

### 3. Applied Number Theory in Computing/Cryptography

*To the layman, a lot of math (like primality testing and factoring large numbers) may seem a frivolous waste of time. However, this research often pays off unexpectedly years later. Factoring and primality testing have become important because of the need to make electronic communications secure. . . . So, what used to be an esoteric playground for mathematicians has become applicable research.*

DAVID GRIES AND FRED B. SCHNEIDER  
*A Logical Approach to Discrete Math* [93]

The aim of this chapter is to introduce some novel applications of elementary and particularly algorithmic number theory to the design of computer (both hardware and software) systems, coding and cryptography, and information security, especially network/communication security.

#### 3.1 Why Applied Number Theory?

The eminent American number theorist Leonard Dickson<sup>1</sup> once said:

Thank God that number theory is unsullied by any application.

---

1



Leonard Eugene Dickson (1874–1954), one of the key figures of 20th century mathematics, particularly number theory, was born in Independence, Iowa, a descendant of one William Dickson who had emigrated from Londonderry, Northern Ireland to Londonderry, New Hampshire in the 18th century. Dickson obtained his PhD in 1896 from the University of Chicago, the first PhD awarded in Mathematics by the institution. Following periods at the Universities of Leipzig, Paris, California and Texas, he returned to Chicago in 1900, becoming a full professor in 1910. One of the most productive of all mathematicians, Dickson wrote over 250 papers and 18 books, including the three volume 1600 page *History of the Theory of Numbers* [65]. It is amusing to note that he stopped to write papers and books in mathematics abruptly and completely on reaching the age of 65 in 1939 and devoted himself to his recreations, including bridge, tennis and billiards. (Photo by courtesy of the American Mathematical Society.)

The most famous English mathematician G. H. Hardy (1877–1947) also in his *Apology* ([98], page 120) expressed that

If the theory of numbers could be employed for any practical and obviously honourable purpose, if it could be turned directly to the furtherance of human happiness or relief of human suffering, as physiology and even chemistry can, then surely neither Gauss or any other mathematician would have been so foolish as to decry or regret such applications.

He then further proudly stated on page 140 that

Real mathematics has no effects on war. No one has yet discovered any warlike purpose to be served by the theory of numbers.

The above famous quotations made by the two greatest mathematicians of the 20th century may be true before 1950, but certainly are not true at the present time, since, e.g., number theory now can help the generals to plan their battles in a completely secret way. Remarkably enough, the great Russian mathematician Nikolay Lobachevsky (1792–1856) predicated nearly 200 years ago that

There is no branch of mathematics, however abstract, which may not some day be applied to phenomena of the real world.

In fact, any branch of pure mathematics will eventually find real world applications. Number theory, for example, was considered the purest branch of pure mathematics, with no direct applications to the real world. The advent of digital computers and digital communications and particularly public-key cryptography revealed that number theory could provide unexpected answers to real-world problems. As showed in Schroeder [222] and Waldschmidt, Moussa, Luck and Itzykson [250], and Guterl [96], number theory has already been successfully applied to such diverse areas as physics, biology, chemistry, computing, engineering, coding and cryptography, random number generation, acoustics, communications, graphic design, and even music and business. It is also interesting to note that the eminent mathematician Shiing-Shen Chern (1911– ), the 1980 Wolf Prize Winner, even considers number theory as a branch of applied mathematics [48] because of its strong applicability in other fields. Today, number theory is used widely in computing and information theory/technology, due in part to the invention of the high-speed computers based on e.g., the residue number systems and the cryptographic schemes based on e.g., large prime numbers. For example, the feasibility of several modern cryptographic schemes rests on our ability to find large primes easily, while their security rests on our inability to factor the product of large primes.

Number theory is generally considered to be laid in the discrete, finite side of mathematics, along with algebra and combinatorics, and is intimately connected to computing science and technology, since computers are basically

finite machines; they have finite storage and can only deal with numbers of some finite length. Because of these features in computing, number theory is particularly useful and applicable to computing. For example, congruence theory has been used for devising systematic methods for storing computer files, generating random numbers, designing highly secure and reliable encryption schemes and even developing high-speed residue computers. Since most computer scientists are more interested in the applications of number theory in computing, rather than the number theory itself, in this chapter, we shall apply the number-theoretic results and algorithms from the previous two chapters to the design of fast computer architectures, and more secure, more reliable computer/network systems.

## 3.2 Computer Systems Design

*... virtually every theorem in elementary number theory arises in a natural, motivated way in connection with the problem of making computers do high-speed numerical calculations.*

DONALD E. KNUTH

Computer Science and its Relation to Mathematics [121]

### 3.2.1 Representing Numbers in Residue Number Systems

The way we do arithmetic on numbers is intimately related to the way we represent the numbers. There are essentially two different types of methods to represent numbers: *nonpositional* and *positional*. The Roman numerals i, ii, iii, iv, v, vi, vii, viii, ix, x, xi, xii, xiii, ... are a classical example of a nonpositional number system; whereas the familiar *decimal* or *binary* number system are good examples of a positional number system. The positional number system using base  $b$  (or radix  $b$ ) is defined by the rule

$$(\cdots a_3 a_2 a_1 a_0 a_{-1} a_{-2} a_{-3} \cdots)_b = \cdots + a_3 b^3 + a_2 b^2 + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} a_{-2} b^{-2} a_{-3} b^{-3} + \cdots \quad (3.1)$$

It is clear that when  $b = 10$ , it is the decimal system, whereas when  $b = 2$  we have the binary system. This type of positional number system is said to have a *fixed-base* or *fixed-radix*. A positional number system which is not fixed-base is said to be *mixed-base*. The number systems, residue number systems, we shall study in this section are a type of mixed-base system.

Let us first recall the Fundamental Theorem of Arithmetic: any positive integer  $n \in \mathbb{N}_{>1}$  can be uniquely written as

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k} = n_1 n_2 \cdots n_k \quad (3.2)$$

where  $p_1, p_2, \dots, p_k$  are distinct primes,  $\alpha_1, \alpha_2, \dots, \alpha_k$  are natural numbers,  $n_i = p_i^{\alpha_i}$ ,  $i = 1, 2, \dots, k$ , and  $\gcd(n_i, n_j) = 1$  for  $i \neq j$ . The prime decomposition of  $n$  can be used to represent any number in  $\mathbb{Z}/n\mathbb{Z}$  in terms of the numbers in  $\mathbb{Z}/n_i\mathbb{Z}$ , for  $i = 1, 2, \dots, k$ .

**Definition 3.2.1.** Let  $x$  be any number in  $\mathbb{Z}/n\mathbb{Z}$  and

$$\left. \begin{array}{l} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \\ \dots\dots\dots \\ x \equiv a_k \pmod{n_k} \end{array} \right\} \quad (3.3)$$

then the  $k$ -tuple

$$\langle a_1, a_2, \dots, a_k \rangle = \langle x \bmod n_1, x \bmod n_2, \dots, x \bmod n_k \rangle \quad (3.4)$$

is called the residue (congruence, or modular) representation of  $x$ . For simplicity, we often write the residue representation of  $x$  as follows:

$$x \iff \langle x \bmod n_1, x \bmod n_2, \dots, x \bmod n_k \rangle \quad (3.5)$$

**Example 3.2.1.** Let  $n_1 = 3$ ,  $n_2 = 5$ ,  $n_3 = 7$ , then the residue representation of the integer 103 will be

$$\left\{ \begin{array}{l} 103 \equiv 1 \pmod{3} \\ 103 \equiv 3 \pmod{5} \\ 103 \equiv 5 \pmod{7} \end{array} \right.$$

That is

$$103 \iff (1, 3, 5).$$

Note that the residue representation of an integer  $x$  wrt moduli  $n_1, n_2, \dots, n_k$  is unique. However, the inverse is not true.

**Example 3.2.2.** Let again  $n_1 = 3$ ,  $n_2 = 5$ ,  $n_3 = 7$ , then all the numbers in the form

$$105t + 103, \quad \text{for } t \in \mathbb{N}$$

have the same residue representation  $(1, 3, 5)$ . That is,

$$105t + 103 \iff (1, 3, 5).$$

**Definition 3.2.2.** Let  $(\mathbb{Z}/n\mathbb{Z})^*$  be the “direct-product” decomposition of  $\mathbb{Z}/n\mathbb{Z}$ . That is,

$$(\mathbb{Z}/n\mathbb{Z})^* = \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z} \times \dots \times \mathbb{Z}/n_k\mathbb{Z} \quad (3.6)$$

where  $n_i = p_i^{\alpha_i}$  for  $i = 1, 2, \dots, k$  is the prime decomposition of  $n$ .

**Theorem 3.2.1.** Let  $m_1 > 0, m_2 > 0, \dots, m_k > 0$ , and  $\gcd(m_i, m_j) = 1$  with  $0 < i < j \leq k$ . Then two integers  $x$  and  $x'$  have the same residue representation if and only if

$$x \equiv x' \pmod{M} \quad (3.7)$$

where  $M = m_1 m_2 \cdots m_k$ .

So if we restrict  $0 \leq x < M = m_1 m_2 \cdots m_k$ , then different integers  $x$  will have different residue representation moduli  $m_1, m_2, \dots, m_k$ .

**Theorem 3.2.2.** Let  $f : \mathbb{Z}/n\mathbb{Z} \rightarrow (\mathbb{Z}/n\mathbb{Z})^*$  be such that for any  $x \in \mathbb{Z}/n\mathbb{Z}$ , we have

$$\begin{aligned} f(x) &= (a_1, a_2, \dots, a_k) \\ &= (x \bmod n_1, x \bmod n_2, \dots, x \bmod n_k) \end{aligned} \quad (3.8)$$

then  $f$  is a bijection (one-to-one and onto).

