This page is intentionally left blank

PART THREE: ASYMMETRIC CIPHERS

CHAPTER 9

PUBLIC-KEY CRYPTOGRAPHY AND RSA

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Present an overview of the basic principles of public-key cryptosystems.
- Explain the two distinct uses of public-key cryptosystems.
- List and explain the requirements for a public-key cryptosystem.
- Present an overview of the RSA algorithm.
- Understand the timing attack.
- Summarize the relevant issues related to the complexity of algorithms.

The development of public-key, or asymmetric, cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography. From its earliest beginnings to modern times, virtually all cryptographic systems have been based on the elementary tools of substitution and permutation. After millennia of working with algorithms that could be calculated by hand, a major advance in symmetric cryptography occurred with the development of the rotor encryption/decryption machine. The electromechanical rotor enabled the development of fiendishly complex cipher systems. With the availability of computers, even more complex systems were devised, the most prominent of which was the Lucifer effort at IBM that culminated in the Data Encryption Standard (DES). But both rotor machines and DES, although representing significant advances, still relied on the bread-and-butter tools of substitution and permutation.

Public-key cryptography provides a radical departure from all that has gone before. For one thing, public-key algorithms are based on mathematical functions rather than on substitution and permutation. More important, public-key cryptography is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication, as we shall see.

Before proceeding, we should mention several common misconceptions concerning public-key encryption. One such misconception is that public-key encryption is more secure from cryptanalysis than is symmetric encryption. In fact, the security of any encryption scheme depends on the length of the key and the computational work involved in breaking a cipher. 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.

A second misconception is that public-key encryption is a general-purpose technique that has made symmetric encryption obsolete. On the contrary, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that symmetric encryption will be abandoned. As one of the inventors of public-key encryption has put it [DIFF88], "the restriction of public-key cryptography to key management and signature applications is almost universally accepted."

Table 9.1 Terminology Related to Asymmetric Encryption

Asymmetric Keys Two related keys, a public key and a private key, that are used to perform complementary operations, such as encryption and decryption or signature generation and signature verification.
Public Key Certificate A digital document issued and digitally signed by the private key of a Certification Authority that binds the name of a subscriber to a public key. The certificate indicates that the subscriber identified in the certificate has sole control and access to the corresponding private key.
Public Key (Asymmetric) Cryptographic Algorithm A cryptographic algorithm that uses two related keys, a public key and a private key. The two keys have the property that deriving the private key from the public key is computationally infeasible.
Public Key Infrastructure (PKI) A set of policies, processes, server platforms, software and workstations used for the purpose of administering certificates and public-private key pairs, including the ability to issue, maintain, and revoke public key certificates.

Source: *Glossary of Key Information Security Terms*, NISTIR 7298.

Finally, there is a feeling that key distribution is trivial when using public-key encryption, compared to the rather cumbersome handshaking involved with key distribution centers for symmetric encryption. In fact, some form of protocol is needed, generally involving a central agent, and the procedures involved are not simpler nor any more efficient than those required for symmetric encryption (e.g., see analysis in [NEED78]).

This chapter and the next provide an overview of public-key cryptography. First, we look at its conceptual framework. Interestingly, the concept for this technique was developed and published before it was shown to be practical to adopt it. Next, we examine the RSA algorithm, which is the most important encryption/decryption algorithm that has been shown to be feasible for public-key encryption. Other important public-key cryptographic algorithms are covered in Chapter 10.

Much of the theory of public-key cryptosystems is based on number theory. If one is prepared to accept the results given in this chapter, an understanding of number theory is not strictly necessary. However, to gain a full appreciation of public-key algorithms, some understanding of number theory is required. Chapter 2 provides the necessary background in number theory.

Table 9.1 defines some key terms.

9.1 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. The first problem is that of key distribution, which is examined in some detail in Chapter 14.

As Chapter 14 discusses, 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. As Diffie put it [DIFF88], "what good would it do after all to develop impenetrable cryptosystems, if their users were forced to share their keys with a KDC that could be compromised by either burglary or subpoena?"

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. That is, could a method be devised that would stipulate, to the satisfaction of all parties, that a digital message had been sent by a particular person? This is a somewhat broader requirement than that of authentication, and its characteristics and ramifications are explored in Chapter 13.

Diffie and Hellman achieved an astounding breakthrough in 1976 [DIFF76 a, b] by coming up with a method that addressed both problems and was radically different from all previous approaches to cryptography, going back over four millennia.

In the next subsection, we look at the overall framework for public-key cryptography. Then we examine the requirements for the encryption/decryption algorithm that is at the heart of the scheme.

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 (Figure 9.1a; compare with Figure 3.1).

- **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.

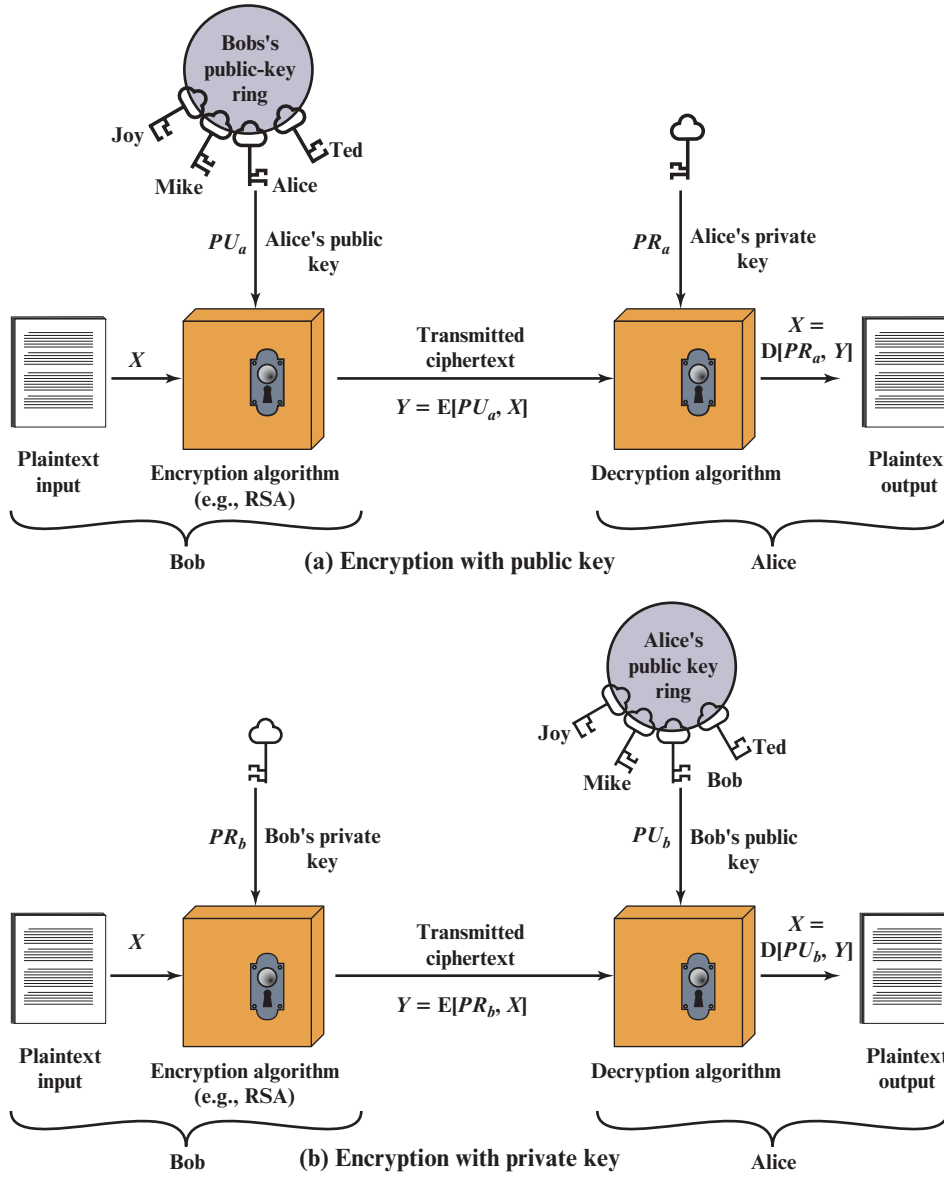

Figure 9.1 Public-Key Cryptography

- **Ciphertext:** This is the encrypted 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 9.2 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**.[1] 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.

