)(*q* 1) The relationship between *e* and *d* can be expressed as

$$ed \bmod \phi(n) = 1$$

This is equivalent to saying

$ed \equiv 1 \bmod f(n)$

$d\, e \equiv 1 \bmod f(n)$

That is, *e* and *d* are multiplicative inverses mod f(*n*). Note that, according to the rules of modular arithmetic, this is true only if *d* (and therefore *e*) is relatively prime to f(*n*). Equivalently, gcd(f(*n*),*d*) = 1.

We are now ready to state the RSA scheme. The ingredients are the following:

p,*q*, two prime numbers	(private, chosen)
$n = pq$	(public, calculated)
e, with $\gcd(f(n),e) = 1; 1 < e < f(n)$	(public, chosen)
$d\, e \equiv 1 (\bmod f(n))$	(private, calculated)

The private key consists of {*d*, *n*} and the public key consists of {*e*, *n*}. Suppose that user A has published its public key and that user B wishes to send the message *M* to A. Then B calculates $C = M^e \bmod n$ and transmits C. On receipt of this ciphertext, user A decrypts by calculating $M = C^d \bmod n$.

For this example, the keys were generated as follows:

1. Select two prime numbers, $p = 17$ and $q = 11$.

2. Calculate $n = pq = 17 \times 11 = 187$.

3. Calculate f(n) = (p 1)(q 1) = 16 x 10 = 160.

4. Select e such that e is relatively prime to f(n) = 160 and less than f(n) we choose $e = 7$.

5. Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because 23 x 7 = 161 = 10 x 160 + 1; d can be calculated using the extended Euclid's algorithm

Figure 9.5. The RSA Algorithm

Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

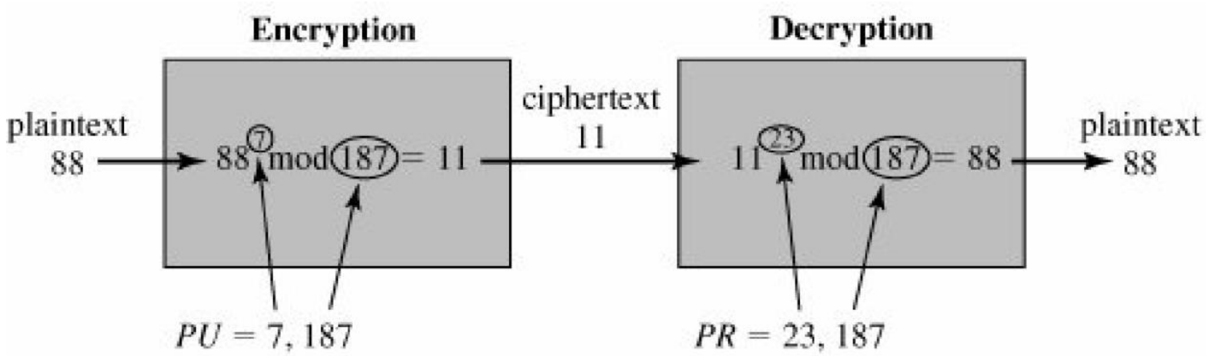
Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Figure 9.6. Example of RSA Algorithm



The resulting keys are public key *PU* = {7,187} and private key *PR* = {23,187}. The example shows the use of these keys for a plaintext input of *M* = 88. For encryption, we need to calculate $C = 88^7$ mod 187. Exploiting the properties of modular arithmetic, we can do this as follows:

88^7mod 187 = [(88^4mod 187) x (88^2mod 187) x (88^1mod 187)] mod 187
88^1mod 187 = 88
88^2mod 187 = 7744 mod 187 = 77
88^4mod 187 = 59,969,536 mod 187 = 132
88^7mod 187 = (88 x 77 x 132) mod 187 = 894,432 mod 187 = 11