**Remark 3.2.1.** Theorem 3.2.1 is just another form of the Chinese Remainder Theorem.

**Example 3.2.3.** Let  $m = 30$ , so that  $m_1 = 2, m_2 = 3, m_3 = 5$  with

$$(\mathbb{Z}/30\mathbb{Z})^* = \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/5\mathbb{Z}.$$

Then the residue representations for integers in  $\mathbb{Z}/30\mathbb{Z}$  will be:

$0 \iff (0, 0, 0)$	$1 \iff (1, 1, 1)$
$2 \iff (0, 2, 2)$	$3 \iff (1, 0, 3)$
$4 \iff (0, 1, 4)$	$5 \iff (1, 2, 0)$
$6 \iff (0, 0, 1)$	$7 \iff (1, 1, 2)$
$8 \iff (0, 2, 3)$	$9 \iff (1, 0, 4)$
$10 \iff (0, 1, 0)$	$11 \iff (1, 2, 1)$
$12 \iff (0, 0, 2)$	$13 \iff (1, 1, 3)$
$14 \iff (0, 2, 4)$	$15 \iff (1, 0, 0)$
$16 \iff (0, 1, 1)$	$17 \iff (1, 2, 2)$
$18 \iff (0, 0, 3)$	$19 \iff (1, 1, 4)$
$20 \iff (0, 2, 0)$	$21 \iff (1, 0, 1)$
$22 \iff (0, 1, 2)$	$23 \iff (1, 2, 3)$
$24 \iff (0, 0, 4)$	$25 \iff (1, 1, 0)$
$26 \iff (0, 2, 1)$	$27 \iff (1, 0, 2)$
$28 \iff (0, 1, 3)$	$29 \iff (1, 2, 4)$

Once the residue representation

$$(a_1, a_2, \dots, a_k) = (x \bmod n_1, x \bmod n_2, \dots, x \bmod n_k)$$

of an integer  $x$  is given, then we can uniquely solve  $x$  by using the Chinese Remainder Theorem (see the following example).

**Example 3.2.4.** Suppose we have the residue representations of  $x$  as follows:

$$x = (x \bmod 3, x \bmod 5, x \bmod 7) = (1, 3, 5).$$

Then we have

$$\begin{aligned} x &\equiv 1 \pmod{3}, \\ x &\equiv 3 \pmod{5}, \\ x &\equiv 5 \pmod{7}. \end{aligned}$$

By using the Chinese Remainder Theorem, we get:

$$x = 103.$$

On most computers the word size is a large power of 2, with  $2^{32}$  a common value. So to use residue arithmetic and the Chinese Remainder Theorem to do arithmetic, we need the moduli less than, say  $2^{32}$ , pairwise relatively prime and multiplying together to give a large integer.

### 3.2.2 Fast Computations in Residue Number Systems

In this subsection, we shall discuss fast arithmetic operations in residue number systems. More specifically, we shall discuss the fast arithmetic operations of addition  $+_n$ , subtraction  $-_n$  and multiplication  $\cdot_n$  in  $(\mathbb{Z}/n\mathbb{Z})^*$  in terms of the corresponding operations  $+_{n_i}$ ,  $-_{n_i}$ ,  $\cdot_{n_i}$  in  $\mathbb{Z}/n_i\mathbb{Z}$ , for  $i = 1, 2, \dots, k$ .

**Definition 3.2.3.** Let  $x = (a_1, a_2, \dots, a_k)$  and  $y = (b_1, b_2, \dots, b_k)$  in  $\mathbb{Z}/n\mathbb{Z}$ . Then

$$\begin{aligned} x + y &= (a_1, a_2, \dots, a_k) +_n (b_1, b_2, \dots, b_k) \\ &= f(x) +_n f(y) \\ &= ((x \bmod n_1) +_{n_1} (y \bmod n_1), \\ &\quad (x \bmod n_2) +_{n_2} (y \bmod n_2), \\ &\quad \dots \dots \dots \\ &\quad \dots \dots \dots \\ &\quad (x \bmod n_k) +_{n_k} (y \bmod n_k)). \end{aligned}$$

$$\begin{aligned} x - y &= (a_1, a_2, \dots, a_k) -_n (b_1, b_2, \dots, b_k) \\ &= f(x) -_n f(y) \\ &= ((x \bmod n_1) -_{n_1} (y \bmod n_1), \\ &\quad (x \bmod n_2) -_{n_2} (y \bmod n_2), \\ &\quad \dots \dots \dots \\ &\quad \dots \dots \dots \\ &\quad (x \bmod n_k) -_{n_k} (y \bmod n_k)). \end{aligned}$$

$$\begin{aligned}
x \cdot y &= (a_1, a_2, \dots, a_k) \cdot_n (b_1, b_2, \dots, b_k) \\
&= f(x) \odot_n f(y) \\
&= ((x \bmod n_1) \cdot_{n_1} (y \bmod n_1), \\
&\quad (x \bmod n_2) \cdot_{n_2} (y \bmod n_2), \\
&\quad \dots \dots \dots \\
&\quad \dots \dots \dots \\
&\quad (x \bmod n_k) \cdot_{n_k} (y \bmod n_k)).
\end{aligned}$$

**Definition 3.2.4.** Given groups  $(\mathcal{G}, *)$  and  $(\mathcal{H}, \star)$ , a function  $f : \mathcal{G} \rightarrow \mathcal{H}$  is called an *isomorphism* if the following conditions hold:

- (1)  $f$  is one-to-one and onto.
- (2)  $f(a * b) = f(a) \star f(b)$ , for all  $a, b \in \mathcal{G}$ .

We say that  $(\mathcal{G}, *)$  is *isomorphic* to  $(\mathcal{H}, \star)$  and write  $(\mathcal{G}, *) \cong (\mathcal{H}, \star)$ .

**Example 3.2.5.** Show that the function  $f : (\mathbb{R}, +) \rightarrow (\mathbb{R}^+, \cdot)$  defined by  $f(x) = 2^x$  is an isomorphism. First, we have:

- (1)  $f$  is one-to-one, since  $f(x) = f(y)$  implies  $2^x = 2^y$ , which implies  $x = y$ .  
Also  $f$  is onto, since for each  $r \in \mathbb{R}^+$  there is  $t \in \mathbb{R}$  such that  $2^t = r$ , namely  $t = \log_2 r$ .
- (2) Let  $a, b \in \mathbb{R}$ , then  $f(a + b) = 2^{a+b} = 2^a \cdot 2^b = f(a) \cdot f(b)$ .

Therefore  $f$  is an isomorphism. That is

$$(\mathbb{R}, +) \cong (\mathbb{R}^+, \cdot), \quad f(x) = 2^x.$$

**Theorem 3.2.3.** Let  $f : \mathbb{Z}/n\mathbb{Z} \rightarrow (\mathbb{Z}/n\mathbb{Z})^*$  defined by

$$f(x) = (x \bmod n_1, x \bmod n_2, \dots, x \bmod n_k)$$

be one-to-one and onto. Then

- (1)  $(\mathbb{Z}/n\mathbb{Z}, +_n) \cong ((\mathbb{Z}/n\mathbb{Z})^*, +_{n_i})$ .
- (2)  $(\mathbb{Z}/n\mathbb{Z}, -_n) \cong ((\mathbb{Z}/n\mathbb{Z})^*, -_{n_i})$ .
- (3)  $(\mathbb{Z}/n\mathbb{Z}, \cdot_n) \cong ((\mathbb{Z}/n\mathbb{Z})^*, \cdot_{n_i})$ .

The above theorem tells us that the arithmetic operations  $+_n, -_n$  and  $\cdot_n$  in  $\mathbb{Z}/n\mathbb{Z}$  can be done in  $(\mathbb{Z}/n\mathbb{Z})^*$  by means of the corresponding operations  $+_{n_i}, -_{n_i}$  and  $\cdot_{n_i}$  in  $(\mathbb{Z}/n_i\mathbb{Z})^*$ , for  $i = 1, 2, \dots, k$ . This is exactly what we need. In what follows, we shall give two examples of adding and multiplying two large integers in residue number systems. Later in the next subsection we shall also discuss its hardware implementation.

**Example 3.2.6.** Compute  $z = x + y = 123684 + 413456$  on a computer of word size 100. Firstly we have

$$\begin{aligned} x &\equiv 33 \pmod{99}, & y &\equiv 32 \pmod{99}, \\ x &\equiv 8 \pmod{98}, & y &\equiv 92 \pmod{98}, \\ x &\equiv 9 \pmod{97}, & y &\equiv 42 \pmod{97}, \\ x &\equiv 89 \pmod{95}, & y &\equiv 16 \pmod{95}, \end{aligned}$$

so that

$$\left. \begin{aligned} z = x + y &\equiv 65 \pmod{99}, \\ z = x + y &\equiv 2 \pmod{98}, \\ z = x + y &\equiv 51 \pmod{97}, \\ z = x + y &\equiv 10 \pmod{95}. \end{aligned} \right\} \quad (3.9)$$

Now, we use the Chinese Remainder Theorem to find

$$x + y \pmod{99 \times 98 \times 97 \times 95}.$$

Note that the solution to (3.9) is

$$z \equiv \sum_{i=1}^4 M_i M'_i z_i \pmod{m},$$

where

$$\begin{aligned} m &= m_1 m_2 m_3 m_4, \\ M_i &= m/m_i, \\ M'_i M_i &\equiv 1 \pmod{m_i}, \end{aligned}$$

for  $i = 1, 2, 3, 4$ . Now we have

$$M = 99 \times 98 \times 97 \times 95 = 89403930,$$

and

$$\begin{aligned} M_1 &= M/99 = 903070, \\ M_2 &= M/98 = 912285, \\ M_3 &= M/97 = 921690, \\ M_4 &= M/95 = 941094. \end{aligned}$$

We need to find the inverse  $M'_i$ , for  $i = 1, 2, 3, 4$ . To do this, we solve the following four congruences

$$\begin{aligned} 903070 M'_1 &\equiv 91 M'_1 \equiv 1 \pmod{99}, \\ 912285 M'_2 &\equiv 3 M'_2 \equiv 1 \pmod{98}, \\ 921690 M'_3 &\equiv 93 M'_3 \equiv 1 \pmod{97}, \\ 941094 M'_4 &\equiv 24 M'_4 \equiv 1 \pmod{95}. \end{aligned}$$

We find that

$$\begin{aligned} M'_1 &\equiv 37 \pmod{99}, \\ M'_2 &\equiv 38 \pmod{98}, \\ M'_3 &\equiv 24 \pmod{97}, \\ M'_4 &\equiv 4 \pmod{95}. \end{aligned}$$

Hence we get:



$$\begin{aligned}
x + y &\equiv \sum_{i=1}^4 z_i M_i M'_i \pmod{m} \\
&\equiv 65 \times 903070 \times 37 + 2 \times 912285 \times 38 + 51 \times 921690 \times 24 \\
&\quad + 10 \times 941094 \times 4 \pmod{89403930} \\
&\equiv 3397886480 \pmod{89403930} \\
&\equiv 537140 \pmod{89403930}
\end{aligned}$$

Since  $0 < x + y = 537140 < 89403930$ , we conclude that  $x + y = 537140$  is the correct answer.