Let us take a closer look at the essential elements of a public-key encryption scheme, using Figure 9.2 (compare with Figure 3.2). There is some source A that produces a message in plaintext, $X = [X_1, X_2, \ldots, X_M]$. 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, PU_b, and a private key, PR_b. PR_b is known only to B, whereas PU_b is publicly available and therefore accessible by A.

With the message X and the encryption key PU_b as input, A forms the ciphertext $Y = [Y_1, Y_2, \ldots, Y_N]$:

$$Y = \mathrm{E}(PU_b, X)$$

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

$$X = \mathrm{D}(PR_b, Y)$$

[1]The following notation is used consistently throughout. A secret key is represented by K_m, where m is some modifier; for example, K_a is a secret key owned by user A. A public key is represented by PU_a, for user A, and the corresponding private key is PR_a. Encryption of plaintext X can be performed with a secret key, a public key, or a private key, denoted by $\mathrm{E}(K_a, X)$, $\mathrm{E}(PU_a, X)$, and $\mathrm{E}(PR_a, X)$, respectively. Similarly, decryption of ciphertext Y can be performed with a secret key, a public key, or a private key, denoted by $\mathrm{D}(K_a, Y)$, $\mathrm{D}(PU_a, Y)$, and $\mathrm{D}(PR_a, Y)$, respectively.

Table 9.2 Conventional and Public-Key Encryption

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if the key is kept secret. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and a related algorithm for decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if one of the keys is kept secret. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

An adversary, observing Y and having access to PU_b, but not having access to PR_b or X, must attempt to recover X and/or PR_b. 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 $\hat{X}$. Often, however, the adversary is interested in being able to read future messages as well, in which case an attempt is made to recover PR_b by generating an estimate $\hat{PR}_b$.

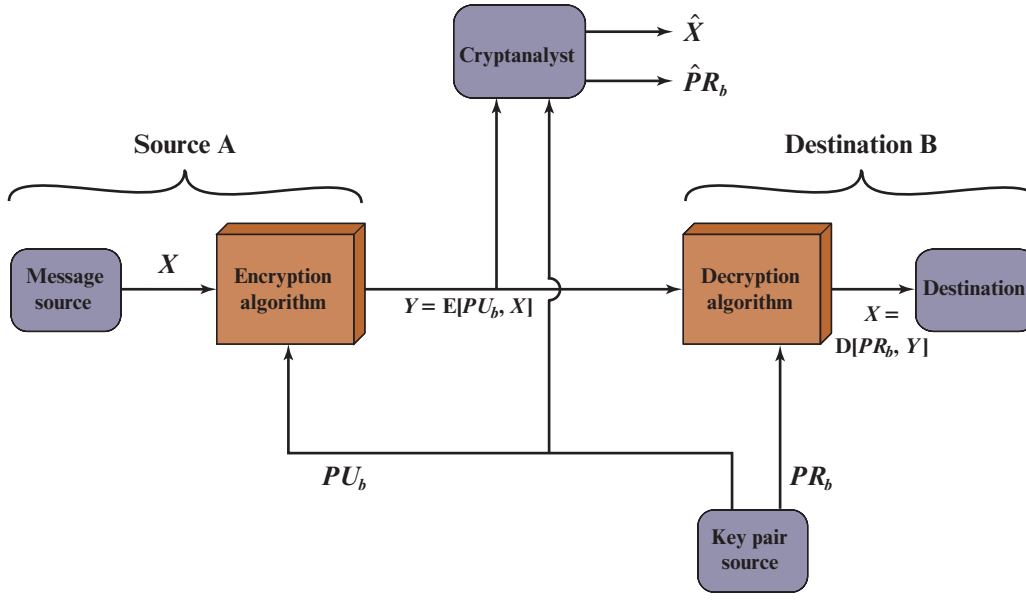

Figure 9.2 Public-Key Cryptosystem: Confidentiality

We mentioned earlier that either of the two related keys can be used for encryption, with the other being used for decryption. This enables a rather different cryptographic scheme to be implemented. Whereas the scheme illustrated in Figure 9.2 provides confidentiality, Figures 9.1b and 9.3 show the use of public-key encryption to provide authentication:

$$Y = \mathrm{E}(PR_a, X)$$
$$X = \mathrm{D}(PU_a, Y)$$

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 sequencing. Chapter 13 examines this technique in detail.

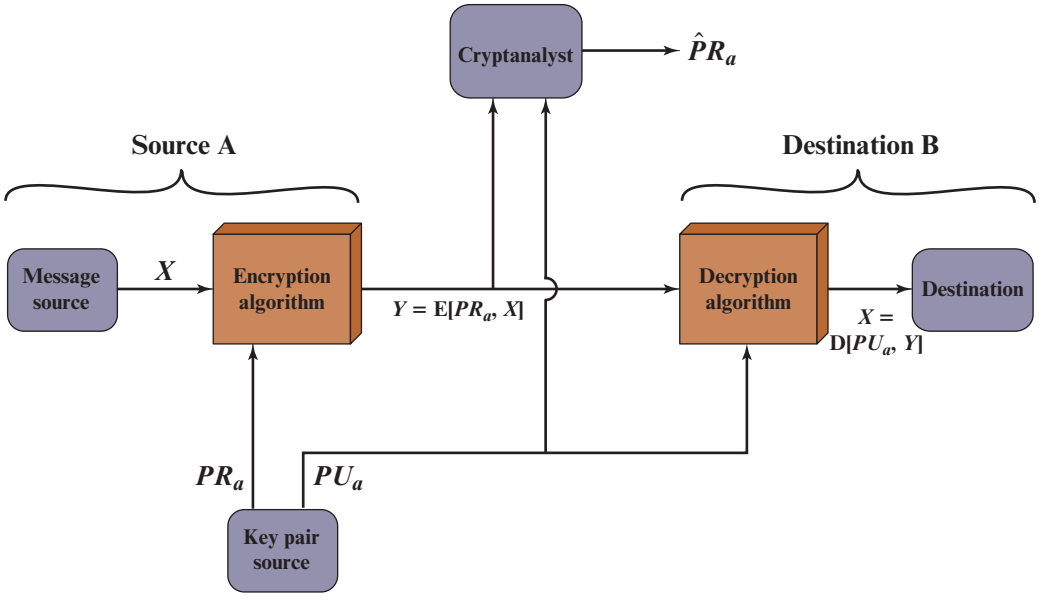

Figure 9.3 Public-Key Cryptosystem: Authentication

It 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):

$$Z = \mathrm{E}(PU_b, \mathrm{E}(PR_a, X))$$
$$X = \mathrm{D}(PU_a, \mathrm{D}(PR_b, Z))$$

