Chapter Six
Public-Key Cryptography and RSA

6.1. 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.

As we have seen, 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,

Diffie and Hellman first publicly introduced the concepts of public-key cryptography in 1976. However, this is not the true beginning. Admiral Bobby Inman, while director of the National Security Agency (NSA), claimed that public-key cryptography had been discovered at NSA in the mid-1960s [SIMM93]. The first documented introduction of these concepts came in 1970, from the Communications-Electronics Security Group, Britain's counterpart to NSA, in a classified report by James Ellis [ELLI70]. Ellis referred to the technique as nonsecret encryption and describes the discovery in [ELLI99].

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.

6.1.1. 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 6.1a; compare with Figure 2.1):

- 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.
- 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.

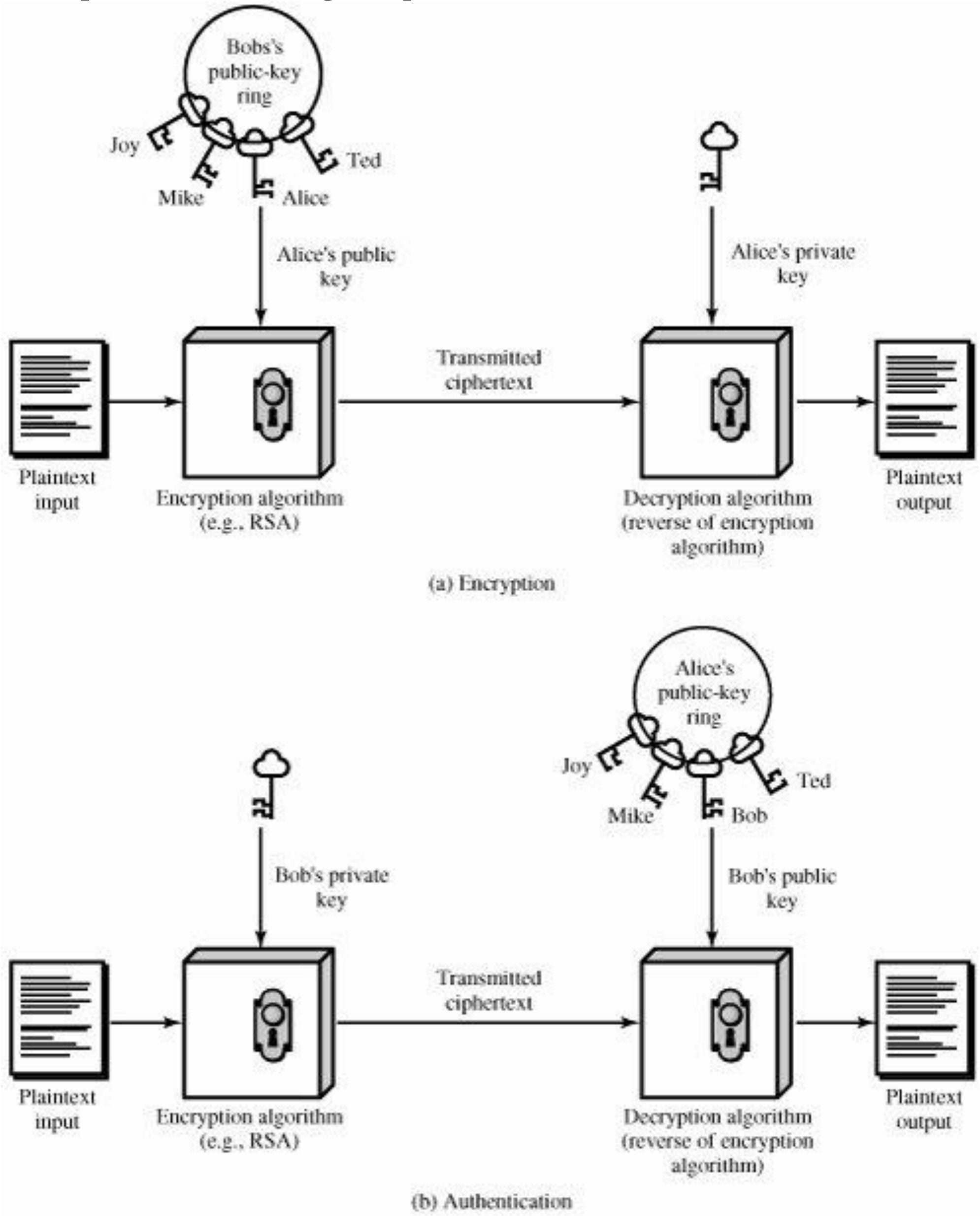

Figure 6.1. Public-Key Cryptography

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 6.1a 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 6.1 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**. 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 6.1. Conventional and Public-Key Encryption