**Example 3.2.7.** Suppose now we want to multiply  $x = 123684$  and  $y = 413456$  on a computer of word size 100. We then have

$$\begin{array}{ll}
x \equiv 33 \pmod{99}, & y \equiv 32 \pmod{99}, \\
x \equiv 8 \pmod{98}, & y \equiv 92 \pmod{98}, \\
x \equiv 9 \pmod{97}, & y \equiv 42 \pmod{97}, \\
x \equiv 89 \pmod{95}, & y \equiv 16 \pmod{95}, \\
x \equiv 63 \pmod{89}, & y \equiv 51 \pmod{89}, \\
x \equiv 14 \pmod{83}, & y \equiv 33 \pmod{83},
\end{array}$$

so that

$$\begin{aligned}
x \cdot y &\equiv 66 \pmod{99}, \\
x \cdot y &\equiv 50 \pmod{98}, \\
x \cdot y &\equiv 87 \pmod{97}, \\
x \cdot y &\equiv 94 \pmod{95}, \\
x \cdot y &\equiv 9 \pmod{89}, \\
x \cdot y &\equiv 47 \pmod{83}.
\end{aligned}$$

Now using the Chinese Remainder Theorem to solve the above system of congruences, we get

$$x \cdot y = 51137891904.$$

Since

$$0 < x \cdot y = 51137891904 < 803651926770 = n_1 n_2 n_3 n_4 n_5 n_6$$

we conclude that  $x \cdot y = 51137891904$  is the correct answer.

In what follows, we shall present a general algorithm for residue arithmetic in  $(\mathbb{Z}/n\mathbb{Z})^*$ , where  $n = n_1 n_2 \cdots n_k$ , by means of the corresponding operations in  $(\mathbb{Z}/n_i\mathbb{Z})^*$ , for  $i = 1, 2, \dots, k$ . Readers note that there are three different types of arithmetic:

- (1) Integer arithmetic: arithmetic in  $\mathbb{Z}$ .
- (2) Modular arithmetic: special integer arithmetic in  $\mathbb{Z}/n\mathbb{Z}$ .
- (3) Residue arithmetic: special modular arithmetic in  $(\mathbb{Z}/n\mathbb{Z})^*$ .

In this book, we are actually more interested in the last two types of arithmetic.

**Algorithm 3.2.1 (Residue arithmetic).** This algorithm performs the residue arithmetic in  $(\mathbb{Z}/n\mathbb{Z})^*$ , where  $n = n_1 n_2 \cdots n_k$ :

- [1] Convert integers to their residue representation: Represent integers, for example,  $x$  and  $y$  as elements of the group  $(\mathbb{Z}/n\mathbb{Z})^*$ , where

$$(\mathbb{Z}/n\mathbb{Z})^* = \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$$

by taking the congruence class of  $x$  or  $y$  modulo each  $n_i$ ; for example, the following is the residue representation of  $x$  and  $y$  modulo each  $n_i$ :

$$(x \equiv x_1 \pmod{n_1}, x \equiv x_2 \pmod{n_2}, \dots, x \equiv x_k \pmod{n_k}),$$

$$(y \equiv y_1 \pmod{n_1}, y \equiv y_2 \pmod{n_2}, \dots, y \equiv y_k \pmod{n_k}).$$

- [2] Perform the residue arithmetic for each  $\mathbb{Z}/n_i\mathbb{Z}$ : For example, if  $\star$  denotes one of the three binary operations  $+$ ,  $-$  and  $\cdot$ , then we need to perform the following operations in  $\mathbb{Z}/n_i\mathbb{Z}$ :

$$(x_1 \star y_1 \pmod{n_1}, x_2 \star y_2 \pmod{n_2}, \dots, x_k \star y_k \pmod{n_k}).$$

- [3] Convert the residue representations back to integers: Use the Chinese Remainder Theorem to convert the computation results for each  $\mathbb{Z}/n_i\mathbb{Z}$  into their integer form in  $\mathbb{Z}/n\mathbb{Z}$

$$x \star y \equiv \sum_{i=1}^k M_i M'_i z_i \pmod{M},$$

where

$$M = n_1 n_2 \cdots n_k,$$

$$M_i = M/n_i,$$

$$M'_i \equiv M_i^{-1} \pmod{n_i},$$

$$z_i \equiv x_i \star y_i \pmod{n_i},$$

for  $i = 1, 2, \dots, k$ .

The above algorithm can be implemented entirely in special computer hardware, which is the subject matter of our next subsection.

### 3.2.3 Residue Computers

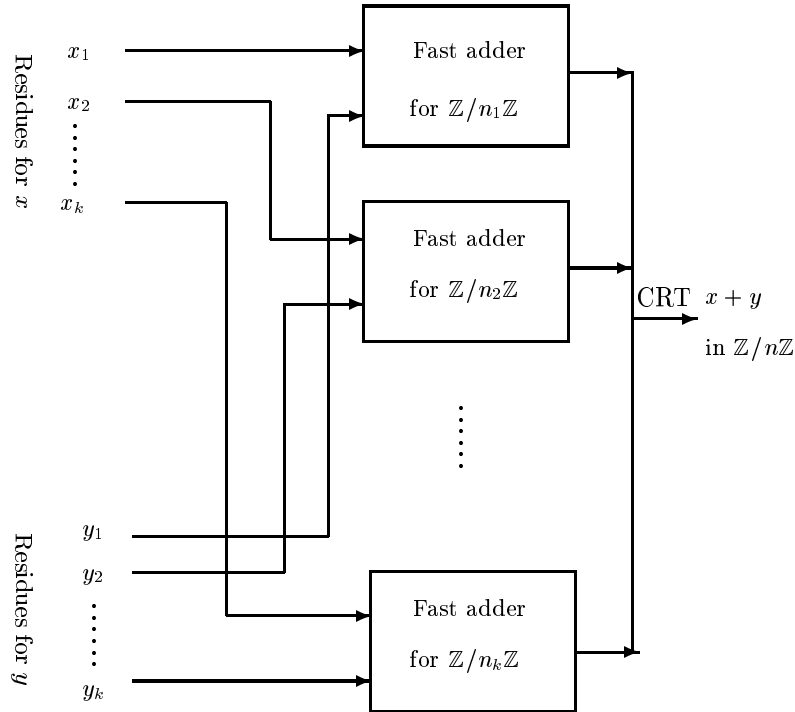
The conventional “binary computers” have a serious problem that restricts the speed of performing arithmetic operations, caused by, for example, the carry propagation and time delay. Fortunately, the residue number system (RNS) is not a fixed-base number system, and all arithmetic operations (except division) in RNS are inherently carry-free; that is, each digit in the computed result is a function of only the corresponding digits of the operands.

Consequently, addition, subtraction and multiplication can be performed in “residue computers” in less time than that needed in equivalent binary computers.

The construction of residue computers is much easier than that of binary computers; for example, to construct fast adders of a residue computer for

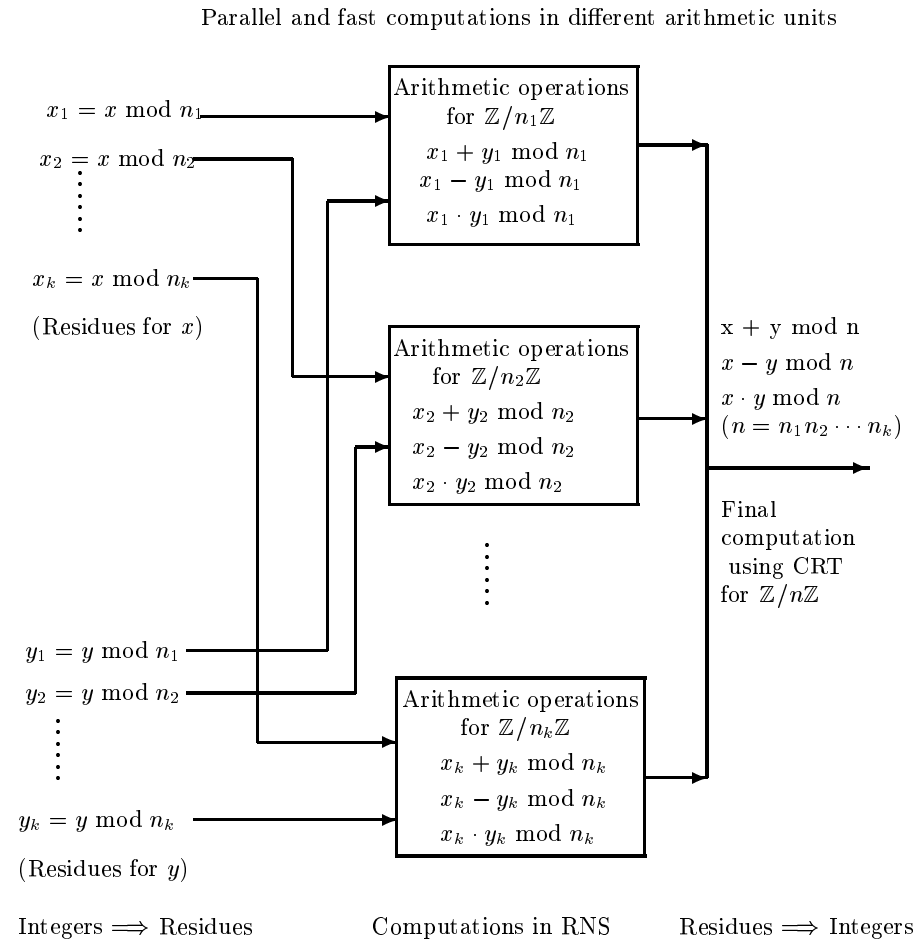
$$(\mathbb{Z}/n\mathbb{Z})^* = \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$$

it is sufficient to just construct some smaller adders for each  $\mathbb{Z}/n_i\mathbb{Z}$ , ( $i = 1, 2, \dots, k$ ) (see Figure 3.1). More generally, we can construct residue com-



**Figure 3.1.** Fast adders for residue arithmetic

puters performing fast additions, subtractions and multiplications as in Figure 3.2. Since  $n_i$  is substantially less than  $n$ , computations in each  $\mathbb{Z}/n_i\mathbb{Z}$  will certainly be much easier than those in  $\mathbb{Z}/n\mathbb{Z}$ . More importantly, additions, subtractions and multiplications in each  $\mathbb{Z}/n_i\mathbb{Z}$  are carry-free, so residue computers will be substantially faster than conventional binary computers inherently with carry propagation. The idea of decomposing a large computation



**Figure 3.2.** A model of residue computers

in  $\mathbb{Z}/n\mathbb{Z}$  into several smaller computations in the  $\mathbb{Z}/n_i\mathbb{Z}$  is exactly the idea of “divide-and-conquer” used in algorithm design. Of course, the central idea behind residue arithmetic and residue computers is the Chinese Remainder Theorem which enables us to combine separate results in each  $\mathbb{Z}/n_i\mathbb{Z}$  to a final result in  $\mathbb{Z}/n\mathbb{Z}$ . So, if Euclid’s algorithm is regarded as the first nontrivial algorithm, then the Chinese Remainder Theorem should be regarded as the first nontrivial divide-and-conquer algorithm.