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

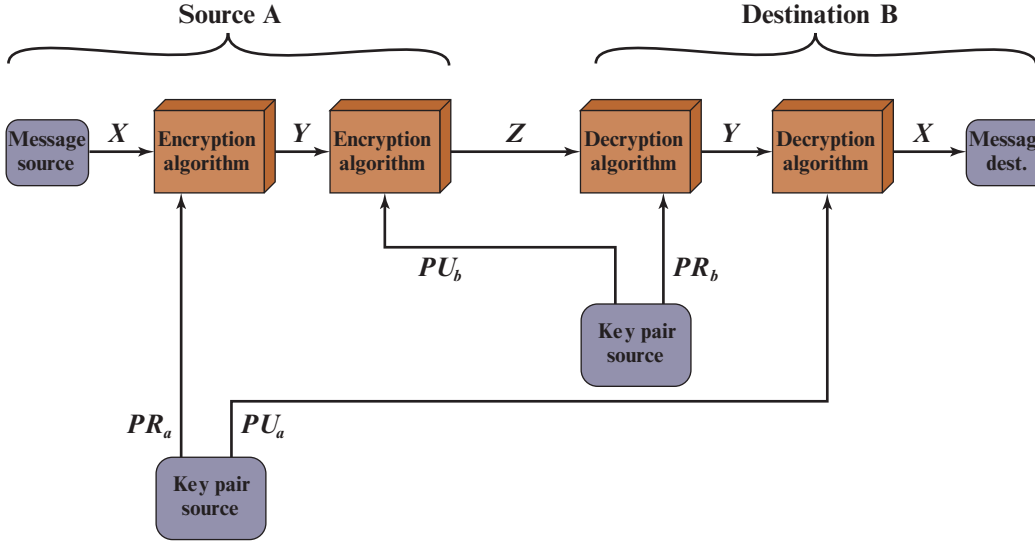

Figure 9.4 Public-Key Cryptosystem: Authentication and Secrecy

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, and the recipient decrypts the message with the recipient's private 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, which is a secret key for symmetric encryption generated for use for a particular transaction (or session) and valid for a short period of time. Several different approaches are possible, involving the private key(s) of one or both parties; this is discussed in Chapter 10.

Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. Table 9.3 indicates the applications supported by the algorithms discussed in this book.

Requirements for Public-Key Cryptography

The cryptosystem illustrated in Figures 9.2 through 9.4 depends on a cryptographic algorithm based on two related keys. Diffie and Hellman postulated this system without demonstrating that such algorithms exist. However, they did lay out the conditions that such algorithms must fulfill [DIFF76b].

1. It is computationally easy for a party B to generate a key pair (public key PU_b, private key PR_b).
2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, M, to generate the corresponding ciphertext:

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

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

$$M = \mathrm{D}(PR_b, C) = \mathrm{D}[PR_b, \mathrm{E}(PU_b, M)]$$

4. It is computationally infeasible for an adversary, knowing the public key, PU_b, to determine the private key, PR_b.

Table 9.3 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

5. It is computationally infeasible for an adversary, knowing the public key, PU_b, and a ciphertext, C, to recover the original message, M.

We can add a sixth requirement that, although useful, is not necessary for all public-key applications:

6. The two keys can be applied in either order:

$$M = \mathrm{D}[PU_b, \mathrm{E}(PR_b, M)] = \mathrm{D}[PR_b, \mathrm{E}(PU_b, M)]$$

These are formidable requirements, as evidenced by the fact that only a few algorithms (RSA, elliptic curve cryptography, Diffie–Hellman, DSS) have received widespread acceptance in the several decades since the concept of public-key cryptography was proposed.

Before elaborating on why the requirements are so formidable, let us first recast them. The requirements boil down to the need for a trap-door one-way function. A **one-way function**[2] is one that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy, whereas the calculation of the inverse is infeasible:

$$\begin{aligned} Y &= \mathrm{f}(X) \quad \text{easy} \\ X &= \mathrm{f}^{-1}(Y) \quad \text{infeasible} \end{aligned}$$

Generally, *easy* is defined to mean a problem that can be solved in polynomial time as a function of input length. Thus, if the length of the input is n bits, then the time to compute the function is proportional to n^a, where a is a fixed constant. Such algorithms are said to belong to the class **P**. The term *infeasible* is a much fuzzier concept. In general, we can say a problem is infeasible if the effort to solve it grows faster than polynomial time as a function of input size. For example, if the length of the input is n bits and the time to compute the function is proportional to 2^n, the problem is considered infeasible. Unfortunately, it is difficult to determine if a particular algorithm exhibits this complexity. Furthermore, traditional notions of computational complexity focus on the worst-case or average-case complexity of an algorithm. These measures are inadequate for cryptography, which requires that it be infeasible to invert a function for virtually all inputs, not for the worst case or even average case. [LAI18] provides an excellent introduction to complexity.

We now turn to the definition of a trap-door one-way function, which is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known. With the additional information the inverse can be calculated in polynomial time. We can summarize as follows: A trap-door one-way function is a family of invertible functions f_k, such that

$$\begin{aligned} Y &= \mathrm{f}_k(X) \quad \text{easy, if } k \text{ and } X \text{ are known} \\ X &= \mathrm{f}_k^{-1}(Y) \quad \text{easy, if } k \text{ and } Y \text{ are known} \\ X &= \mathrm{f}_k^{-1}(Y) \quad \text{infeasible, if } Y \text{ is known but } k \text{ is not known} \end{aligned}$$

[2]Not to be confused with a one-way hash function, which takes an arbitrarily large data field as its argument and maps it to a fixed output. Such functions are used for authentication (see Chapter 11).

Thus, the development of a practical public-key scheme depends on discovery of a suitable trap-door one-way function.

Public-Key Cryptanalysis

As with symmetric encryption, a public-key encryption scheme is vulnerable to a brute-force attack. The countermeasure is the same: Use large keys. However, there is a tradeoff to be considered. Public-key systems depend on the use of some sort of invertible mathematical function. The complexity of calculating these functions may not scale linearly with the number of bits in the key but grow more rapidly than that. Thus, the key size must be large enough to make brute-force attack impractical but small enough for practical encryption and decryption. In practice, the key sizes that have been proposed do make brute-force attack impractical but result in encryption/decryption speeds that are too slow for general-purpose use. Instead, as was mentioned earlier, public-key encryption is currently confined to key management and signature applications.

Another form of attack is to find some way to compute the private key given the public key. To date, it has not been mathematically proven that this form of attack is infeasible for a particular public-key algorithm. Thus, any given algorithm, including the widely used RSA algorithm, is suspect. The history of cryptanalysis shows that a problem that seems insoluble from one perspective can be found to have a solution if looked at in an entirely different way.

Finally, there is a form of attack that is peculiar to public-key systems. This is, in essence, a probable-message attack. Suppose, for example, that a message were to be sent that consisted solely of a 56-bit DES key. An adversary could encrypt all possible 56-bit DES keys using the public key and could discover the encrypted key by matching the transmitted ciphertext. Thus, no matter how large the key size of the public-key scheme, the attack is reduced to a brute-force attack on a 56-bit key. This attack can be thwarted by appending some random bits to such simple messages.

9.2 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 successful responses to the challenge was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978 [RIVE78]. 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 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

RSA 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 $\log_2(n) + 1$; in practice, the block size is i bits, where $2^i < n \leq 2^{i+1}$. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C.