For decryption, we calculate $M = 11^{23}$mod 187:
11^{23}mod 187 = [(11^1mod 187) x (11^2mod 187) x (11^4mod 187) x (11^8mod 187) x (11^8mod 187)] mod 187
11^1mod 187 = 11
11^2mod 187 = 121
11^4mod 187 = 14,641 mod 187 = 55
11^8mod 187 = 214,358,881 mod 187 = 33
11^{23}mod 187 = (11 x 121 x 55 x 33 x 33) mod 187 = 79,720,245 mod 187 = 88

The Security of RSA

Four possible approaches to attacking the RSA algorithm are as follows:
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.
The defense against the brute-force approach is the same for RSA as for other cryptosystems, namely, use a large key space. Thus, the larger the number of bits in *d*, the better. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run.

Key Management

One of the major roles of public-key encryption has been to address the problem of key distribution. There are actually two distinct aspects to the use of public-key cryptography in this regard:
The distribution of public keys
The use of public-key encryption to distribute secret keys

Distribution of Public Keys

Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:
Public announcement
Publicly available directory
Public-key authority
Public-key certificates

Public Announcement of Public Keys

On the face of it, the point of public-key encryption is that the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large (Figure 10.1). For example, because of the growing popularity of PGP which makes use of RSA, many PGP users have adopted the practice of appending their public key to messages that they send to public forums, such as USENET newsgroups and Internet mailing lists.

Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication

Publicly Available Directory

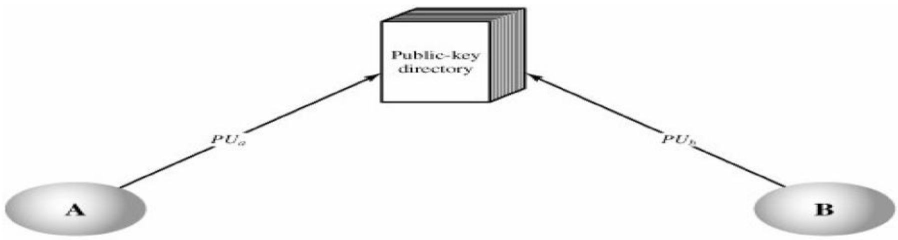
A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization (Figure 10.2). Such a scheme would include the following elements:

1. The authority maintains a directory with a {name, public key} entry for each participant.

2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.

3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way.
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

Figure 10.2. Public-Key Publication



This scheme is clearly more secure than individual public announcements but still has vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Another way to achieve the same end is for the adversary to tamper with the records kept by the authority.

Public-Key Authority

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. A typical scenario is illustrated in Figure 10.3, which is based on a figure in [POPE79]. As before, the scenario assumes that a central authority maintains a dynamic directory of public keys of all participants. In addition, each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key. The following steps (matched by number to Figure 10.3) occur:

1. A sends a timestamped message to the public-key authority containing a request for the current public key of B.

2. The authority responds with a message that is encrypted using the authority's private key, *PRauth* Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following: B's public key, *PUb* which A can use to encrypt messages destined for B

The original request, to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority

The original timestamp, so A can determine that this is not an old message from the authority containing a key other than B's current public key

3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (*IDA*) and a nonce (*N*1), which is used to identify this transaction uniquely.

4, 5. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.