Residue computers are a special type of high-speed computer, that has found many important applications in several central areas of computer science and electrical engineering, particularly in image and digital signal processing (Krishna, Krishna, Lin, and Sun [133]).

### 3.2.4 Complementary Arithmetic

The main memory of a computer is divided into a number of units of equal size, called *words*. Each word consists of  $n = 2^m$  bits, where  $n$  is typically 16, 32, 64 or 128. Provided that a positive integer is not too large, it can then be represented simply by its binary form in a single word of the computer memory. For example, a 16-bit word could hold the positive values from 0 to 65535. The problem is then how to represent negative integers. There are a number of ways to do this; the most obvious way is the signed-magnitude representation.

**Definition 3.2.5.** In the signed-magnitude representation, the first bit of the  $m$ -bit word is used to denote the sign (0 for + and 1 for -), called the sign bit, and the remaining  $m - 1$  bits are used to represent the size, or magnitude of the number in binary form.

**Example 3.2.8.** Let  $m = 8$ . Then to represent the integers +117 and -127 in signed-magnitude representation, we have:

$$\begin{array}{rcl}
 +117 & \Rightarrow & \begin{array}{cc} 0 & 1110101 \\ \downarrow & \downarrow \\ \text{sign bit} & \text{number magnitude} \end{array} \\
 \\ 
 -127 & \Rightarrow & \begin{array}{cc} 1 & 1111111 \\ \downarrow & \downarrow \\ \text{sign bit} & \text{number magnitude.} \end{array}
 \end{array}$$

In a computer with 16-bit words, using the signed-magnitude representation, the largest integer that can be stored is

$$0 \underbrace{111111111111111}_{15 \text{ ones}} = 2^{15} - 1 = 32767$$

and the smallest is

$$1 \underbrace{111111111111111}_{15 \text{ ones}} = 1 - 2^{15} = -32767.$$

**Example 3.2.9.** Let  $m = 8$ . To compute  $117 + (-127)$  and  $117 + 127$  in the signed-magnitude representation, we have:

$$\begin{array}{rcl}
 & \begin{array}{cc} 01110101 & +1110101 \\ + & 11111111 \\ \hline \end{array} & \Rightarrow \begin{array}{cc} -0001010 & \Rightarrow 10001010 \Rightarrow -10 \end{array} \\
 & \begin{array}{cc} & -1111111 \\ \hline \end{array} & 
 \end{array}$$

$$\begin{array}{rcl}
& 01110101 & +1110101 \\
+ & 01111111 & +1111111 \\
\hline
& \Rightarrow & \hline
& +1110100 & \Rightarrow 01110100 \Rightarrow 116.
\end{array}$$

Note that in the above computation the most significant bit is the sign bit that does not take part in the computation itself: we need first to convert the sign bit to either + or −, then perform the computation and convert the sign of the result into a sign bit. Note also that  $117 + 127 = 116 \neq 244$ ; this is because the adder in a computer operates modulo  $n$ . Computers cannot deal with all integers but just a finite set of them, even using the multiple-precision representation. When two binary strings  $a$  and  $b$  are added together, the adder treats  $n = 2^{m-1}$  as if it were 0! The computer sum  $a \oplus b$  is not necessarily  $a + b$ , but  $a + b$  modulo  $2^{m-1}$ . If  $a + b \geq 2^{m-1}$ , then  $a \oplus b = a + b - 2^{m-1}$ . Since  $2^{m-1} \equiv 0 \pmod{2^{m-1}}$ , we have

$$a + b \equiv a \oplus b \pmod{2^{m-1}}. \quad (3.10)$$

Again in the above example, we have

$$117 + 127 = 244 \equiv 116 \pmod{2^{8-1}}.$$

While the signed-magnitude representation was used in several early computers, modern computers usually use either the one's or two's complement representation, rather than the signed-magnitude representation.

**Definition 3.2.6.** Let  $x$  be a binary number, then the complement of  $x$ , denoted by  $x'$ , is obtained by replacing each 0 in  $x$  by 1 and each 1 in  $x$  by 0. In the one's complement representation, a positive integer is represented as in the signed-magnitude representation, whereas a negative integer is represented by the complement of the corresponding binary number. In the two's complement representation, a positive integer is represented as in the one's representation, but a negative integer is represented by adding one to its one's complement representation.

The range of a number (positive or negative) with  $m$  bits in one's complement representation is given by

$$1 - 2^{m-1}, 1 - 2^{m-1} + 1, \dots, -1, -0, 0, 1, \dots, 2^{m-1} - 1. \quad (3.11)$$

The range of a number with  $m$  bits in two's complement representation is given by

$$-2^{m-1}, -2^{m-1} + 1, \dots, -2, -1, 0, 1, \dots, 2^{m-1} - 1. \quad (3.12)$$

For example, let  $m = 5$ , then the range of a number in one's complement is

$$\begin{array}{ccccccccccc}
1 - 2^{5-1}, & 1 - 2^{5-1} + 1, & \dots, & -1, & -0, & 0, & 1, & \dots, & 2^{5-1} - 1 \\
\downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow \\
-15, & -14, & \dots, & -1, & -0, & 0, & 1, & \dots, & 15
\end{array}$$

and the range of a number in two's complement is

$$\begin{array}{cccccccccccc}
 -2^{5-1}, & 1 - 2^{5-1} + 1, & \dots, & -2, & -1, & 0, & 1, & \dots, & 2^{5-1} - 1 \\
 \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow \\
 -16, & -15, & \dots, & -2, & -1, & 0, & 1, & \dots, & 15
 \end{array}$$

One interesting observation about the two's complement representation is that it has only one representation for zero, whilst there are two zeros in, either the one's complement representation, or the signed-magnitude representation. For example, let  $m = 5$ , then in the one's complement representation, 00000 represents 0 but 11111 represents  $-0$ , whilst in the signed-magnitude representation, 00000 represents 0 but 10000 represents  $-0$ . Table 3.1 gives a comparison of different representations of numbers. By using either the one's complement or two's complement, rather than the signed-magnitude representation, the operation of subtraction is considerably simplified; this is the reason that modern computers use, either the one's complement, or two's complement, not the signed-magnitude representation.

**Example 3.2.10.** Let  $a = 117$  and  $b = -127$ , compute  $a + b$  in the one's complement representation. First note that in the signed-magnitude representation,

$$a = 01110101 \qquad b = 11111111$$

thus in the one's complement representation,

$$a' = 01110101 \qquad b' = 00000000$$

therefore,  $a + b$  becomes

$$\begin{array}{r}
 01110101 \\
 + 00000000 \\
 \hline
 01110101 \implies 10001010 \implies -10.
 \end{array}$$

### 3.2.5 Hash Functions

Hashing is a very important technique in algorithm and database design, as well as in cryptography. In this subsection, we shall introduce an interesting application of number theory in hash function design.

**Definition 3.2.7.** Let  $k$  be the key of the file to be stored, and  $n$  be a positive integer. We define the hash function  $h(k)$  by

$$h(k) \equiv k \pmod{n} \tag{3.13}$$

where  $0 \leq h(k) < n$ , so that  $h(k)$  is the least positive residue of  $k$  modulo  $n$ .

There are two fundamental problems here in the design a good hash function:

**Table 3.1.** Comparison of different representations of numbers

Pure Binary	Binary	Signed-Magnitude	1's Complement	2's Complement
0	00000	0	0	0
1	00001	1	1	1
2	00010	2	2	2
3	00011	3	3	3
4	00100	4	4	4
5	00101	5	5	5
6	00110	6	6	6
7	00111	7	7	7
8	01000	8	8	8
9	01001	9	9	9
10	01010	10	10	10
11	01011	11	11	11
12	01100	12	12	12
13	01101	13	13	13
14	01110	14	14	14
15	01111	15	15	15
16	10000	-0	-15	-16
17	10001	-1	-14	-15
18	10010	-2	-13	-14
19	10011	-3	-12	-13
20	10100	-4	-11	-12
21	10101	-5	-10	-11
22	10110	-6	-9	-10
23	10111	-7	-8	-9
24	11000	-8	-7	-8
25	11001	-9	-6	-7
26	11010	-10	-5	-6
27	11011	-11	-4	-5
28	11100	-12	-3	-4
29	11101	-13	-2	-3
30	11110	-14	-1	-2
31	11111	-15	-0	-1

- (1) How to intelligently choose the value of  $n$ ,
- (2) How to avoid collisions.

The first problem can be solved (at least partially) by selecting  $n$  a prime close to the size of the memory. For example, if the memory size is 5000, we could pick  $n$  to be 4969, a prime close to 5000.

To solve the second problem, we could use the so-called *double hash* technique. The first hash function is the same as (3.13), defined previously, whilst the second hash function is taken as follows:

$$g(k) \equiv k + 1 \pmod{n - 2} \quad (3.14)$$



where  $0 \leq g(k) < n - 1$ , is such that  $\gcd(h(k), n) = 1$ . The probing sequence is defined as follows:

$$h_j(k) \equiv h(k) + j \cdot g(k) \pmod{n} \quad (3.15)$$

where  $0 \leq h_j(k) < n$ . Since  $\gcd(h(k), n) = 1$ , as  $j$  runs through the integers  $1, 2, 3, \dots, n - 1$ , all memory locations will be traced out. Since  $n$  is prime, the ideal selection for the moduli  $n - 2$  would be also prime, that is,  $n$  and  $n - 2$  are twin primes.