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

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 $PR = \{d, n\}$. 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, and n such that $M^{ed} \bmod n = M$ for all $M < n$.
2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ 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 $\phi(n)$, where $\phi(n)$ is the Euler totient function. It is shown in Chapter 2 that for p, q prime, $\phi(pq) = (p - 1)(q - 1)$. The relationship between e and d can be expressed as

$$ed \bmod \phi(n) = 1 \tag{9.1}$$

This is equivalent to saying

$$ed \equiv 1 \bmod \phi(n)$$
$$d \equiv e^{-1} \bmod \phi(n)$$

That is, e and d are multiplicative inverses mod $\phi(n)$. Note that, according to the rules of modular arithmetic, this is true only if d (and therefore e) is relatively prime to $\phi(n)$. Equivalently, $\gcd(\phi(n), d) = 1$. A proof that Equation (9.1) satisfies the requirement for RSA can be found in the original RSA paper [RIVE78].

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 $\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$	(public, chosen)
$d \equiv e^{-1} \pmod{\phi(n)}$	(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 $M = C^d \bmod n$.

Figure 9.5 summarizes the RSA algorithm. It corresponds to Figure 9.1a: Alice generates a public/private key pair; Bob encrypts using Alice's public key; and Alice decrypts using her private key. An example from [SING99] is shown in Figure 9.6. 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 $\phi(n) = (p - 1)(q - 1) = 16 \times 10 = 160$.
4. Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
5. Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because $23 \times 7 = 161 = (1 \times 160) + 1$; d can be calculated using the extended Euclid's algorithm (Chapter 2).

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 $C = 88^7 \bmod 187$. Exploiting the properties of modular arithmetic, we can do this as follows.

$$\begin{aligned}
88^7 \bmod 187 &= [(88^4 \bmod 187) \times (88^2 \bmod 187) \\
&\quad \times (88^1 \bmod 187)] \bmod 187 \\
88^1 \bmod 187 &= 88 \\
88^2 \bmod 187 &= 7744 \bmod 187 = 77 \\
88^4 \bmod 187 &= 59{,}969{,}536 \bmod 187 = 132 \\
88^7 \bmod 187 &= (88 \times 77 \times 132) \bmod 187 = 894{,}432 \bmod 187 = 11
\end{aligned}$$

For decryption, we calculate $M = 11^{23} \bmod 187$:

$$\begin{aligned}
11^{23} \bmod 187 &= [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \\
&\quad \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187 \\
11^1 \bmod 187 &= 11 \\
11^2 \bmod 187 &= 121 \\
11^4 \bmod 187 &= 14{,}641 \bmod 187 = 55 \\
11^8 \bmod 187 &= 214{,}358{,}881 \bmod 187 = 33 \\
11^{23} \bmod 187 &= (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 \\
&= 79{,}720{,}245 \bmod 187 = 88
\end{aligned}$$

We now look at an example from [HELL79], which shows the use of RSA to process multiple blocks of data. In this simple example, the plaintext is an alphanumeric string. Each plaintext symbol is assigned a unique code of two decimal

Key Generation by Alice	
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 by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Private Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Figure 9.5 The RSA Algorithm

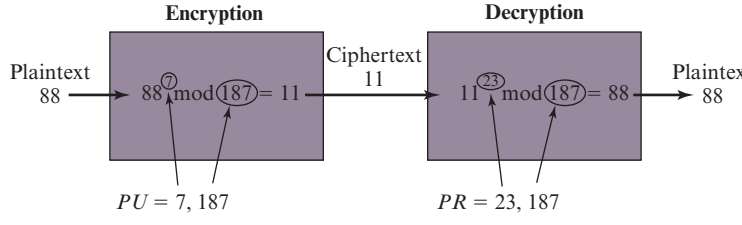

Figure 9.6 Example of RSA Algorithm

digits (e.g., a = 00, A = 26).[3] A plaintext block consists of four decimal digits, or two alphanumeric characters. Figure 9.7a illustrates the sequence of events for the encryption of multiple blocks, and Figure 9.7b gives a specific example. The circled numbers indicate the order in which operations are performed.

Computational Aspects

We now turn to the issue of the complexity of the computation required to use RSA. There are actually two issues to consider: encryption/decryption and key generation. Let us look first at the process of encryption and decryption and then consider key generation.

[3]The complete mapping of alphanumeric characters to decimal digits is at box.com/Crypto8e in the document `RSAexample.pdf`.

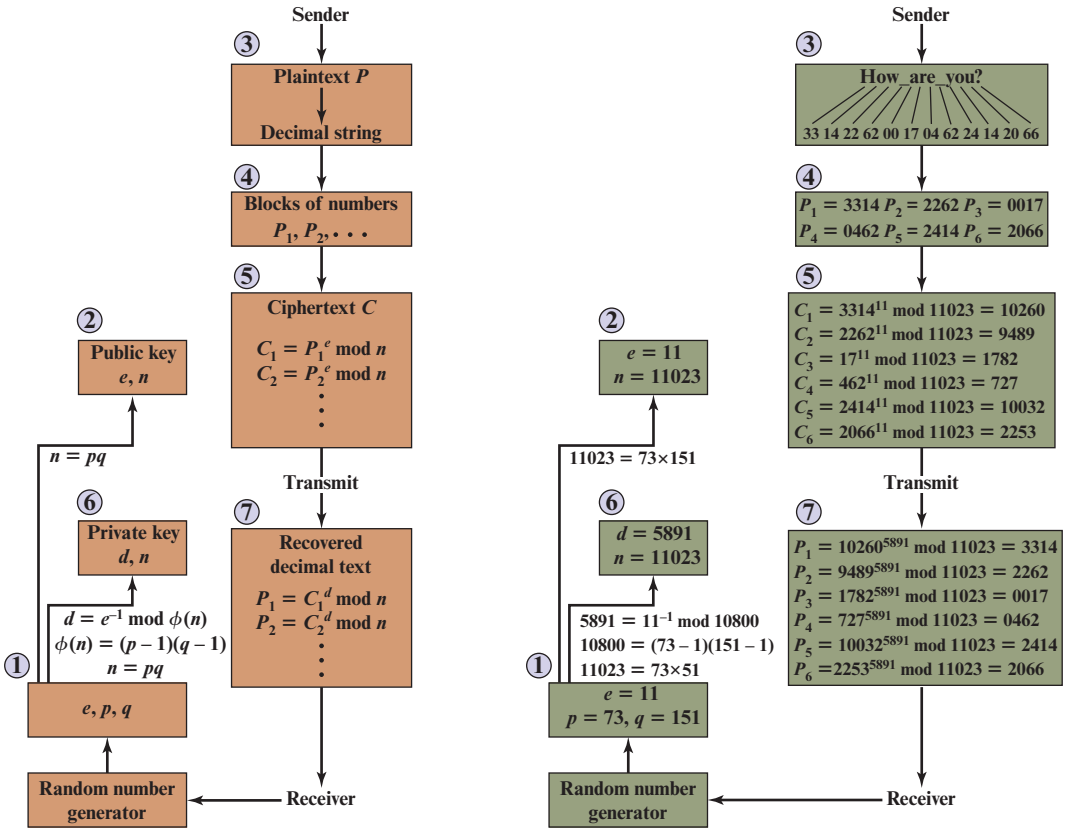

Figure 9.7 RSA Processing of Multiple Blocks

Exponentiation in Modular Arithmetic Both encryption and decryption in RSA involve raising an integer to an integer power, mod n. If the exponentiation is done over the integers and then reduced modulo n, the intermediate values would be gargantuan. Fortunately, as the preceding example shows, we can make use of a property of modular arithmetic:

$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

Thus, we can reduce intermediate results modulo n. This makes the calculation practical.

