

CYB650

Introduction to Cryptography

Public Key Algorithms



Prepared by Dr. Esraa Saleh Alomari
Lecture #9



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

The Part 1 of the module 5 covers the Public-Key Algorithms are as follows:

➤ **Rivest-Shamir-Adleman (RSA)**

Introduction

- ❑ 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. A number of algorithms have been proposed for public-key cryptography. Some of these, though initially promising, turned out to be breakable.

- ❑ 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].⁵ 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.

- ❑ 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 $2^{i-6} \leq n < 2^{i+1}$. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C .

RSA Description of the Algorithm

$$C = M^e \bmod n$$

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

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 $M \in \mathbb{Z}_n$.
2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M \in \mathbb{Z}_n$.
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 8 that for p, q prime, $\phi(pq) = (p - 1)(q - 1)$. The relationship between e and d can be expressed as

$$ed \bmod \phi(n) = 1 \quad (9.1)$$

This is equivalent to saying

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

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, $\gcd(\phi(n), d) = 1$. See Appendix R for a proof that Equation (9.1) satisfies the requirement for RSA. 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 cipher-text, 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 $n = pq = 17 * 11 = 187$.
3. Calculate $f(n) = (p - 1)(q - 1) = 16 * 10 = 160$.
4. Select e such that e is relatively prime to $f(n) = 160$ and less than $f(n)$; we choose $e = 7$.
5. Determine d such that $de \equiv 1 \pmod{160}$ and $d \leq 160$. The correct value is $d = 23$, because $23 * 7 = 161 = (1 * 160) + 1$; 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.

RSA Description of the Algorithm

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187)]$$

$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894,432 \bmod 187 = 11$$

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

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

$$\begin{aligned} 11^{23} \bmod 187 &= (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 \\ &= 79,720,245 \bmod 187 = 88 \end{aligned}$$

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

RSA Description of the Algorithm

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:	$C = M^e \pmod{n}$

Decryption by Alice with Alice's Public Key

Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Figure 9.5 The RSA Algorithm

RSA Description of the Algorithm

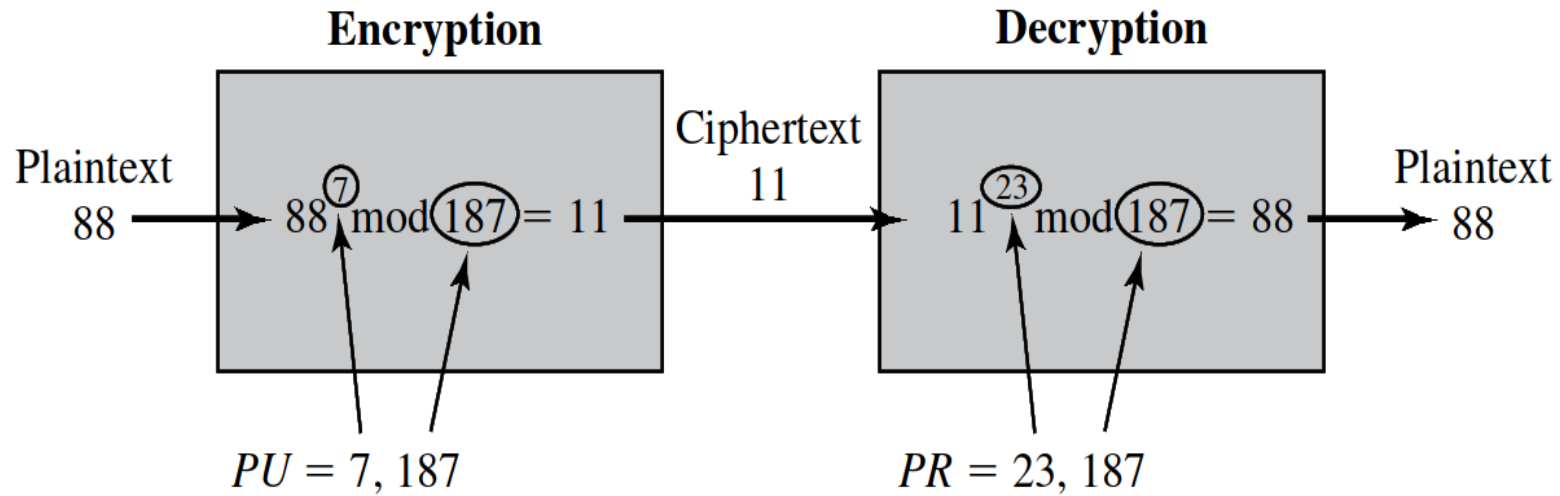


Figure 9.6 Example of RSA Algorithm

digits (e.g., $a = 00$, $A = 26$).⁶ 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.