**Example 3.2.11.** Suppose we wish to assign memory locations to files with the following index numbers:

$$\begin{array}{ll} k_1 = 197654291 & k_2 = 087365203 \\ k_3 = 528972276 & k_4 = 197354864 \\ k_5 = 873032731 & k_6 = 732975102 \\ k_7 = 216510386 & k_8 = 921001536 \\ k_9 = 933185952 & k_{10} = 109231931 \\ k_{11} = 132489973 & \end{array}$$

We first choose  $n = 5881$ , compute  $h(k_i) = k_i \bmod n$ , and get:

$$\begin{array}{l} h(k_1) = 197654291 \bmod 5881 = 5643 \\ h(k_2) = 087365203 \bmod 5881 = 2948 \\ h(k_3) = 528972276 \bmod 5881 = 5643 \\ h(k_4) = 197354864 \bmod 5881 = 266 \\ h(k_5) = 873032731 \bmod 5881 = 4162 \\ h(k_6) = 732975102 \bmod 5881 = 2548 \\ h(k_7) = 216510386 \bmod 5881 = 1371 \\ h(k_8) = 921001536 \bmod 5881 = 1650 \\ h(k_9) = 933185952 \bmod 5881 = 634 \\ h(k_{10}) = 109231931 \bmod 5881 = 4162 \\ h(k_{11}) = 132489973 \bmod 5881 = 2805 \end{array}$$

Since

$$\begin{array}{l} h(k_1) \equiv h(k_3) \equiv 5643 \pmod{5881}, \\ h(k_5) \equiv h(k_{10}) \equiv 4162 \pmod{5881}. \end{array}$$

we then need to find new locations  $h_1(k_3)$  and  $h_1(k_{10})$  for the 3rd and the 10th record by the formula

$$h_1(k) \equiv h(k) + 1 \cdot g(k) \pmod{n}, \quad \text{with} \quad g(k) \equiv k + 1 \pmod{n - 2}$$

as follows:

$$\begin{array}{l} g(k_3) = 1 + k_3 \bmod 5879 = 1 + 528972276 \bmod 5879 = 3373, \\ g(k_{10}) = 1 + k_{10} \bmod 5879 = 1 + 109231931 \bmod 5879 = 112, \\ h_1(k_3) = h(k_3) + 1 \cdot g(k_3) \bmod 5881 \\ \quad = 528972276 + 1 \cdot 3373 \bmod 5881 = 3222, \\ h_1(k_{10}) = h(k_{10}) + 1 \cdot g(k_{10}) \bmod 5881 \\ \quad = 109231931 + 1 \cdot 112 \bmod 5881 = 4239. \end{array}$$

So finally we have:

Index Number	$h(k)$	$h_1(k)$
197654291	5643	
087365203	2948	
528972276	5643	3222
197354864	266	
873032731	4162	
732975102	2548	
216510386	1371	
921001536	1650	
933185952	634	
109231931	4162	4239
132489973	2805	

Since we can repeatedly compute  $h(k), h_1(k), h_2(k), \dots$ , a suitable location for a record will be eventually found. However, by using the Chinese Remainder Theorem, it is possible to construct a collision-free hash function.

**Definition 3.2.8.** Let  $W = \{w_0, w_1, \dots, w_{m-1}\}$  and  $I = \{0, 1, \dots, (n-1)\}$  be sets with  $n \geq m$ . The hash function  $h : W \rightarrow I$  is called a perfect hash function (PHF), if for all  $x, y \in W$  and  $x \neq y$ ,  $h(x) \neq h(y)$ . In particular, if  $m = n$ ,  $h$  is called a minimal perfect hash function (MPHF). A minimal perfect hash function is also called a minimal collision-free hash function.

The MPHF technique is better than any existing information retrieval method, but the problem is that it is computationally intractable. Recent research shows, however, that we can use the Chinese Remainder Theorem to efficiently construct a MPHF. We describe in the following one such construction, due to Jaeschke [113].

**Theorem 3.2.4.** For a given finite set  $W$  (without loss of generality, we assume that  $W$  is a finite set of positive integers), there exist three constants  $C$ ,  $D$  and  $E$ , such that the function  $h$  defined by

$$h(w) \equiv \lfloor C/(Dw + E) \rfloor \pmod{n-1}, \quad |W| = n-1 \quad (3.16)$$

is a minimal perfect hash function.

The function is clearly a bijection from  $W$  onto the set  $I$ . The proof of this theorem can be done by using a generalization of the Chinese Remainder Theorem (CRT) for non-pairwise (i.e., not necessarily pairwise) relatively prime moduli. First note that for a given set  $W = \{w_0, w_1, \dots, w_{n-1}\}$  of positive integers there exist two integer constants  $D$  and  $E$  such that

$$Dw_0 + E, Dw_1 + E, \dots, Dw_{n-1} + E$$

are pairwise relatively prime, so by the CRT there exists an integer  $C$  such that

$$\left. \begin{array}{l} C \equiv a_0 \pmod{(n-1)(Dw_0 + E)} \\ C \equiv a_1 \pmod{(n-1)(Dw_1 + E)} \\ \dots\dots\dots \\ C \equiv a_{n-1} \pmod{(n-1)(Dw_{n-1} + E)}. \end{array} \right\} \quad (3.17)$$

Finally, we introduce another type of hash function, called one-way hash function, also called message digest or fingerprint.

**Definition 3.2.9.** A one-way hash function maps a string (message)  $m$  of arbitrary length to an integer  $d = H(m)$  with a fixed number of bits, called digest of  $m$ , that satisfies the following conditions:

- [1] Given  $m$ ,  $d$  is easy to compute.
- [2] Given  $d$ ,  $m$  is computationally infeasible to find.

A one-way hash function is said to be *collision resistant* if it is computationally infeasible to find two strings  $m_1$  and  $m_2$  that have the same digest  $d$ .

Several one-way hash functions believed to be collision resistant; the ones used most in practice are MD5, which produces a 128-bit digest, and SHA-1, which produces a 160-bit digest (MD stands for message digest and SHA stands for secure hash algorithm). The most important application of one-way collision resistant hash functions is to speed up the construction of digital signatures (we shall discuss digital signatures later), since we can sign the digest of the message,  $d = H(m)$ , rather than the message itself,  $m$ . That is,

$$S = D(H(m)), \quad (3.18)$$

where  $D$  is the digital signature algorithm.

### 3.2.6 Error Detection and Correction Methods

In this subsection, we shall discuss an interesting application of the theory of congruences in error detections and corrections.

It is evident that manipulating and transmitting bit strings can introduce errors. A simple error detection method, called *parity check* works in the following way (suppose the bit string to be sent is  $x_1x_2 \cdots x_n$ ):

- [1] (Precomputation) Append to the bit string a *parity check bit*  $x_{n+1}$

$$x_{n+1} \equiv x_1 + x_2 + \cdots + x_n \pmod{2}, \quad (3.19)$$

so that

$$x_{n+1} = \begin{cases} 0, & \text{if there is an even number of} \\ & 1 \text{ in } x_1 x_2 \cdots x_n, \\ 1, & \text{otherwise.} \end{cases} \quad (3.20)$$

The appended string  $x_1 x_2 \cdots x_n x_{n+1}$  should satisfy the following congruence

$$x_1 + x_2 + \cdots + x_n + x_{n+1} \equiv 0 \pmod{2}. \quad (3.21)$$

[2] (Error Detection) Suppose now we send the string  $x = x_1 x_2 \cdots x_n x_{n+1}$  and the string  $y = y_1 y_2 \cdots y_n y_{n+1}$  is received. If  $x = y$ , then there are no errors, but if  $x \neq y$ , there will be errors. We check whether or not

$$y_1 + y_2 + \cdots + y_n + y_{n+1} \equiv 0 \pmod{2} \quad (3.22)$$

holds. If this congruence fails, at least one error is present; but if it holds, errors may still exist. Clearly, we can detect an odd number of errors, but not an even number of errors.

The above method can be easily extended to checking for errors in strings of *digits*, rather than just bits. The use of check digits with identification numbers for error detection is now a standard practice. Notable examples include social security numbers, telephone numbers, serial numbers on currency predate computers, Universal Product Codes (UPC) on grocery items, and International Standard Book Numbers (ISBN) on published books. In what follows, we shall introduce a modulus 11 error correction and detection scheme for ISBN numbers.

Every recently published book has a 10-digit codeword called its International Standard Book Number (ISBN). This is a sequence of nine digits  $x_1 x_2 \cdots x_9$ , where each  $x_i \in \{0, 1, 2, \dots, 9\}$ , together with a check digit  $x_{10} \in \{0, 1, 2, \dots, 9, X\}$  (we use the single letter  $X$  to represent the two digit number 10). This last digit  $x_{10}$  is included so as to give a check that the previous nine digits have been correctly transcribed;  $x_{10}$  can be obtained by

$$x_{10} = \sum_{i=1}^9 i x_i \pmod{11}. \quad (3.23)$$

Note that if we arrange the ten digit ISBN number in the order of  $x_{10} x_9 \cdots x_2 x_1$ , then the check digit  $x_1$  is determined by

$$x_1 = 11 - \sum_{i=10}^2 i x_i \pmod{11}. \quad (3.24)$$

The whole 10-digit number satisfies the following so-called *check congruence*

$$\sum_{i=1}^{10} i x_i \equiv 0 \pmod{11}. \quad (3.25)$$

**Example 3.2.12.** The first nine digits of the ISBN number of the book by Ireland and Rosen [111] are as follows:

0-387-97329.

Find the check digit of this ISBN number. We first let

$$\begin{array}{cccccccc} 0 & 3 & 8 & 7 & 9 & 7 & 3 & 2 & 9 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \end{array}$$

Then

$$\begin{aligned} x_{10} &\equiv \sum_{i=1}^9 ix_i \pmod{11} \\ &\equiv [1 \cdot 0 + 2 \cdot 3 + 3 \cdot 8 + 4 \cdot 7 + 5 \cdot 9 + \\ &\quad 6 \cdot 7 + 7 \cdot 3 + 8 \cdot 2 + 9 \cdot 9] \pmod{11} \\ &= 10 = X \end{aligned}$$

If we let

$$\begin{array}{cccccccc} 0 & 3 & 8 & 7 & 9 & 7 & 3 & 2 & 9 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ x_{10} & x_9 & x_8 & x_7 & x_6 & x_5 & x_4 & x_3 & x_2 \end{array}$$

Then

$$\begin{aligned} x_1 &\equiv 11 - \sum_{i=10}^2 ix_i \pmod{11} \\ &\equiv 11 - [10 \cdot 0 + 9 \cdot 3 + 8 \cdot 8 + 7 \cdot 7 + \\ &\quad 6 \cdot 9 + 5 \cdot 7 + 4 \cdot 3 + 3 \cdot 2 + 2 \cdot 9] \pmod{11} \\ &= 10 = X \end{aligned}$$

So the complete ISBN number of the book is

0-387-97329-X.