Another consideration is the efficiency of exponentiation, because with RSA, we are dealing with potentially large exponents. To see how efficiency might be increased, consider that we wish to compute x^{16}. A straightforward approach requires 15 multiplications:

$$x^{16} = x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x$$

However, we can achieve the same final result with only four multiplications if we repeatedly take the square of each partial result, successively forming (x^2, x^4, x^8, x^{16}). As another example, suppose we wish to calculate $x^{11} \bmod n$ for some integers x and n. Observe that $x^{11} = x^{1+2+8} = (x)(x^2)(x^8)$. In this case, we compute $x \bmod n$, $x^2 \bmod n$, $x^4 \bmod n$, and $x^8 \bmod n$ and then calculate $[(x \bmod n) \times (x^2 \bmod n) \times (x^8 \bmod n)] \bmod n$.

More generally, suppose we wish to find the value $a^b \bmod n$ with a, b, and m positive integers. If we express b as a binary number $b_k b_{k-1} \ldots b_0$, then we have

$$b = \sum_{b_i \neq 0} 2^i$$

Therefore,

$$a^b = a^{\left(\sum_{b_i \neq 0} 2^i\right)} = \prod_{b_i \neq 0} a^{(2^i)}$$

$$a^b \bmod n = \left[\prod_{b_i \neq 0} a^{(2^i)}\right] \bmod n = \left(\prod_{b_i \neq 0} \left[a^{(2^i)} \bmod n\right]\right) \bmod n$$

We can therefore develop the algorithm[4] for computing $a^b \bmod n$, shown in Figure 9.8. Table 9.4 shows an example of the execution of this algorithm. Note that the variable c is not needed; it is included for explanatory purposes. The final value of c is the value of the exponent.

Efficient Operation Using the Public Key To speed up the operation of the RSA algorithm using the public key, a specific choice of e is usually made. The most common choice is 65537 $(2^{16} + 1)$; two other popular choices are 3 and 17. Each of these choices has only two 1 bits, so the number of multiplications required to perform exponentiation is minimized.

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

Note: The integer b is expressed as a binary number $b_k b_{k-1} \ldots b_0$.

Figure 9.8 Algorithm for Computing $a^b \bmod n$

[4]The algorithm has a long history; this particular pseudocode expression is from [CORM09].

Table 9.4 Result of the Fast Modular Exponentiation Algorithm for $a^b \bmod n$, where $a = 7$, $b = 560 = 1000110000$, and $n = 561$

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
f	7	49	157	526	160	241	298	166	67	1

However, with a very small public key, such as $e = 3$, RSA becomes vulnerable to a simple attack. Suppose we have three different RSA users who all use the value $e = 3$ but have unique values of n, namely (n_1, n_2, n_3). If user A sends the same encrypted message M to all three users, then the three ciphertexts are $C_1 = M^3 \bmod n_1$, $C_2 = M^3 \bmod n_2$, and $C_3 = M^3 \bmod n_3$. It is likely that n_1, n_2, and n_3 are pairwise relatively prime. Therefore, one can use the Chinese remainder theorem (CRT) to compute $M^3 \bmod (n_1 n_2 n_3)$. By the rules of the RSA algorithm, M is less than each of the n_i; therefore $M^3 < n_1 n_2 n_3$. Accordingly, the attacker need only compute the cube root of M^3. This attack can be countered by adding a unique pseudorandom bit string as padding to each instance of M to be encrypted. This approach is discussed subsequently.

The reader may have noted that the definition of the RSA algorithm (Figure 9.5) requires that during key generation the user selects a value of e that is relatively prime to $\phi(n)$. Thus, if a value of e is selected first and the primes p and q are generated, it may turn out that $\gcd(\phi(n), e) \neq 1$. In that case, the user must reject the p, q values and generate a new p, q pair.

Efficient Operation Using the Private Key We cannot similarly choose a small constant value of d for efficient operation. A small value of d is vulnerable to a brute-force attack and to other forms of cryptanalysis [WIEN90]. However, there is a way to speed up computation using the CRT. We wish to compute the value $M = C^d \bmod n$. Let us define the following intermediate results:

$$V_p = C^d \bmod p \quad V_q = C^d \bmod q$$

Following the CRT using Equation (8.8), define the quantities

$$X_p = q \times (q^{-1} \bmod p) \quad X_q = p \times (p^{-1} \bmod q)$$

The CRT then shows, using Equation (8.9), that

$$M = (V_p X_p + V_q X_q) \bmod n$$

Furthermore, we can simplify the calculation of V_p and V_q using Fermat's theorem, which states that $a^{p-1} \equiv 1 \pmod p$ if p and a are relatively prime. Some thought should convince you that the following are valid.

$$V_p = C^d \bmod p = C^{d \bmod (p-1)} \bmod p \quad V_q = C^d \bmod q = C^{d \bmod (q-1)} \bmod q$$

The quantities $d \bmod (p - 1)$ and $d \bmod (q - 1)$ can be precalculated. The end result is that the calculation is approximately four times as fast as evaluating $M = C^d \bmod n$ directly [BONE02].

Key Generation Before the application of the public-key cryptosystem, each participant must generate a pair of keys. This involves the following tasks.

- Determining two prime numbers, p and q.
- Selecting either e or d and calculating the other.

First, consider the selection of p and q. Because the value of $n = pq$ will be known to any potential adversary, in order to prevent the discovery of p and q by exhaustive methods, these primes must be chosen from a sufficiently large set (i.e., p and q must be large numbers). On the other hand, the method used for finding large primes must be reasonably efficient.

At present, there are no useful techniques that yield arbitrarily large primes, so some other means of tackling the problem is needed. The procedure that is generally used is to pick at random an odd number of the desired order of magnitude and test whether that number is prime. If not, pick successive random numbers until one is found that tests prime.

A variety of tests for primality have been developed (e.g., see [KNUT98] for a description of a number of such tests). Almost invariably, the tests are probabilistic. That is, the test will merely determine that a given integer is *probably* prime. Despite this lack of certainty, these tests can be run in such a way as to make the probability as close to 1.0 as desired. As an example, one of the more efficient and popular algorithms, the Miller–Rabin algorithm, is described in Chapter 2. With this algorithm and most such algorithms, the procedure for testing whether a given integer n is prime is to perform some calculation that involves n and a randomly chosen integer a. If n "fails" the test, then n is not prime. If n "passes" the test, then n may be prime or nonprime. If n passes many such tests with many different randomly chosen values for a, then we can have high confidence that n is, in fact, prime.

In summary, the procedure for picking a prime number is as follows.

1. Pick an odd integer n at random (e.g., using a pseudorandom number generator).
2. Pick an integer $a < n$ at random.
3. Perform the probabilistic primality test, such as Miller–Rabin, with a as a parameter. If n fails the test, reject the value n and go to step 1.
4. If n has passed a sufficient number of tests, accept n; otherwise, go to step 2.

This is a somewhat tedious procedure. However, remember that this process is performed relatively infrequently: only when a new pair (PU, PR) is needed.

It is worth noting how many numbers are likely to be rejected before a prime number is found. A result from number theory, known as the prime number theorem, states that the primes near N are spaced on the average one every

ln (N) integers. Thus, on average, one would have to test on the order of ln(N) integers before a prime is found. Actually, because all even integers can be immediately rejected, the correct figure is ln(N)/2. For example, if a prime on the order of magnitude of 2^{200} were sought, then about $\ln(2^{200})/2 = 70$ trials would be needed to find a prime.