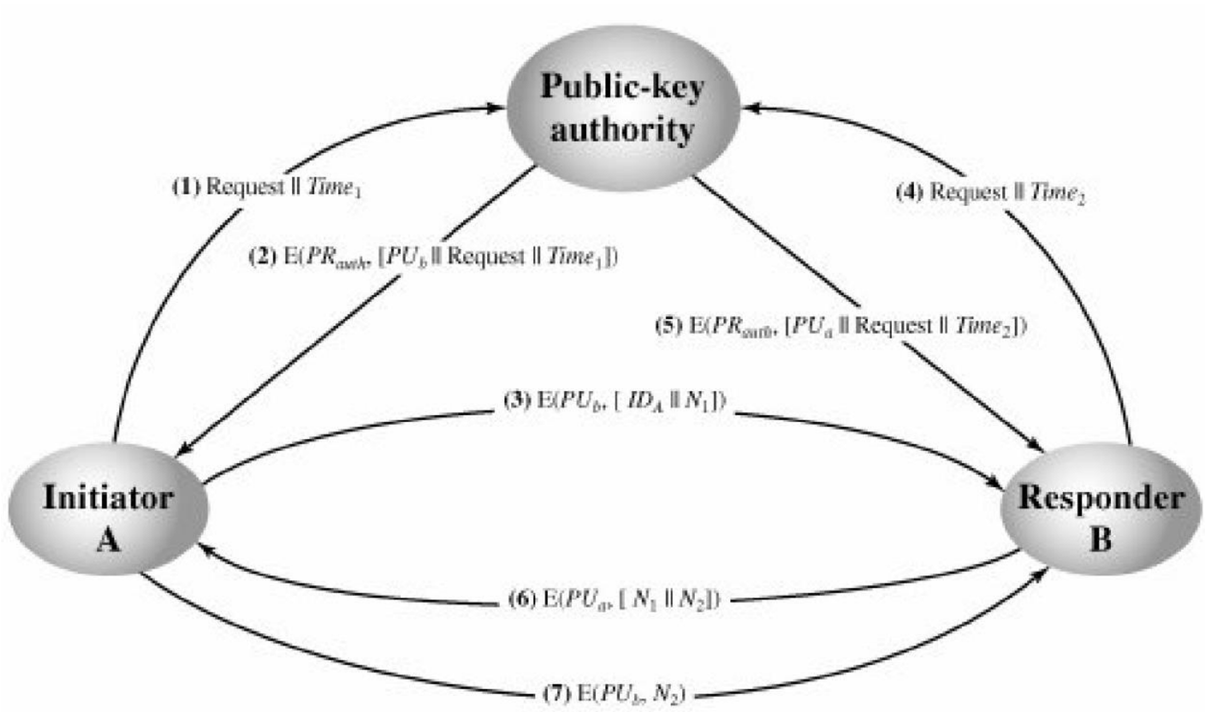
At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

6. B sends a message to A encrypted with *PUa* and containing A's nonce (*N*1) as well as a new nonce generated by B (*N*2)

Because only B could have decrypted message (3), the presence of *N*1 in message (6) assures A that the correspondent is B.

7. A returns *N*2, encrypted using B's public key, to assure B that its correspondent is A.

Figure 10.3. Public-Key Distribution Scenario



Thus, a total of seven messages are required. However, the initial four messages need be used only infrequently because both A and B can save the other's public key for future use, a technique known as caching. Periodically, a user should request fresh copies of the public keys of its correspondents to ensure currency.

Public-Key Certificates

The scenario of Figure 10.3 is attractive, yet it has some drawbacks. The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact. As before, the directory of names and public keys maintained by the authority is vulnerable to tampering.

An alternative approach, first suggested by Kohnfelder [KOHN78], is to use **certificates** that can be used by participants to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. In essence, a certificate consists of a public key plus an identifier of the key owner, with the whole block signed by a trusted third party.

Typically, the third party is a certificate authority, such as a government agency or a financial institution, that is trusted by the user community. A user can present his or her public key to the authority in a secure manner, and obtain a certificate. The user can then publish the certificate. Anyone needed this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority. We can place the following requirements on this scheme:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.

2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.