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.

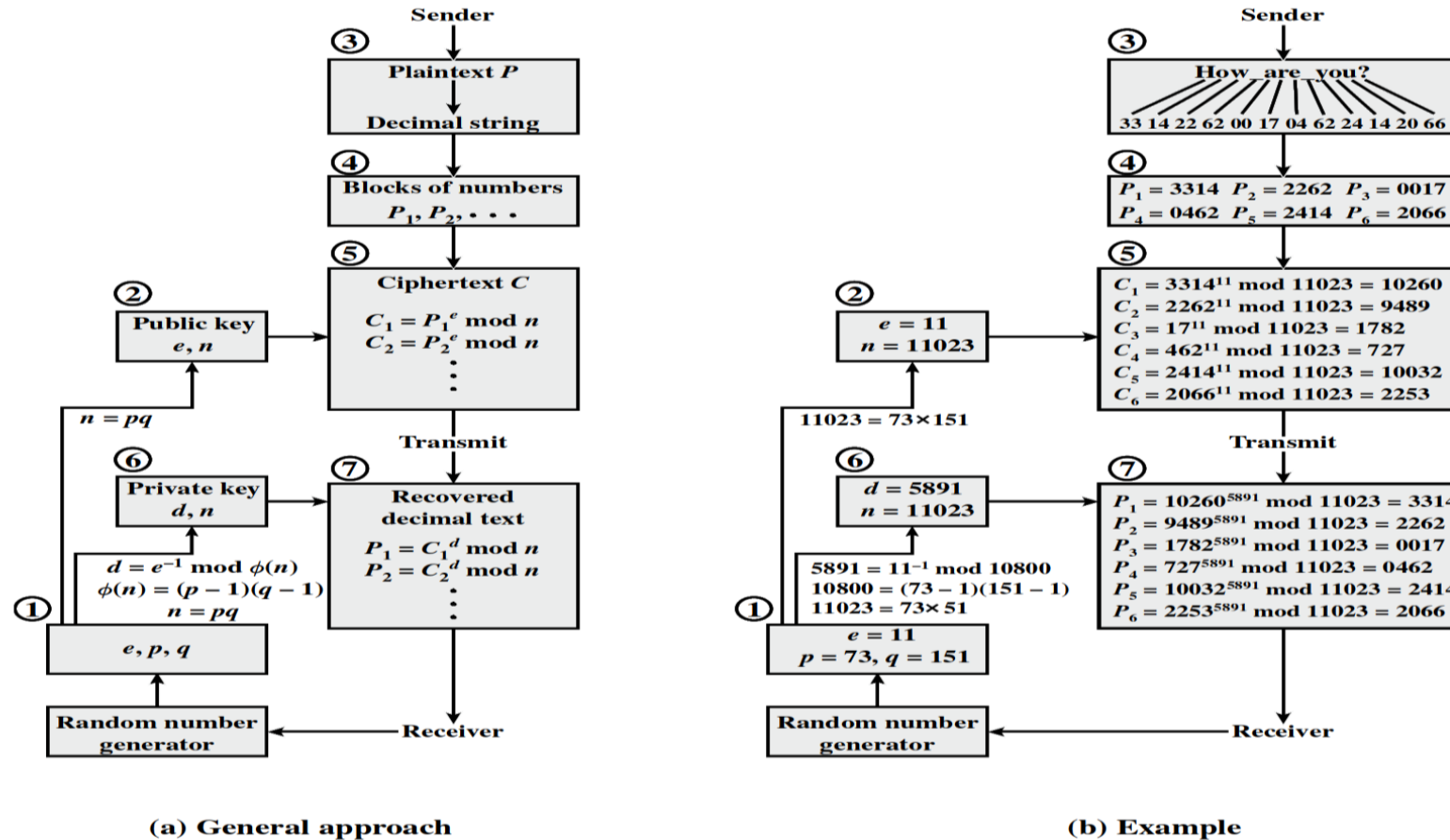


Figure 9.7 RSA Processing of Multiple Blocks

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:

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

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:

$$x^{16} = x * x * x * x * x * x * x * x * x * x * x * x * x * x * x * x$$

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

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.