Having determined prime numbers p and q, the process of key generation is completed by selecting a value of e and calculating d or, alternatively, selecting a value of d and calculating e. Assuming the former, then we need to select an e such that $\gcd(\phi(n), e) = 1$ and then calculate $d \equiv e^{-1} \pmod{\phi(n)}$. Fortunately, there is a single algorithm that will, at the same time, calculate the greatest common divisor of two integers and, if the gcd is 1, determine the inverse of one of the integers modulo the other. The algorithm, referred to as the extended Euclid's algorithm, is explained in Chapter 2. Thus, the procedure is to generate a series of random numbers, testing each against $\phi(n)$ until a number relatively prime to $\phi(n)$ is found. Again, we can ask the question: How many random numbers must we test to find a usable number, that is, a number relatively prime to $\phi(n)$? It can be shown easily that the probability that two random numbers are relatively prime is about 0.6; thus, very few tests would be needed to find a suitable integer (see Problem 2.18).

The Security of RSA

Five possible approaches to attacking the RSA algorithm 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.
- **Hardware fault-based attack:** This involves inducing hardware faults in the processor that is generating digital signatures.
- **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, to 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.

In this subsection, we provide an overview of mathematical and timing attacks.

The Factoring Problem We can identify three approaches to attacking RSA mathematically.

1. Factor n into its two prime factors. This enables calculation of $\phi(n) = (p - 1) \times (q - 1)$, which in turn enables determination of $d \equiv e^{-1} \pmod{\phi(n)}$.
2. Determine $\phi(n)$ directly, without first determining p and q. Again, this enables determination of $d \equiv e^{-1} \pmod{\phi(n)}$.
3. Determine d directly, without first determining $\phi(n)$.

Most discussions of the cryptanalysis of RSA have focused on the task of factoring n into its two prime factors. Determining $\phi(n)$ given n is equivalent to factoring n [RIBE96]. With presently known algorithms, determining d given e and n appears to be at least as time-consuming as the factoring problem [KALI95]. Hence, we can use factoring performance as a benchmark against which to evaluate the security of RSA.

For a large n with large prime factors, factoring is a hard problem, but it is not as hard as it used to be. A striking illustration of this is the following. In 1977, the three inventors of RSA dared *Scientific American* readers to decode a cipher they printed in Martin Gardner's "Mathematical Games" column [GARD77]. They offered a $100 reward for the return of a plaintext sentence, an event they predicted might not occur for some 40 quadrillion years. In April of 1994, a group working over the Internet claimed the prize after only eight months of work [LEUT94]. This challenge used a public key size (length of n) of 129 decimal digits, or around 428 bits. In the meantime, just as they had done for DES, RSA Laboratories had issued challenges for the RSA cipher with key sizes of 100, 110, 120, and so on, digits. The latest challenge to be met is the RSA-768 challenge with a key length of 232 decimal digits, or 768 bits.

A striking fact about the factoring of the successive challenges concerns the method used. Until the mid-1990s, factoring attacks were made using an approach known as the quadratic sieve. The attack on RSA-130 used a newer algorithm, the generalized number field sieve (GNFS), and was able to factor a larger number than RSA-129 at only 20% of the computing effort.

The threat to larger key sizes is twofold: the continuing increase in computing power and the continuing refinement of factoring algorithms. We have seen that the move to a different algorithm resulted in a tremendous speedup. We can expect further refinements in the GNFS, and the use of an even better algorithm is also a possibility. In fact, a related algorithm, the special number field sieve (SNFS), can factor numbers with a specialized form considerably faster than the generalized number field sieve. It is reasonable to expect a breakthrough that would enable a general factoring performance in about the same time as SNFS, or even better [ODLY95]. Thus, we need to be careful in choosing a key size for RSA. The team that produced the 768-bit factorization [KLEI10] observed that factoring a 1024-bit RSA modulus would be about a thousand times harder than factoring a 768-bit modulus, and a 768-bit RSA modulus is several thousands times harder to factor than a 512-bit one. Based on the amount of time between the 512-bit and 768-bit factorization successes, the team felt it to be reasonable to expect that the 1024-bit RSA moduli could be factored well within the next decade by a similar academic effort. Thus, they recommended phasing out usage of 1024-bit RSA within the next few years (from 2010).

A number of government agencies have issued recommendations for RSA key size:

- NIST SP 800-131A (*Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*, November 2015) recommends a key length of 2048 bits or longer.
- The European Union Agency for Network and Information Security, in *Algorithms, Key Size and Parameters Report – 2014* recommends a key length of 3072 bits or longer for all new development.

- The government of Canada's Communications Security Establishment, in *Cryptographic Algorithms for UNCLASSIFIED, PROTECTED A, and PROTECTED B Information* (August 2016) recommends a length of at least 2048 bits, extended to at least 3072 bits by 2030.

In addition to specifying the size of n, a number of other constraints have been suggested by researchers. To avoid values of n that may be factored more easily, the algorithm's inventors suggest the following constraints on p and q.

1. p and q should differ in length by only a few digits. Thus, for a 1024-bit key (309 decimal digits), both p and q should be on the order of magnitude of 10^{75} to 10^{100}.
2. Both $(p - 1)$ and $(q - 1)$ should contain a large prime factor.
3. $\gcd(p - 1, q - 1)$ should be small.

In addition, it has been demonstrated that if $e < n$ and $d < n^{1/4}$, then d can be easily determined [WIEN90].

Timing Attacks If one needed yet another lesson about how difficult it is to assess the security of a cryptographic algorithm, the appearance of timing attacks provides a stunning one. Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages [KOCH96, KALI96b]. Timing attacks are applicable not just to RSA, but to other public-key cryptography systems. This attack is alarming for two reasons: It comes from a completely unexpected direction, and it is a ciphertext-only attack.

A **timing attack** is somewhat analogous to a burglar guessing the combination of a safe by observing how long it takes for someone to turn the dial from number to number. We can explain the attack using the modular exponentiation algorithm of Figure 9.8, but the attack can be adapted to work with any implementation that does not run in fixed time. In this algorithm, modular exponentiation is accomplished bit by bit, with one modular multiplication performed at each iteration and an additional modular multiplication performed for each 1 bit.

As Kocher points out in his paper, the attack is simplest to understand in an extreme case. Suppose the target system uses a modular multiplication function that is very fast in almost all cases but in a few cases takes much more time than an entire average modular exponentiation. The attack proceeds bit-by-bit starting with the leftmost bit, b_k. Suppose that the first j bits are known (to obtain the entire exponent, start with $j = 0$ and repeat the attack until the entire exponent is known). For a given ciphertext, the attacker can complete the first j iterations of the **for** loop. The operation of the subsequent step depends on the unknown exponent bit. If the bit is set, $d \leftarrow (d \times a) \bmod n$ will be executed. For a few values of a and d, the modular multiplication will be extremely slow, and the attacker knows which these are. Therefore, if the observed time to execute the decryption algorithm is always slow when this particular iteration is slow with a 1 bit, then this bit is assumed to be 1. If a number of observed execution times for the entire algorithm are fast, then this bit is assumed to be 0.

In practice, modular exponentiation implementations do not have such extreme timing variations, in which the execution time of a single iteration can exceed the mean execution time of the entire algorithm. Nevertheless, there is enough variation to make this attack practical. For details, see [KOCH96].

Although the timing attack is a serious threat, there are simple countermeasures that can be used, including the following.

- **Constant exponentiation time:** Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.
- **Random delay:** Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. Kocher points out that if defenders don't add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays.
- **Blinding:** Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack.

RSA Data Security incorporates a blinding feature into some of its products. The private-key operation $M = C^d \bmod n$ is implemented as follows.

1. Generate a secret random number r between 0 and $n - 1$.
2. Compute $C' = C(r^e) \bmod n$, where e is the public exponent.
3. Compute $M' = (C')^d \bmod n$ with the ordinary RSA implementation.
4. Compute $M = M'r^{-1} \bmod n$. In this equation, r^{-1} is the multiplicative inverse of $r \bmod n$; see Chapter 2 for a discussion of this concept. It can be demonstrated that this is the correct result by observing that $r^{ed} \bmod n = r \bmod n$.