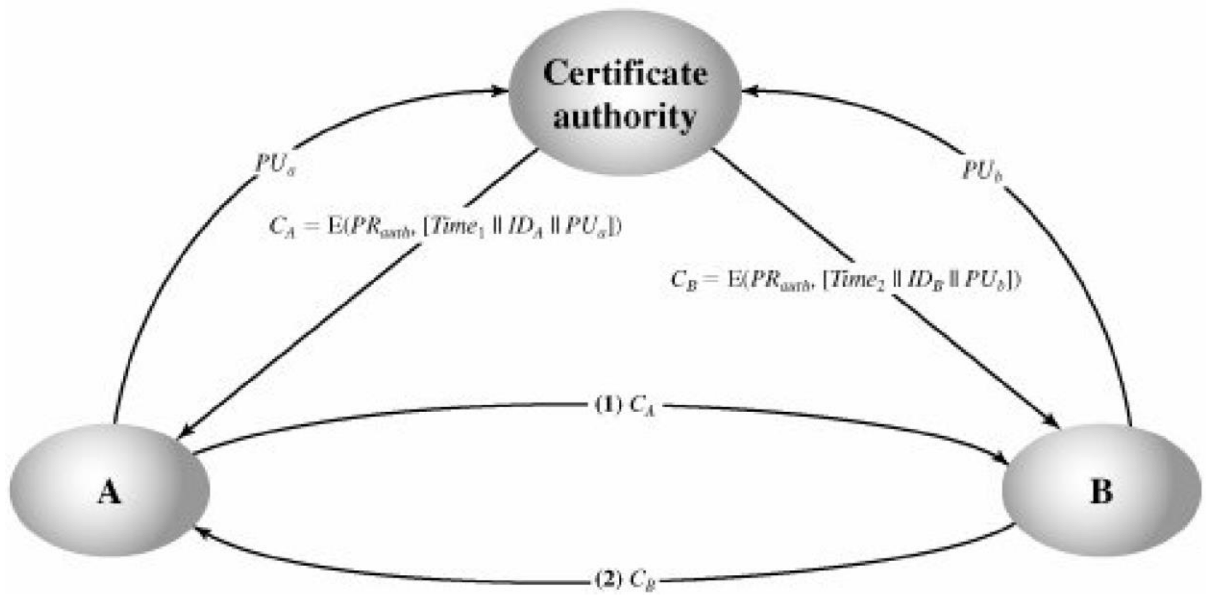
3. Only the certificate authority can create and update certificates.

These requirements are satisfied by the original proposal in [KOHN78]. Denning [DENN83] added the following additional requirement:

4. Any participant can verify the currency of the certificate.

A certificate scheme is illustrated in Figure 10.4. Each participant applies to the certificate authority, supplying a public key and requesting a certificate.

Figure 10.4. Exchange of Public-Key Certificates



Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form

CA = E(*PRauth*, [*T*||*IDA*||*PUa*])

where *PRauth* is the private key used by the authority and *T* is a timestamp. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

D(*PUauth*, *CA*) = D(*PUauth*, E(*PR*auth, [*T*||*IDA*||*PUa*])) = (*T*||*IDA*||*PUa*)

The recipient uses the authority's public key, *PUauth* to decrypt the certificate. Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority. The elements *IDA* and *PUa* provide the recipient with the name and public key of the certificate's holder. The timestamp *T* validates the currency of the certificate. The timestamp counters the following scenario. A's private key is learned by an adversary. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages.

In this context, the compromise of a private key is comparable to the loss of a credit card. The owner cancels the credit card number but is at risk until all possible communicants are aware that the old credit card is obsolete. Thus, the timestamp serves as something like an expiration date. If a certificate is sufficiently old, it is assumed to be expired.

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME.

Diffie-Hellman Key Exchange

The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography and is generally referred to as Diffie-Hellman key exchange.

A number of commercial products employ this key exchange technique.

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.

The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. Briefly, we can define the discrete logarithm in the following way. First, we define a primitive root of a prime number *p* as one whose powers modulo *p* generate all the integers from 1 to *p* 1. That is, if *a* is a primitive root of the prime number *p*, then the numbers

a mod p, a^2 mod p,..., a^{p1} mod p are distinct and consist of the integers from 1 through *p* 1 in some permutation.