Generally speaking, the coefficients  $a_i$ , for  $i = 1, 2, \dots, n$  (or  $i = n, n-1, \dots, 1$ ) could be any numbers as long as the  $n$  digits satisfies the check congruence:

$$a_1x_1 + a_2x_2 + \dots + a_nx_n \equiv 0 \pmod{m}.$$

**Example 3.2.13.** The ISBN number of the present book is

3-540-65472-0

and it satisfies its check congruence

$$\sum_{i=1}^{10} ix_i \equiv [1 \cdot 3 + 2 \cdot 5 + 3 \cdot 4 + 4 \cdot 0 + 5 \cdot 6 + 6 \cdot 5 + 7 \cdot 4 + 8 \cdot 7 + 9 \cdot 2 + 10 \cdot 0] \pmod{11}.$$

Suppose the first nine digits of the ISBN number are given and we are asked to find the check digit  $x_{10}$ , then we have

$$x_{10} = \sum_{i=1}^9 ix_i = [1 \cdot 3 + 2 \cdot 5 + 3 \cdot 4 + 4 \cdot 0 + 5 \cdot 6 + 6 \cdot 5 + 7 \cdot 4 + 8 \cdot 7 + 9 \cdot 2] \pmod{11} = 0.$$

**Example 3.2.14.** Suppose a book whose ISBN number is as follows

9-810- $x$ 3422-8

where  $x$  is an unknown digit. What is  $x$ ? To find the value for  $x$ , we perform the following computation:

$$[1 \cdot 9 + 2 \cdot 8 + 3 \cdot 1 + 4 \cdot 0 + 5x + 6 \cdot 3 + 7 \cdot 4 + 8 \cdot 2 + 9 \cdot 2 + 10 \cdot 8] \pmod{11} = 1.$$

So, we have

$$1 + 5x \equiv 0 \pmod{11}.$$

To solve this linear congruence, we get

$$\begin{aligned} x &\equiv -\frac{1}{5} \pmod{11} \\ &\equiv -9 \pmod{11} \quad \left( \text{since } \frac{1}{5} \equiv 9 \pmod{11} \right) \\ &\equiv 2 \pmod{11}. \end{aligned}$$

Thus,  $x = 2$ .

**Exercise 3.2.1.** Find the value of  $x$  in each of the following ISBN numbers:

0-201-07981- $x$ ,  
0-8053- $x$ 340-2,  
0-19-8 $x$ 3171-0.

The ISBN code can detect

- (1) 100% of all single digit errors,
- (2) 100% of double errors created by the transposition of two digits.

The detection process is as follows. Let  $\mathbf{x} = x_1x_2 \cdots x_{10}$  be the original codeword sent,  $\mathbf{y} = y_1y_2 \cdots y_{10}$  the received string, and  $S = \sum_{i=1}^{10} iy_i$ . If  $S \equiv 0 \pmod{11}$ , then  $\mathbf{y}$  is the legitimate codeword and we assume it is correct, whereas if  $S \not\equiv 0 \pmod{11}$ , then we have detected error(s):

- (1) Suppose the received string  $\mathbf{y} = y_1 y_2 \cdots y_{10}$  is the same as  $\mathbf{x} = x_1 x_2 \cdots x_{10}$  except that the  $y_k = x_k + a$  with  $1 \leq k \leq 10$  and  $a \neq 0$ . Then

$$S = \sum_{i=1}^{10} i y_i = \sum_{i=1}^{10} i x_i + k a \not\equiv 0 \pmod{11},$$

since  $k$  and  $a$  are all non-zero elements in  $\mathbb{Z}/11\mathbb{Z}$ .

- (2) Suppose the received string  $\mathbf{y} = y_1 y_2 \cdots y_{10}$  is the same as  $\mathbf{x} = x_1 x_2 \cdots x_{10}$  except that  $y_j$  and  $x_k$  have been transposed. Then

$$\begin{aligned} S &= \sum_{i=1}^{10} i y_i = \sum_{i=1}^{10} i x_i + (k-j)x_j + (j-k)x_k \\ &= (k-j)(x_j - x_k) \not\equiv 0 \pmod{11}, \quad \text{if } k \neq j \text{ and } x_j \neq x_k. \end{aligned}$$

Note that since  $\mathbb{Z}/11\mathbb{Z}$  is a field, the product of two non-zero elements is also non-zero but this does not hold in  $\mathbb{Z}/10\mathbb{Z}$ , which is only a ring (say, for example,  $2 \cdot 5 \equiv 0 \pmod{10}$ ); this is why we work with modulo 11 rather than modulo 10. Note also that the ISBN code cannot be used to correct errors unless we know that just one digit is in error. Interested readers are suggested to consult Gallian [77] and Hill [104] for more information about error detection and correction codes.

We now move to the introduction of another interesting error detection technique for programs (Brent [38]). The Galileo spacecraft is somewhere near Jupiter, but its main radio antenna is not working, so communication with it is slow. Suppose we want to check that a critical program in Galileo's memory is correct. How can we do this without transmitting the whole program from/to Galileo? The following is a method (possibly the simplest method) for checking out Galileo's program based on some simple number-theoretic ideas; the method was first proposed by Michael Rabin:

Let  $P_g$  be the program in Galileo and  $P_e$  the program on Earth, each represented as an integer. Assuming  $P_e$  is correct, this algorithm will try to determine whether or not  $P_g$  is correct:

- [1] Choose a prime number  $10^9 < p < 2 \cdot 10^9$  and transmit  $p$  ( $p$  has no more than 32 bits) to Galileo and ask it to compute  $r_g \leftarrow P_g \bmod p$  and send the remainder  $r_g$  back to Earth ( $r_g$  has no more than 32 bits).
- [2] On Earth, we compute  $r_e \leftarrow P_e \bmod p$ , and check if  $r_g = r_e$ .
- [3] If  $r_g \neq r_e$ , we conclude that  $P_g \neq P_e$ . That is, Galileo's program has been corrupted!
- [4] If  $r_g = r_e$ , we conclude that  $P_g$  is *probably* correct. That is, if  $P_g$  is not correct, there is only a small probability of  $< 10^{-9}$  that  $r_g = r_e$ . If this error probability is too large to accept for the quality-assurance team, just goto step [1] to start the process all over again, else terminate the algorithm by saying that  $P_g$  is "almost surely" correct! It is clear that if we repeat the

process, for example, ten times on ten different random primes, then the error probability will be less than  $10^{-90}$ , an extremely small number.

Clearly the idea underlying the method for program testing is exactly the same as that of the probabilistic method for primality testing.

### 3.2.7 Random Number Generation

*Anyone who considers arithmetic methods of producing random digits is, of course, in a state of sin.*

JOHN VON NEUMANN (1903–1957)

“Random” numbers have a great many uses in, e.g., numerical simulations, sampling, numerical analysis, testing computer chips for defects, decision making, coding and cryptography, and programming slot machines, etc. They are a valuable resource: in some cases, they can speed up computations, they can improve the rate of communication of partial information between two users, and they can also be used to solve problems in asynchronous distributed computation that is impossible to solve by deterministic means. A *sequence of numbers* is random if each number in the sequence is independent of the preceding numbers; there are no patterns to help us to predict any number of the sequence. Of course, truly *random* numbers are hard to come by, or even impossible to get. Thus, the so-called random numbers are actually *pseudorandom numbers*. Since the invention of the first electronic computer, researchers have been trying to find *efficient* ways to generate random numbers on a computer. We have, in fact, already seen some applications of random numbers in this book; for example, Pollard’s  $\rho$ -method, introduced in Chapter 2, uses random numbers in finding prime factorization of large integers. In this subsection, we shall briefly introduce some methods for generating random numbers based on linear congruences.

Firstly, let us introduce an arithmetic method, called the *middle-square method*, suggested by John von Neumann<sup>2</sup> in 1946. The algorithmic description of the method is as follows:

<sup>2</sup>



John von Neumann (1903–1957) was born in Budapest, Hungary, but lived in the U.S.A. from 1930 onwards. He is one of the legendary figures of 20th century mathematics. He made important contributions to logic, quantum physics, optimization theory and game theory. His lifelong interest in mechanical devices led to his being involved crucially in the initial development of the modern electronic computer and the important concept of the *stored program*. He was also involved in the development of the first atomic bomb.



**Algorithm 3.2.2 (Von Neumann's middle-square method).** This algorithm uses the so-called middle-square method to generate random numbers:

- [1] Let  $m$  be the number of random numbers we wish to generate (all with, for example, 10 digits), and set  $i \leftarrow 0$ .
- [2] Randomly choose a starting 10-digit number  $n_0$ .
- [3] Square  $n_i$  to get an intermediate number  $M$ , with 20 or less digits.
- [4] Set  $i = i + 1$  and take the middle ten digits of  $M$  as the new random number  $n_i$ .
- [5] If  $i < m$  then goto step [3] to generate a new random number, else stop the generating process.

**Example 3.2.15.** Let  $n_0 = 9524101765$ , and  $m = 10$ . Then by Algorithm 3.2.2 we have

$$\begin{aligned}
 9524101765^2 &= 90708514430076115225 \implies n_1 = 5144300761 \\
 5144300761^2 &= 26463830319625179121 \implies n_2 = 8303196251 \\
 8303196251^2 &= 68943067982620455001 \implies n_3 = 0679826204 \\
 0679826204^2 &= 462163667645049616 \implies n_4 = 6366764504 \\
 6366764504^2 &= 40535690249394366016 \implies n_5 = 6902493943 \\
 6902493943^2 &= 47644422633151687249 \implies n_6 = 4226331516 \\
 4226331516^2 &= 17861878083134858256 \implies n_7 = 8780831348 \\
 8780831348^2 &= 77102999162019497104 \implies n_8 = 9991620194 \\
 9991620194^2 &= 99832474101148597636 \implies n_9 = 4741011485 \\
 4741011485^2 &= 22477189900901905225 \implies n_{10} = 1899009019.
 \end{aligned}$$

A serious problem with the middle-square method is that for many choices of the initial integer, the method produces the same small set of numbers over and over. For example, working with numbers that have four digits, starting from 4100, we obtain the sequence

$$8100, 6100, 2100, 4100, 8100, 6100, 2100, \dots$$

In what follows, we shall introduce some methods based on congruence theory, which can generate a sequence of numbers that appear to be *essentially* random.

Congruence theory is useful in generating a list of random numbers. At present, the most popular random number generators in use are special cases of the so-called *linear congruential generator* (LCG for short), introduced first by D. H. Lehmer in 1949. In the linear congruential method, we first choose four “magic” numbers as follows:

$$\begin{array}{lll}
 n: & \text{the modulus;} & n > 0 \\
 x_0: & \text{the seed;} & 0 \leq x_0 \leq n \\
 a: & \text{the multiplier;} & 0 \leq a \leq n \\
 b: & \text{the increment;} & 0 \leq b \leq n
 \end{array}$$

then the sequence of random numbers is defined recursively by:

$$x_j \equiv ax_{j-1} + b \pmod{n}, \quad j > 0, \quad (3.26)$$

for  $1 \leq j \leq l$ , where  $l \in \mathbb{N}$  is the least value such that  $x_{l+1} \equiv x_j \pmod{n}$  for some  $j \leq l$ . We call  $l$  the period length of the LCG generator. Clearly, the maximum length of distinct random numbers generated by the LCG is the modulus  $n$ . The best random number generator is, of course, the one that has the maximum length of distinct random numbers. Knuth gives a necessary and sufficient condition for a LCG to have maximum length:

**Theorem 3.2.5 (Knuth [123]).** A LCG has period length  $l = n$  if and only if  $\gcd(b, n) = 1$ ,  $a \equiv 1 \pmod{p}$  for all primes  $p \mid n$  and  $a \equiv 1 \pmod{4}$  if  $4 \mid n$ .