RSA Data Security reports a 2 to 10% performance penalty for blinding.

Fault-Based Attack Still another unorthodox approach to attacking RSA is reported in [PELL10]. The approach is an attack on a processor that is generating RSA digital signatures. The attack induces faults in the signature computation by reducing the power to the processor. The faults cause the software to produce invalid signatures, which can then be analyzed by the attacker to recover the private key. The authors show how such an analysis can be done and then demonstrate it by extracting a 1024-bit private RSA key in approximately 100 hours, using a commercially available microprocessor.

The attack algorithm involves inducing single-bit errors and observing the results. The details are provided in [PELL10], which also references other proposed hardware fault-based attacks against RSA.

This attack, while worthy of consideration, does not appear to be a serious threat to RSA. It requires that the attacker have physical access to the target machine and that the attacker is able to directly control the input power to the

processor. Controlling the input power would for most hardware require more than simply controlling the AC power, but would also involve the power supply control hardware on the chip.

Chosen Ciphertext Attack and Optimal Asymmetric Encryption Padding The basic RSA algorithm is vulnerable to a chosen ciphertext attack (CCA). CCA is defined as an attack in which the adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key. Thus, the adversary could select a plaintext, encrypt it with the target's public key, and then be able to get the plaintext back by having it decrypted with the private key. Clearly, this provides the adversary with no new information. Instead, the adversary exploits properties of RSA and selects blocks of data that, when processed using the target's private key, yield information needed for cryptanalysis.

A simple example of a CCA against RSA takes advantage of the following property of RSA:

$$\mathrm{E}(PU, M_1) \times \mathrm{E}(PU, M_2) = \mathrm{E}(PU, [M_1 \times M_2]) \tag{9.2}$$

We can decrypt $C = M^e \bmod n$ using a CCA as follows.

1. Compute $X = (C \times 2^e) \bmod n$.
2. Submit X as a chosen ciphertext and receive back $Y = X^d \bmod n$.

But now note that

$$\begin{aligned} X &= (C \bmod n) \times (2^e \bmod n) \\ &= (M^e \bmod n) \times (2^e \bmod n) \\ &= (2M)^e \bmod n \end{aligned}$$

Therefore, $Y = (2M) \bmod n$. From this, we can deduce M. To overcome this simple attack, practical RSA-based cryptosystems randomly pad the plaintext prior to encryption. This randomizes the ciphertext so that Equation (9.2) no longer holds. However, more sophisticated CCAs are possible, and a simple padding with a random value has been shown to be insufficient to provide the desired security. To counter such attacks, RSA Security Inc., a leading RSA vendor and former holder of the RSA patent, recommends modifying the plaintext using a procedure known as **optimal asymmetric encryption padding (OAEP)**. A full discussion of the threats and OAEP are beyond our scope; see [POIN02] for an introduction and [BELL94a] for a thorough analysis. Here, we simply summarize the OAEP procedure.

Figure 9.9 depicts OAEP encryption. As a first step, the message M to be encrypted is padded. A set of optional parameters, P, is passed through a hash function, H.[5] The output is then padded with zeros to get the desired length in the overall data block (DB). Next, a random seed is generated and passed through another hash function, called the mask generating function (MGF). The resulting hash value is bit-by-bit XORed with DB to produce a maskedDB. The maskedDB is in turn passed through the MGF to form a hash that is XORed with the seed to produce

[5]A hash function maps a variable-length data block or message into a fixed-length value called a hash code. Hash functions are discussed in depth in Chapter 11.

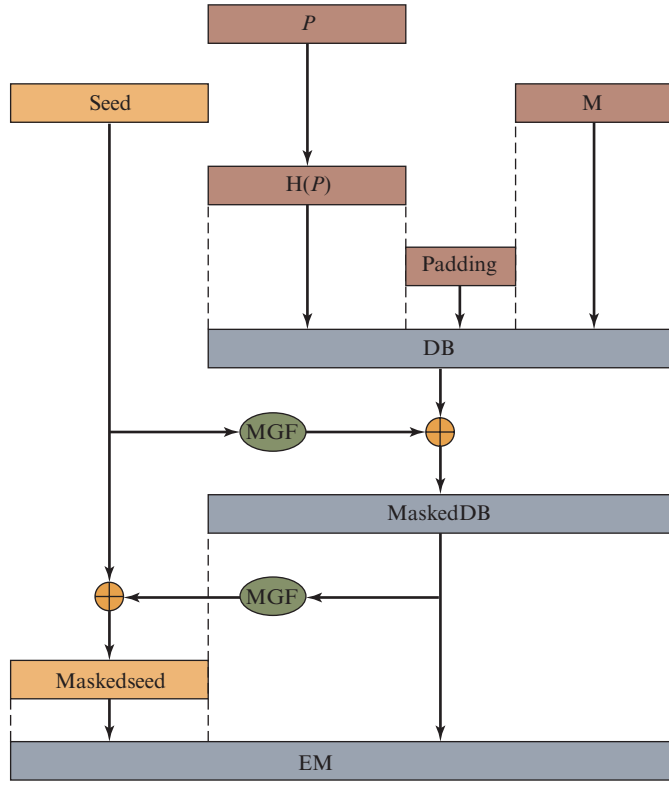

Figure 9.9 Encryption Using Optimal Asymmetric Encryption Padding (OAEP)

the maskedseed. The concatenation of the maskedseed and the maskedDB forms the encoded message EM. Note that the EM includes the padded message, masked by the seed, and the seed, masked by the maskedDB. The EM is then encrypted using RSA.

9.3 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

digital signature	optimal asymmetric encryption padding (OAEP)	public-key encryption
key exchange	private key	RSA
one-way function	public key	secret key
		timing attack

Review Questions

9.1 What is a public key certificate?
9.2 What are the roles of the public and private key?
9.3 What are three broad categories of applications of public-key cryptosystems?
9.4 What requirements must a public-key cryptosystems fulfill to be a secure algorithm?
9.5 How can a probable-message attack be used for public-key cryptanalysis?
9.6 List the different approaches to attack the RSA algorithm.
9.7 Describe the countermeasures to be used against the timing attack.

Problems

9.1 Prior to the discovery of any specific public-key schemes, such as RSA, an existence proof was developed whose purpose was to demonstrate that public-key encryption is possible in theory. Consider the functions $f_1(x_1) = z_1$; $f_2(x_2, y_2) = z_2$; $f_3(x_3, y_3) = z_3$, where all values are integers with $1 \leq x_i, y_i, z_i \leq N$. Function f_1 can be represented by a vector M1 of length N, in which the kth entry is the value of $f_1(k)$. Similarly, f_2 and f_3 can be represented by $N \times N$ matrices M2 and M3. The intent is to represent the encryption/decryption process by table lookups for tables with very large values of N. Such tables would be impractically huge but could be constructed in principle. The scheme works as follows: Construct M1 with a random permutation of all integers between 1 and N; that is, each integer appears exactly once in M1. Construct M2 so that each row contains a random permutation of the first N integers. Finally, fill in M3 to satisfy the following condition:

$$f_3(f_2(f_1(k), p), k) = p \qquad \text{for all } k, p \text{ with } 1 \leq k, p \leq N$$

To summarize,

1. M1 takes an input k and produces an output x.
2. M2 takes inputs x and p giving output z.
3. M3 takes inputs z and k and produces p.