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 $f(n)$ 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. Table 9.5 shows the results to date. Million-instructions-per-second processor running for one year, which is about 3×10^{13} instructions executed. A 1 GHz Pentium is about a 250-MIPS machine.



A striking fact about the progress reflected in Table 9.5 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



Table 9.5 Progress in RSA Factorization ↵

Number of Decimal Digits ¶	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

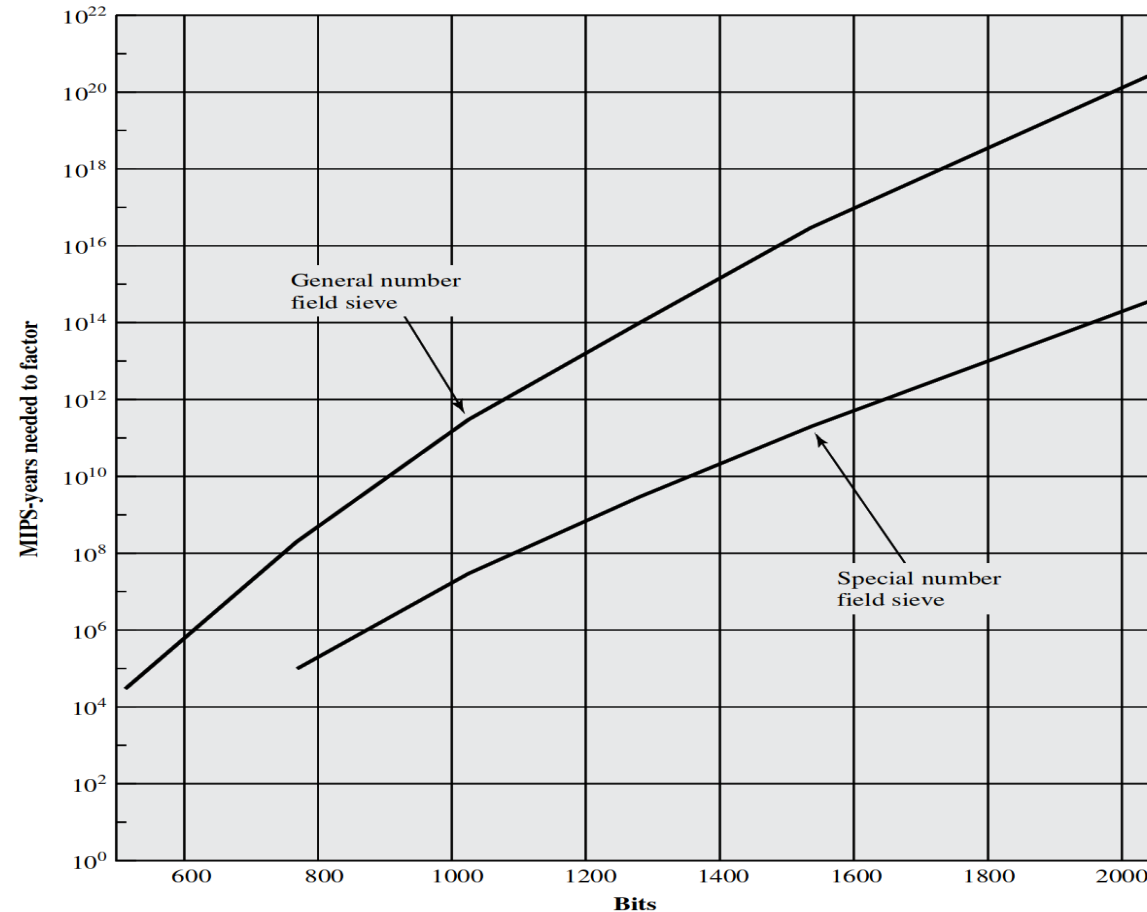


Figure 9.9 MIPS-years Needed to Factor

sieve (SNFS), can factor numbers with a specialized form considerably faster than the generalized number field sieve. Figure 9.9 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. The team that produced the 768-bit factorization made the following observation [KLEI10]:



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. Because the first factorization of a 512-bit RSA modulus

was reported only a decade, it is not unreasonable to expect that 1024-bit RSA moduli can be factored well within the next decade by an academic effort such as ours. Thus, it would be prudent to phase out usage of 1024-bit RSA within the next three to four years.

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].



Thank You

ANY
QUESTIONS?