Note that the parameter  $a$  is sometimes set to be 1; in that case, the LCG is just a “plain” linear congruential generator. When  $a$  is set to be greater than 1, it is sometimes called a multiplicative linear congruential generator. Now we are in a position to give an algorithm for a LCG.

**Algorithm 3.2.3 (Linear Congruential Generator).** This algorithm will generate a sequence of random numbers  $\{x_1, x_2, \dots, x_n\}$ .

- [1] [Initialization] Input  $x_0, a, b, n$  and  $k$  (here  $k$  is just the number of random numbers the user wishes to generate; we can simply set  $k = n$ ). Set  $j \leftarrow 1$ .
- [2] [Random Number Generation] Compute  $x_j \leftarrow (ax_{j-1} + b) \pmod{n}$ , and print  $x_j$ .
- [3] [Increase  $j$ ]  $j \leftarrow j + 1$ . If  $j \geq k$ , then goto Step [4], else goto Step [2].
- [4] [Exit] Terminate the algorithm.

**Example 3.2.16.** Let  $x_0 = 5$ ,  $a = 11$ ,  $b = 73$ ,  $n = 1399$  and  $k = 10$ . Then by Algorithm 3.2.3 we have:

$$\begin{array}{llll}
 x_0 & = & 5 & \\
 x_1 & \equiv & ax_0 + b \pmod{n} & \implies x_1 = 128 \\
 x_2 & \equiv & ax_1 + b \pmod{n} & \implies x_2 = 82 \\
 x_3 & \equiv & ax_2 + b \pmod{n} & \implies x_3 = 975 \\
 x_4 & \equiv & ax_3 + b \pmod{n} & \implies x_4 = 1005 \\
 x_5 & \equiv & ax_4 + b \pmod{n} & \implies x_5 = 1335 \\
 x_6 & \equiv & ax_5 + b \pmod{n} & \implies x_6 = 768 \\
 x_7 & \equiv & ax_6 + b \pmod{n} & \implies x_7 = 127 \\
 x_8 & \equiv & ax_7 + b \pmod{n} & \implies x_8 = 71 \\
 x_9 & \equiv & ax_8 + b \pmod{n} & \implies x_9 = 854 \\
 x_{10} & \equiv & ax_9 + b \pmod{n} & \implies x_{10} = 1073 \\
 & \dots & & \\
 & \dots & & \\
 x_{231} & \equiv & ax_{230} + b \pmod{n} & \implies x_{231} = 1149
 \end{array}$$

$$\begin{aligned}
x_{232} &\equiv ax_{231} + b \pmod{n} &\implies x_{232} &= 121 \\
x_{233} &\equiv ax_{232} + b \pmod{n} &\implies x_{233} &= 5 \\
x_{234} &\equiv ax_{233} + b \pmod{n} &\implies x_{234} &= 128.
\end{aligned}$$

So the length of this random number sequence

$$\begin{aligned}
&(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, \dots, x_{231}, x_{232}, x_{233}) \\
&= (128, 82, 975, 1005, 1335, 768, 127, 71, 854, 1073, \dots, 1149, 121, 5)
\end{aligned}$$

generated by the LCG

$$\begin{aligned}
x_0 &\equiv 5 \pmod{1399}, \\
x_j &\equiv 11 \cdot x_{j-1} + 73 \pmod{1399}, \quad j = 1, 2, \dots,
\end{aligned}$$

is 233, i.e.,  $l = 233$ .

Normally, we could set  $n = 2^r$ ,  $a = 2^i + 1$  with  $i < r$ , and  $b = 1$ . Thus, Equation (3.26) becomes

$$x_j \equiv (2^k + 1)x_{j-1} + 1 \pmod{2^r}, \quad j = 1, 2, \dots \quad (3.27)$$

To make a LCG a good random number generator, it is necessary to find good values for all the four magic numbers (not just the modulus  $n$ ) that define the linear congruential sequence. Interested readers are invited to consult [123] for a thorough discussion about the choice of the parameters. There are many congruential generators based on the linear congruential generator:

(1) Power generator:

$$x_j \equiv (x_{j-1})^d \pmod{n}, \quad j = 1, 2, \dots \quad (3.28)$$

where  $(d, n)$  are parameters describing the generator and  $x_0$  is the seed. There are two important special cases of the power generator, both occurring when  $n = pq$  is a product of two distinct odd primes.

- (i) The RSA<sup>3</sup> Generator: This case occurs when  $\gcd(d, \phi(n)) = 1$ , where  $\phi(n)$  is Euler's  $\phi$ -function. The map  $x \mapsto x^d \pmod{n}$  is one-to-one on  $(\mathbb{Z}/n\mathbb{Z})^*$ , and this operation is the encryption operation of the RSA public-key cryptosystem, where the pair  $(d, n)$  is publicly known. This special case of the power generator is called the *RSA generator*. For example, let  $p = 13$ ,  $q = 23$  and  $d = 17$ , so that  $n = 299$ ,  $\phi(299) = 264$  and  $\gcd(299, 17) = 1$ . Let also  $x_0 = 6$ . Then by the RSA generator

$$\begin{aligned}
x_0 &= 6, \\
x_j &= x_{j-1}^{17} \pmod{299}, \quad j = 1, 2, \dots,
\end{aligned}$$

---

<sup>3</sup> RSA stands for three computer scientists Rivest, Shamir and Adleman [209], who invented the widely used RSA public-key cryptosystem in the 1970s, which will be studied in the next section. The RSA generator has essentially the same idea as the RSA cryptosystem.

we have the following random sequence:

$$\begin{aligned}
x_1 &\equiv x_0^{17} \pmod{299} \implies x_1 \equiv 6^{17} \equiv 288 \pmod{299} \\
x_2 &\equiv x_1^{17} \pmod{299} \implies x_2 \equiv 288^{17} \equiv 32 \pmod{299} \\
x_3 &\equiv x_2^{17} \pmod{299} \implies x_3 \equiv 32^{17} \equiv 210 \pmod{299} \\
x_4 &\equiv x_3^{17} \pmod{299} \implies x_4 \equiv 210^{17} \equiv 292 \pmod{299} \\
x_5 &\equiv x_4^{17} \pmod{299} \implies x_5 \equiv 292^{17} \equiv 119 \pmod{299} \\
x_6 &\equiv x_5^{17} \pmod{299} \implies x_6 \equiv 119^{17} \equiv 71 \pmod{299} \\
x_7 &\equiv x_6^{17} \pmod{299} \implies x_7 \equiv 71^{17} \equiv 41 \pmod{299} \\
x_8 &\equiv x_7^{17} \pmod{299} \implies x_8 \equiv 41^{17} \equiv 123 \pmod{299} \\
x_9 &\equiv x_8^{17} \pmod{299} \implies x_9 \equiv 123^{17} \equiv 197 \pmod{299} \\
x_{10} &\equiv x_9^{17} \pmod{299} \implies x_{10} \equiv 197^{17} \equiv 6 \pmod{299} \\
x_{11} &\equiv x_{10}^{17} \pmod{299} \implies x_{11} \equiv 6^{17} \equiv 288 \pmod{299}.
\end{aligned}$$

Thus, the length of this random number sequence generated by the RSA generator is 10. That is  $l = 10$ .

- (ii) The square generator: This case occurs when  $d = 2$  and  $n = pq$  with  $p \equiv q \equiv 3 \pmod{4}$ ; we call this the *square generator*. In this case, the mapping  $x_i \mapsto (x_{i-1})^2 \pmod{n}$  is four-to-one on  $(\mathbb{Z}/n\mathbb{Z})^*$ . An even more special case of the square generator is the *quadratic residues generator*:

$$y \equiv x^2 \pmod{n} \quad (3.29)$$

for some  $x$ .

- (2) Discrete exponential generator:

$$x_j \equiv g^{x_{j-1}} \pmod{n}, \quad j = 1, 2, \dots \quad (3.30)$$

where  $(g, n)$  are parameters describing the generator and  $x_0$  the seed. A special case of the discrete exponential generator is that when  $n$  is an odd prime  $p$ , and  $g$  is a primitive root modulo  $p$ ; then the problem of recovering  $x_{j-1}$  given by  $(x_j, g, n)$  is the well-known hard *discrete logarithm problem*.

Note that simpler sequences of random numbers can be combined to produce complicated ones by using hashg and composition functions. For more information on this topic, see Lagarias [136] and the references therein.

In some cases, for example, in stream-cipher cryptography (Zeng [265]), a stream of random bits rather than a sequence of random digits (numbers) will be needed. We list in the following some of the widely used random bit generators (more random bit generators can be found, for example, in Lagarias [136]):

- (1) RSA bit generator: Given  $k \geq 2$  and  $m \geq 1$ , select odd primes  $p$  and  $q$  uniformly from the range  $2^k \leq p, q < 2^{k+1}$  and form  $n = pq$ . Select  $e$  uniformly from  $[1, n]$  subject to  $\gcd(e, \phi(n)) = 1$ . Set

$$x_j \equiv (x_{j-1})^e \pmod{n}, \quad j = 1, 2, \dots \quad (3.31)$$

and let the bit  $z_j$  be given by

$$z_j \equiv x_j \pmod{2}, \quad j = 1, 2, \dots \quad (3.32)$$

Then  $\{z_j : 1 \leq j \leq k^m + m\}$  are the random bits generated by the seed  $x_0$  of the length  $2k$  bits.

- (2) Rabin's modified bit generator: Let  $k \geq 2$ , and select odd primes  $p$  and  $q$  uniformly from primes in the range  $2^k \leq p, q < 2^{k+1}$  and form  $n = pq$ , such that  $p \equiv q \equiv 3 \pmod{4}$  (this assumption is used to guarantee that  $-1$  is a quadratic nonresidue for both  $p$  and  $q$ ). Let

$$x_j = \begin{cases} (x_{j-1})^2 \pmod{n}, & \text{if it lies in } [0, n/2), \\ n - (x_{j-1})^2 \pmod{n}, & \text{otherwise,} \end{cases} \quad (3.33)$$

so that  $0 \leq x_j < n/2$ , and the bit  $z_j$  be given by

$$z_j \equiv x_j \pmod{2}, \quad j = 1, 2, \dots \quad (3.34)$$

Then  $\{z_j : 1 \leq j \leq k^m + m\}$  are the random bits generated by the seed  $x_0$  of the length  $2k$  bits.