The three tables, once constructed, are made public.

a. It should be clear that it is possible to construct M3 to satisfy the preceding condition. As an example, fill in M3 for the following simple case:

M1 =

4
3
2
5
1

M2 =

3	5	2	4	1
4	2	5	3	1
5	4	3	1	2
1	3	2	5	4
2	1	4	3	5

M3 =

Convention: The ith element of M1 corresponds to $k = i$. The ith row of M2 corresponds to $x = i$; the jth column of M2 corresponds to $p = j$. The ith row of M3 corresponds to $z = i$; the jth column of M3 corresponds to $k = j$.

b. Describe the use of this set of tables to perform encryption and decryption between two users.

c. Argue that this is a secure scheme.

9.2 Perform encryption and decryption using the RSA algorithm, as in Figure 9.5, for the following:

a. $p = 3; q = 7, e = 5; M = 10$
b. $p = 5; q = 13, e = 5; M = 8$
c. $p = 7; q = 17, e = 11; M = 11$

d. $p = 7; q = 13, e = 11; M = 2$
e. $p = 17; q = 23, e = 9; M = 7$

Hint: Decryption is not as hard as you think; use some finesse.

9.3 In a public-key system using RSA, you intercept the ciphertext $C = 20$ sent to a user whose public key is $e = 13, n = 77$. What is the plaintext M?

9.4 In an RSA system, the public key of a given user is $e = 65, n = 2881$. What is the private key of this user? *Hint:* First use trial and error to determine p and q; then use the extended Euclidean algorithm to find the multiplicative inverse of 31 modulo $\phi(n)$.

9.5 In using the RSA algorithm, if a small number of repeated encodings give back the plaintext, what is the likely cause?

9.6 Public-key cryptography has its norms and requirements that make cryptanalysis relatively simple to understand and explain. In a scenario where party A sends a message to party B using public-key cryptography, how difficult will it be for party B to decipher the message? What is the formula to decrypt the message?

9.7 In RSA, the algorithm encrypts a message and transforms it into a ciphertext equal to the message itself with exponentiation modulo n or $C = M^e \bmod n$. In this case, what happens if the receiver has the value of n in advance?

9.8 Suppose Bob uses the RSA cryptosystem with a very large modulus n for which the factorization cannot be found in a reasonable amount of time. Suppose Alice sends a message to Bob by representing each alphabetic character as an integer between 0 and 25 ($A \rightarrow 0, \ldots, Z \rightarrow 25$) and then encrypting each number separately using RSA with large e and large n. Is this method secure? If not, describe the most efficient attack against this encryption method.

9.9 Using a spreadsheet (such as Excel) or a calculator, perform the operations described below. Document results of all intermediate modular multiplications. Determine a number of modular multiplications per each major transformation (such as encryption, decryption, primality testing, etc.).

a. Test all odd numbers in the range from 215 to 223 for primality using the Miller–Rabin test with base 2.
b. Encrypt the message block $M = 2$ using RSA with the following parameters: $e = 23$ and $n = 233 \times 241$.
c. Compute a private key (d, p, q) corresponding to the public key (e, n) given above.
d. Perform the decryption of the obtained ciphertext
  1. without using the Chinese Remainder Theorem, and
  2. using the Chinese Remainder Theorem.

9.10 The security of the RSA algorithm has been impacted due to advances in technology and the available choices of values. This can make it vulnerable to brute force attacks and timing attacks. In asymmetric algorithms, public keys are made public and private keys should be kept private. Suppose a very small public parameter, like $e = 3$, is generated to encrypt a secret message, would it affect the security of the algorithm at all?

9.11 "I want to tell you, Holmes," Dr. Watson's voice was enthusiastic, "that your recent activities in network security have increased my interest in cryptography. And just yesterday I found a way to make one-time pad encryption practical."

"Oh, really?" Holmes' face lost its sleepy look.

"Yes, Holmes. The idea is quite simple. For a given one-way function F, I generate a long pseudorandom sequence of elements by applying F to some standard sequence of arguments. The cryptanalyst is assumed to know F and the general nature of the sequence, which may be as simple as S, S + 1, S + 2, . . . , but not secret S. And due to the one-way nature of F, no one is able to extract S given F(S + i) for some i, thus even if he somehow obtains a certain segment of the sequence, he will not be able to determine the rest."

"I am afraid, Watson, that your proposal isn't without flaws and at least it needs some additional conditions to be satisfied by F. Let's consider, for instance, the RSA encryption function, that is $F(M) = M^K \bmod N$, K is secret. This function is believed to be one-way, but I wouldn't recommend its use, for example, on the sequence $M = 2, 3, 4, 5, 6, \ldots$"

"But why, Holmes?" Dr. Watson apparently didn't understand. "Why do you think that the resulting sequence $2^K \bmod N, 3^K \bmod N, 4^K \bmod N, \ldots$ is not appropriate for one-time pad encryption if K is kept secret?"

"Because it is—at least partially—predictable, dear Watson, even if K is kept secret. You have said that the cryptanalyst is assumed to know F and the general nature of the sequence. Now let's assume that he will obtain somehow a short segment of the output sequence. In crypto circles, this assumption is generally considered to be a viable one. And for this output sequence, knowledge of just the first two elements will allow him to predict quite a lot of the next elements of the sequence, even if not all of them, thus this sequence can't be considered to be cryptographically strong. And with the knowledge of a longer segment he could predict even more of the next elements of the sequence. Look, knowing the general nature of the sequence and its first two elements $2^K \bmod N$ and $3^K \bmod N$, you can easily compute its following elements."

Show how this can be done.

9.12 Show how RSA can be represented by matrices M1, M2, and M3 of Problem 9.1.

9.13 To understand the initial steps of RSA calculations for generating the public key, consider the following scheme to determine $PU = \{e,n\}$. Suppose you have two prime numbers p and q, where $p = 20$ and $q = 14$.

a. Calculate $n = pq$
b. Calculate $\phi(n) = (p - 1)(q - 1)$
c. Select a value of e such that it is relatively prime to $\phi(n)$.

What needs to be determined to complete the key generation process? Show your calculations based on your answers to parts **a** to **c**.

9.14 Consider the following scheme by which B encrypts a message for A.

1. A chooses two large primes P and Q that are also relatively prime to $(P - 1)$ and $(Q - 1)$.
2. A publishes $N = PQ$ as its public key.
3. A calculates P' and Q' such that $PP' \equiv 1 \pmod{Q - 1}$ and $QQ' \equiv 1 \pmod{P - 1}$.
4. B encrypts message M as $C = M^N \bmod N$.
5. A finds M by solving $M \equiv C^{P'} \pmod{Q}$ and $M \equiv C^{Q'} \pmod{P}$.
   a. Explain how this scheme works.
   b. How does it differ from RSA?
   c. Is there any particular advantage to RSA compared to this scheme?
   d. Show how this scheme can be represented by matrices M1, M2, and M3 of Problem 9.1.

9.15 "This is a very interesting case, Watson," Holmes said. "The young man loves a girl, and she loves him too. However, her father is a strange fellow who insists that his would-be son-in-law must design a simple and secure protocol for an appropriate public-key cryptosystem he could use in his company's computer network. The young man came up with the following protocol for communication between two parties. For example, user A wishing to send message M to user B: (messages exchanged are in the format sender's name, text, receiver's name)"

1. A sends B the following block: $(A, E(PU_b, [M, A]), B)$.
2. B acknowledges receipt by sending to A the following block: $(B, E(PU_a, [M, B]), A)$.

"You can see that the protocol is really simple. But the girl's father claims that the young man has not satisfied his call for a simple protocol, because the proposal contains a certain redundancy and can be further simplified to the following:"