Conventional Encryption	**Public-Key Encryption**
Needed to Work:	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.	1. One algorithm is used for encryption and 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:	Needed for Security:
1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

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 $E(K_a, X)$, $E(PU_a, X)$, and $E(PR_a, X)$, respectively. Similarly, decryption of ciphertext C can be performed with a secret key, a public key, or a private key, denoted by $D(K_a, X)$, $D(PU_a, X)$, and $D(PR_a, X)$, respectively.
Let us take a closer look at the essential elements of a public-key encryption scheme, using Figure 6.2 (compare with Figure 2.2). There is some source A that produces a message in plaintext, $X =[X_1, X_2,..., X_M,]$. 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, PU_b. PU_b is known only to B, whereas PU_b is publicly available and therefore accessible by A.

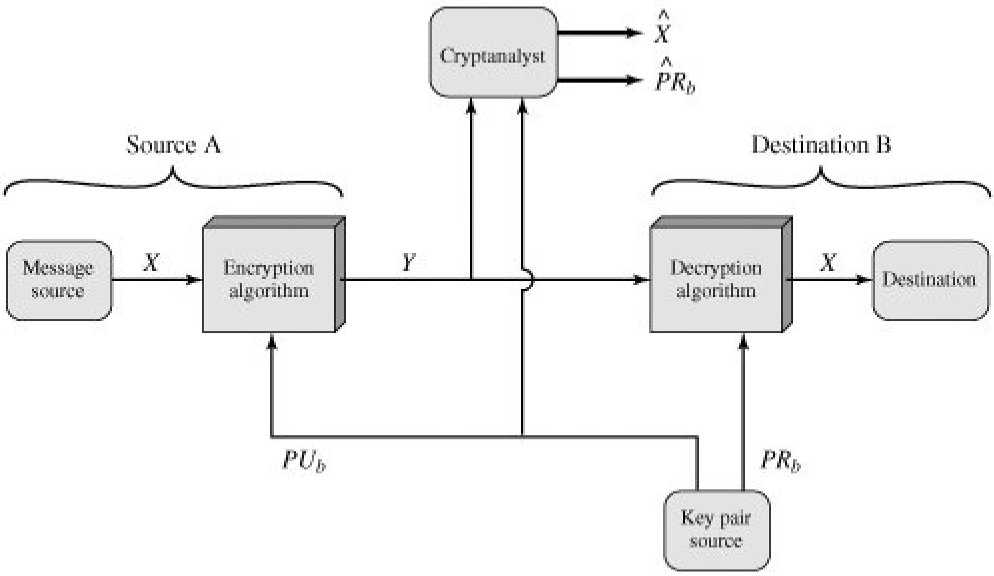

Figure 6.2. Public-Key Cryptosystem: Secrecy

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

$$Y = E(PU_b, X)$$

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

$$X = D(PR_b, Y)$$

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 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 .
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:

$$Y = E(PR_a, X)$$
$$Y = E(PU_a, Y)$$

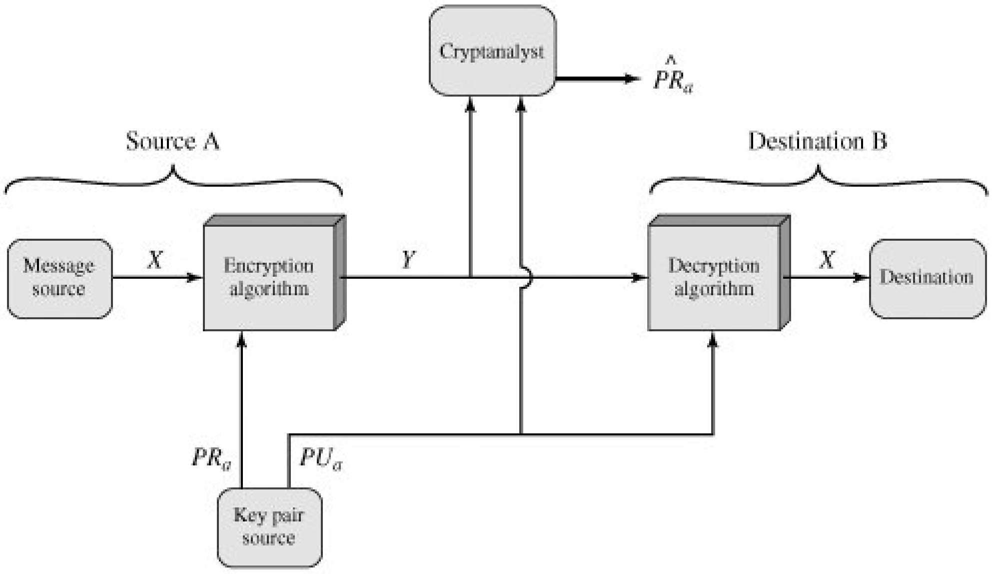

Figure 6.3. Public-Key Cryptosystem: Authentication

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.

It is important to emphasize that the encryption process depicted in Figures 6.1b and 6.3 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 6.3, 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 6.4):

$$Z = E(PU_b, E(PR_a, X))$$
$$X = D(PU_a, E(PR_b, Z))$$

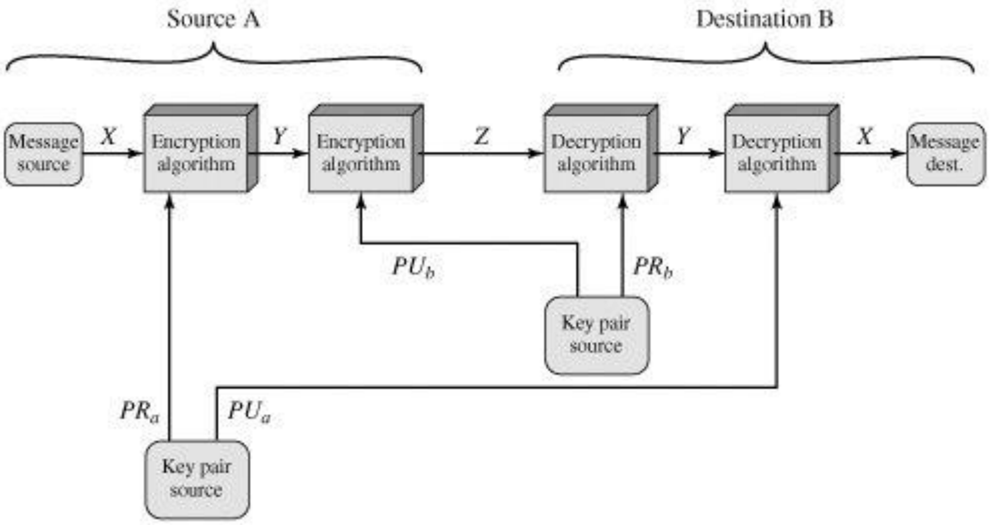

Figure 6.4. Public-Key Cryptosystem: Authentication and Secrecy

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.

6.1.2. 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.

Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. Table 6.2 indicates the applications supported by the algorithms discussed in this book.

Table 6.2. 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

6.1.3. Requirements for Public-Key Cryptography

The cryptosystem illustrated in Figures 6.2 through 6.4 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 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:
   $C = E(PU_b, M)$
3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:
   $M = D(PR_b, C) = D[PR_b, E(PU_b, M)]$
4. It is computationally infeasible for an adversary, knowing the public key, PU_b, to determine the private key, PR_b.
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:
   $M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$

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*[3] 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:

[3] 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

$$Y = f(X) \text{ easy}$$
$$X = f^{1}(X) \text{ infeasible}$$

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.
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

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

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

6.1.4. 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.

6.2. 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 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 [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.

Apparently, the first workable public-key system for encryption/decryption was put forward by Clifford Cocks of Britain's CESG in 1973 [COCK73]; Cocks's method is virtually identical to RSA.