For any integer *b* and a primitive root *a* of prime number *p*, we can find a unique exponent *i* such that

$$b \equiv a^i (\text{mod } p) \text{ where } 0 \leq i \leq (p\ 1)$$

The exponent *i* is referred to as the discrete logarithm of *b* for the base *a*, mod *p*.

The Algorithm

Figure 10.7 summarizes the Diffie-Hellman key exchange algorithm. For this scheme, there are two publicly known numbers: a prime number *q* and an integer that is a primitive root of *q* . Suppose the users A and B wish to exchange a key. User A selects a random integer *X*r <*A* and computes YA = a ^{XA} mod *q*. Similarly, user B independently selects a random integer*X A*

< q and computes YB = a XB mod 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
$K = (YB)^{XA}$ mod q and user B computes the key as $K = (YA)^{XB}$ mod q. These two calculations produce identical results:

$K = (YB)^{XA}$ mod q
$= (a^{XB} \bmod q)^{XA} \bmod q$
$= (aXB)^{XA} \bmod q$ by the rules of modular arithmetic
$= (a^{XB\,XA} \bmod q$
$= (aXA)^{XB} \bmod q$
$= (a^{XA} \bmod q)$
$= (a^{XA} \bmod q)^{XB} \bmod q$
$= (YA)^{XB} \bmod q$

Figure 10.7. The Diffie-Hellman Key Exchange Algorithm

Global Public Elements

q	prime number
α	$\alpha < q$ *and* α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Calculation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Calculation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Authentication Functions

Any message authentication or digital signature mechanism has two levels of functionality. At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message.

This section is concerned with the types of functions that may be used to produce an authenticator. These may be grouped into three classes, as follows:

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

Hash function: A function that maps a message of any length into a fixed-length hash value, which serves as the authenticator