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.

6.2.1. 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 $2^i < n \le 2^{i+1}$. 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 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 ∅(n), where ∅n) is the Euler totient function. that for p, q prime, ∅(pq) = (p - 1)(q - 1) The relationship between e and d can be expressed as

Equation 6-1

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

This is equivalent to saying

$$ed \equiv 1 \bmod \varnothing\,(n)$$
$$d \equiv e^{1} \bmod \varnothing\,(n)$$

That is, e and d are multiplicative inverses mod ∅(n). Note that, according to the rules of modular arithmetic, this is true only if d (and therefore e) is relatively prime to ∅(n). Equivalently, gcd(∅(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(∅(n),e) = 1;1 < e < ∅(n)	(public, chosen)
$d \equiv e^{1}$(mod ∅(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 6.5 summarizes the RSA algorithm. An example, from [SING99], is shown in Figure 6.6. For this example, the keys were generated as follows:

1. Select two prime numbers, p = 17 and q = 11.
2. Calculate n = pq = 17 x 11 = 187.
3. Calculate ∅ (n) = (p 1)(q 1) = 16 x 10 = 160.
4. Select e such that e is relatively prime to ∅ (n) = 160 and less than ∅(n) we choose e = 7.
5. Determine d such that de ≡ 1 (mod 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 (Chapter 4).

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:

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

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

$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$

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

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 6.5. The RSA Algorithm

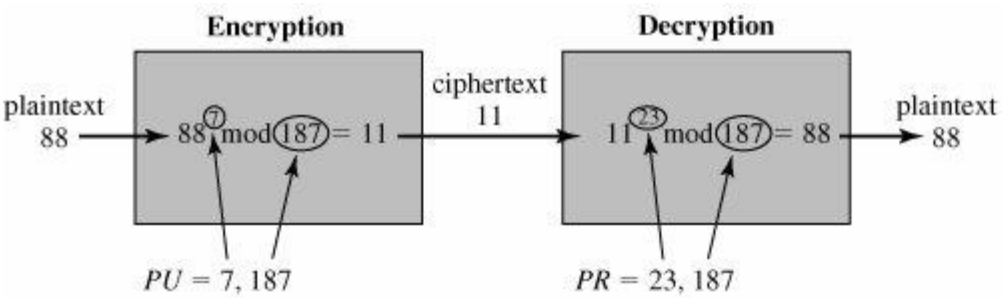

Figure 6.6. Example of RSA Algorithm

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.

6.2.2. 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 mod n) x (b mod n)] mod n = (a x b) mod 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:

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

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} mod 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 mod n, x^2 mod n, x^4 mod n, and x^8 mod n and then calculate [(x mod n)×(x^2 mod n)×(x^8 mod n) mod n. More generally, suppose we wish to find the value a^b with a and b positive integers. If we express b as a binary number $b_k b_{k1} \ldots b_0$ 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 for computing a^b mod n, shown in Figure 6.7. Table 6.3 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. Note: The integer b is expressed as a binary number $b_k b_k 1 \ldots b_0$

```
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
```

Figure 6.7. Algorithm for Computing a^b mod n

Table 6.3. Result of the Fast Modular Exponentiation Algorithm for a^b mod n, where a = 7, b = 560 = 1000110000, 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

6.2.3. 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, 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. 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 $\gcd(\varnothing(n), e) = 1$ and then calculate $d \equiv e^1 (\bmod \varnothing (n))$. 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. Thus, the procedure is to generate a series of random numbers, testing each against $\varnothing(n)$ until a number relatively prime to □(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 □(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.

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.

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

The Factoring Problem

We can identify three approaches to attacking RSA mathematically:

- Factor n into its two prime factors. This enables calculation of □(n) = (p 1) x (q 1), which, in turn, enables determination of $d \equiv e^1 (\bmod \varnothing(n))$.
- Determine $\varnothing(n)$ directly, without first determining p and q. Again, this enables determination of $d \equiv e^1 (\bmod \varnothing(n))$.
- Determine d directly, without first determining $\varnothing(n)$.

Most discussions of the cryptanalysis of RSA have focused on the task of factoring n into its two prime factors. Determining ∅(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 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-200 challenge with a key length of 200 decimal digits, or about 663 bits. Table 6.4 shows the results to date. The level of effort is measured in MIPS-years: a million-instructions-per-second processor running for one year, which is about 3 x 10^{13} instructions executed. A 1 GHz Pentium is about a 250-MIPS machine.

A striking fact about Table 6.4 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.

Table 6.4. Progress in Factorization

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	Quadratic sieve
110	365	April 1992	75	Quadratic sieve
120	398	June 1993	830	Quadratic sieve
129	428	April 1994	5000	Quadratic sieve
130	431	April 1996	1000	Generalized number field sieve
140	465	February 1999	2000	Generalized number field sieve
155	512	August 1999	8000	Generalized number field sieve
160	530	April 2003		Lattice sieve
174	576	December 2003		Lattice sieve
200	663	May 2005		Lattice sieve

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. Figure 6.8 compares the performance of the two algorithms. 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. For the near future, a key size in the range of 1024 to 2048 bits seems reasonable.

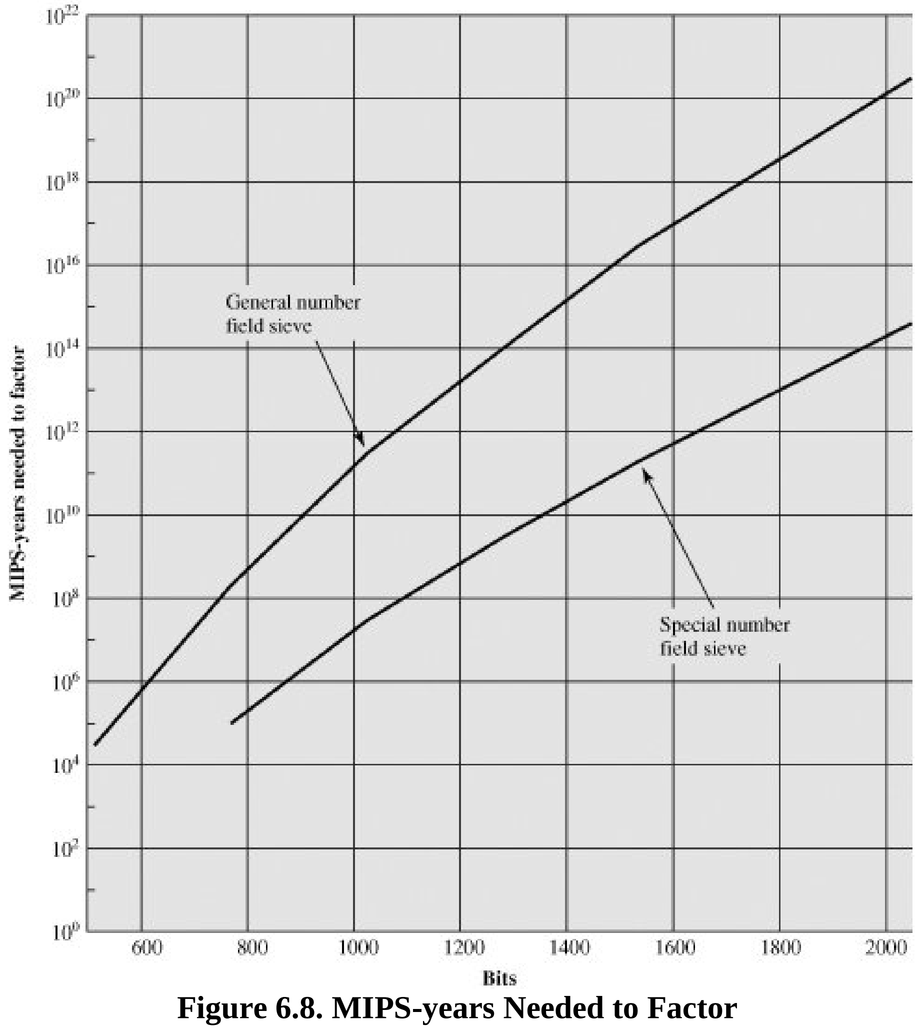

Figure 6.8. MIPS-years Needed to Factor

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].