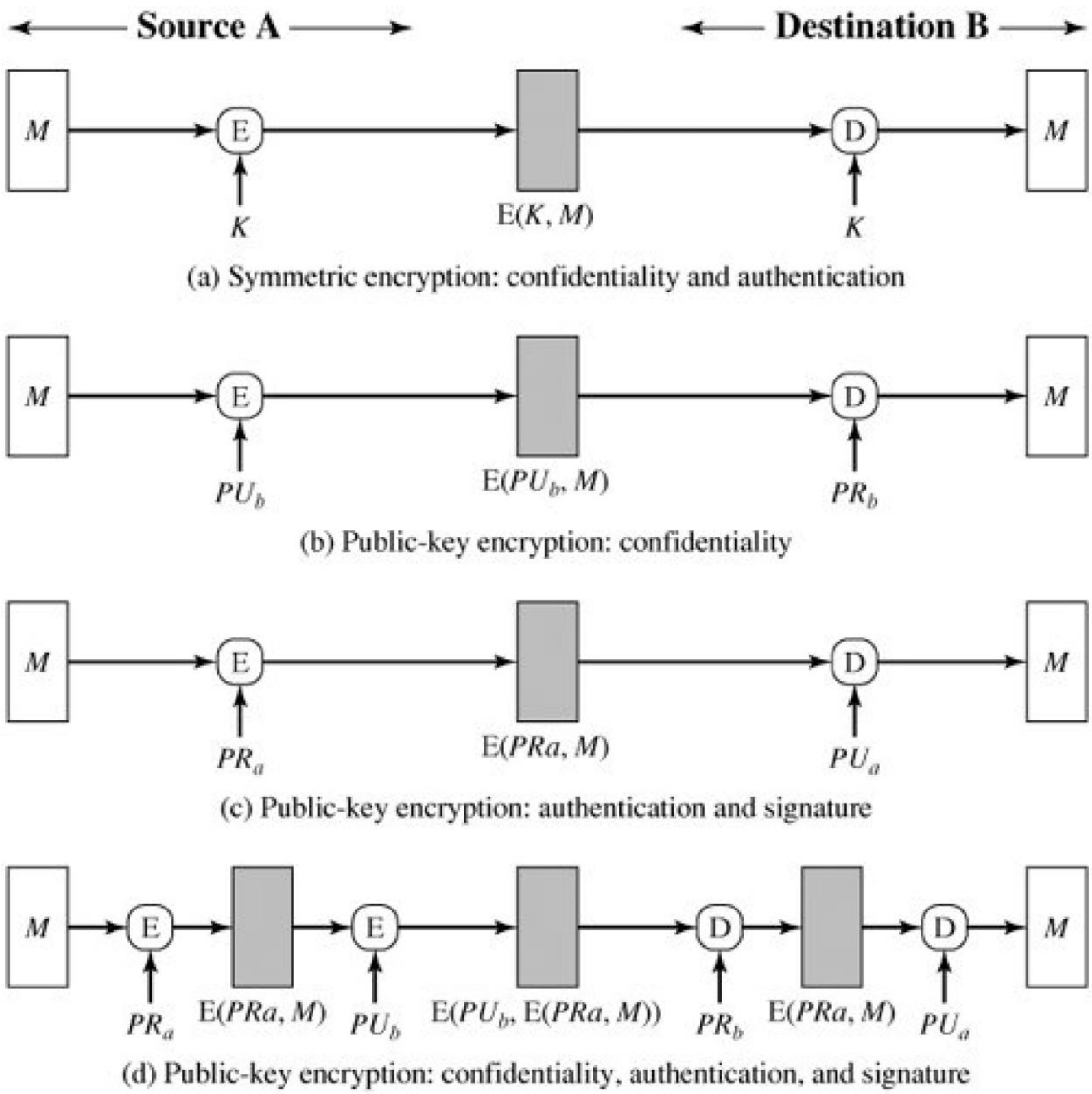
Message Encryption

Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryptionschemes.

Symmetric Encryption

Consider the straightforward use of symmetric encryption (Figure 11.1a). A message *M* transmitted from source A to destination B is encrypted using a secret key *K* shared by A and B. If no other party knows the key, then confidentiality is provided: No other party can recover the plaintext of the message.

Figure 11.1. Basic Uses of Message Encryption



Hash Function

A variation on the message authentication code is the one-way hash function. As with the message authentication code, a hash function accepts a variable-size message *M* as input and produces a fixed-size output, referred to as a **hash code** H(*M*). Unlike a MAC, a hash code does not use a key but is a function only of the input message. The hash code is also referred to as a **message digest** or **hash value**. The hash code is a function of all the bits of the message and provides an error-detection capability: A change to any bit or bits in the message results in a change to the hash code.

Figure 11.5 illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows:

a. The message plus concatenated hash code is encrypted using symmetric encryption. This is identical in structure to the internal error control strategy shown in Figure 11.2a. The same line of reasoning applies: Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

b. Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality. Note that the combination of hashing and encryption results in an overall function that is, in fact, a MAC (Figure 11.4a). That is, E(*K*, H(*M*)) is a function of a variable-length message *M* and a secret key *K*, and it produces a fixed-size output that is secure against an opponent who does not know the secret key.

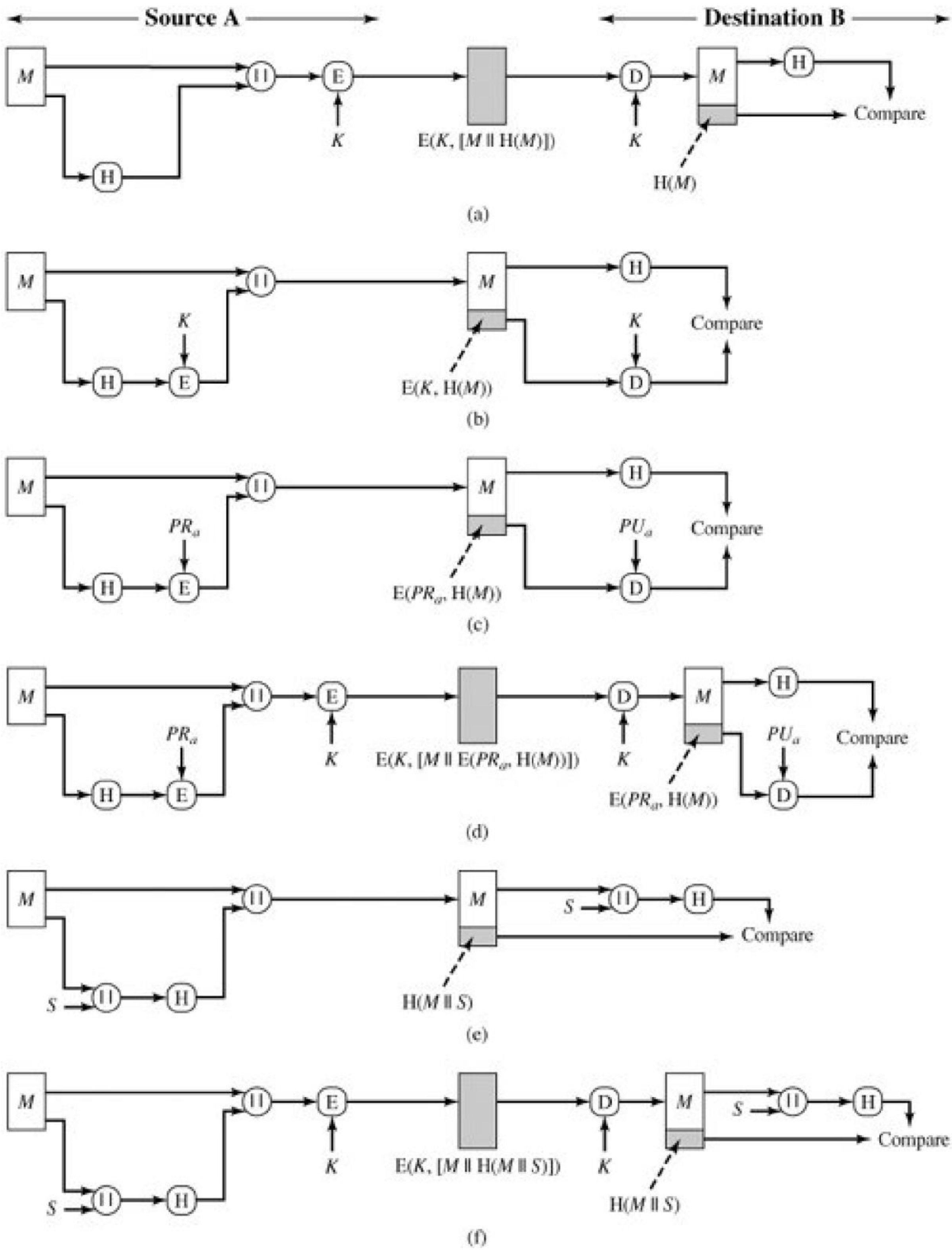
c. Only the hash code is encrypted, using public-key encryption and using the sender's private key. As with (b), this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.

d. If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.

e. It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value *S*. A computes the hash value over the concatenation of *M* and *S* and appends the resulting hash value to *M*. Because B possesses *S*, it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

f. Confidentiality can be added to the approach of (e) by encrypting the entire message plus the hash code.

Figure 11.5. Basic Uses of Hash Function



When confidentiality is not required, methods (b) and (c) have an advantage over those that encrypt the entire message in that less computation is required. Nevertheless, there has been growing interest in techniques that avoid encryption (Figure 11.5e). Several reasons for this interest are pointed out in [TSUD92]:

- Encryption software is relatively slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
- Encryption hardware costs are not negligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.
- Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
- Encryption algorithms may be covered by patents. For example, until the patent expired, RSA was patented and had to be licensed, adding a cost