- (3) Discrete exponential bit generator Let  $k \geq 2$  and  $m \geq 1$ , and select an odd prime  $p$  uniformly from primes in the range  $[2^k, 2^{k+1}]$ , provided with a complete factorization of  $p-1$  and a primitive root  $g$ . Set

$$x_j \equiv g^{x_{j-1}} \pmod{p}, \quad j = 1, 2, \dots \quad (3.35)$$

and let the bit  $z_j$  be the most significant bit

$$z_j \equiv \left\lceil \frac{x_j}{2^k} \right\rceil \pmod{2}. \quad (3.36)$$

Then  $\{z_j : 1 \leq j \leq k^m + m\}$  are the random bits generated by the seed  $x_0$ .

- (4) Elliptic curve bit generator: Elliptic curves, as we have already seen, have applications in primality testing and integer factorization. It is interesting to note that elliptic curves can also be used to generate random bits; interested readers are referred to Kaliski [116] for more information.

### 3.3 Cryptography and Information Security

*Modern cryptography depends heavily on number theory, with primality testing, factoring, discrete logarithms (indices), and elliptic curves being perhaps the most prominent subject areas.*

MARTIN HELLMAN

*Foreword to the present book*

Cryptography was concerned initially with providing secrecy for written messages. Its principles apply equally well to securing data flow between computers, to digitized speech, and to encrypting facsimile and television signals. For example, most satellites routinely encrypt the data flow to and from ground stations to provide both privacy and security for their subscribers. In this section, we shall introduce some basic concepts and techniques of cryptography and discuss their applications to computer-based information security.

#### 3.3.1 Introduction

Cryptography (from the Greek *Kryptós*, “hidden”, and *gráphein*, “to write”) is the study of the principles and techniques by which information can be concealed in ciphertexts and later revealed by legitimate users employing the secret key, but in which it is either impossible or computationally infeasible for an unauthorized person to do so. Cryptanalysis (from the Greek *Kryptós* and *analýein*, “to loosen”) is the science (and art) of recovering information from ciphertexts without knowledge of the key. Both terms are subordinate to the more general term *cryptology* (from the Greek *Kryptós* and *lógos*, “word”). That is,

$$\text{Cryptology} \stackrel{\text{def}}{=} \text{Cryptography} + \text{Cryptanalysis},$$

and

$$\text{Cryptography} \stackrel{\text{def}}{=} \text{Encryption} + \text{Decryption}.$$

Modern cryptography, however, is the study of “mathematical” systems for solving the following two main types of security problems:

- (1) privacy,
- (2) authentication.

A privacy system prevents the extraction of information by unauthorized parties from messages transmitted over a public and often insecure channel, thus assuring the sender of a message that it will only be read by the

intended receiver. An authentication system prevents the unauthorized injection of messages into a public channel, assuring the receiver of a message of the legitimacy of its sender. It is interesting to note that the computational engine, designed and built by a British group led by Alan Turing at Bletchley Park, Milton Keynes to crack the German ENIGMA code is considered to be among the very first real electronic computers; thus one could argue that modern cryptography is the mother (or at least the midwife) of modern computer science.

There are essentially two different types of cryptographic systems (cryptosystems):

- (1) *Secret-key* cryptographic systems (also called symmetric cryptosystems),
- (2) *Public-key* cryptographic systems (also called asymmetric cryptosystems).

Before discussing these two types of different cryptosystems, we present some notation:

- (1) *Message space*  $\mathcal{M}$ : a set of strings (plaintext messages) over some alphabet, that needs to be encrypted.
- (2) *Ciphertext space*  $\mathcal{C}$ : a set of strings (ciphertexts) over some alphabet, that has been encrypted.
- (3) *Key space*  $\mathcal{K}$ : a set of strings (keys) over some alphabet, which includes
  - (i) The *encryption key*  $e_k$ .
  - (ii) The *decryption key*  $d_k$ .
- (4) The *encryption process (algorithm)*  $E$ :  $E_{e_k}(M) = C$ .
- (5) The *decryption process (algorithm)*  $D$ :  $D_{d_k}(C) = M$ .

The algorithms  $E$  and  $D$  must have the property that

$$D_{d_k}(C) = D_{d_k}(E_{e_k}(M)) = M.$$

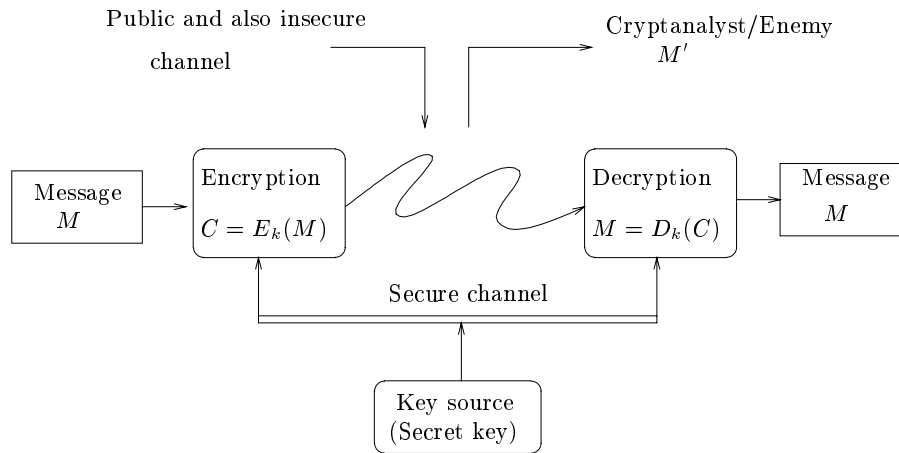
### 3.3.2 Secret-Key Cryptography

*The legend that every cipher is breakable is of course absurd, though still widespread among people who should know better.*

J. E. LITTLEWOOD

Mathematics with Minimum 'Raw Material' [144]

In a conventional secret-key cryptosystem (see Figure 3.3), the same key (i.e.,  $e_k = d_k = k \in \mathcal{K}$ ), called the *secret key*, is used in both encryption and decryption. By same key we mean that someone who has enough information to encrypt messages automatically has enough information to decrypt messages as well. This is why we call it secret-key cryptosystem, or symmetric



**Figure 3.3.** Conventional secret-key cryptosystems

cryptosystem. The sender uses an invertible transformation  $f$  defined by

$$f : \mathcal{M} \xrightarrow{k} \mathcal{C}, \quad (3.37)$$

to produce the cipher text

$$C = E_k(M), \quad M \in \mathcal{M} \text{ and } C \in \mathcal{C}, \quad (3.38)$$

and transmits it over the public *insecure* channel to the receiver. The key  $k$  should also be transmitted to the legitimate receiver for decryption but via a *secure* channel. Since the legitimate receiver knows the key  $k$ , he can decrypt  $C$  by a transformation  $f^{-1}$  defined by

$$f^{-1} : \mathcal{C} \xrightarrow{k} \mathcal{M}, \quad (3.39)$$

and obtain

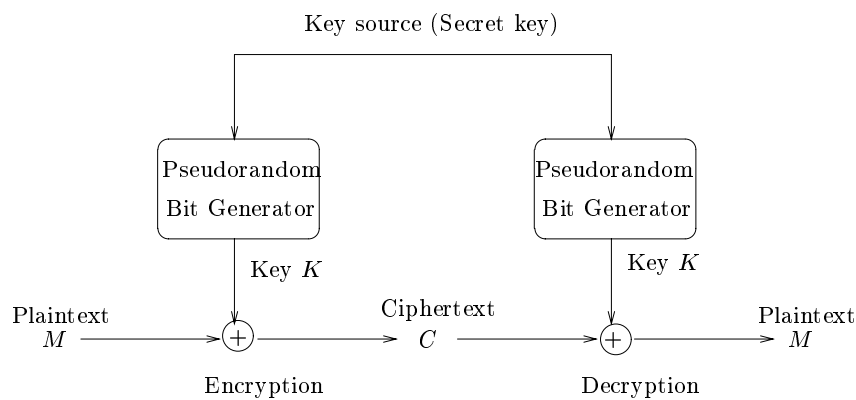
$$D_k(C) = D_k(E_k(M)) = M, \quad C \in \mathcal{C} \text{ and } M \in \mathcal{M}, \quad (3.40)$$

the original plain-text message. There are many different types of secret-key cryptographic systems. In what follows, we shall introduce some of these systems. (Note that the terms cryptographic systems, cryptographic schemes, or ciphers are essentially the same concepts, and we shall use them interchangeably in this chapter.)

**(I) Stream (Bit) Ciphers.** In stream ciphers, the message units are bits, and the key is usually produced by a random bit generator (see Figure 3.4). The plaintext is encrypted on a bit-by-bit basis:

$M$	0	1	1	0	0	0	1	1	1	1	1	1	0	1	0	1	0	...	
$K$	1	0	0	1	1	0	0	1	0	0	0	1	0	1	1	1	0	1	...
$C$	1	1	1	1	1	0	1	0	1	1	1	0	1	1	0	1	1	1	...





**Figure 3.4.** A stream cipher

The key is fed into the random bit generator to create a long sequence of binary signals. This “key-stream”  $K$  is then mixed with the plaintext stream  $M$ , usually by a bit-wise XOR (Exclusive-OR, or modulo-2 addition) to produce the ciphertext stream  $C$ . The decryption is done by XORing with the same key stream, using the same random bit generator and seed:

$C$	1	1	1	1	1	0	1	0	1	1	1	0	1	1	0	1	1	1	...
$K$	1	0	0	1	1	0	0	1	0	0	0	1	0	1	1	1	0	1	...
$M$	0	1	1	0	0	0	1	1	1	1	1	1	1	0	1	0	1	0	...

**(II) Monographic (Character) Ciphers.** Earlier ciphers (cryptosystems) were based on transforming each letter of the plaintext into a different letter to produce the ciphertext. Such ciphers are called *character*, *substitution* or *monographic ciphers*, since each letter is shifted individually to another letter by a substitution. First of all, let us define the numerical equivalents, as in Table 3.2, of the 26 English capital letters, since our operations will be on

**Table 3.2.** Numerical equivalents of English capital letters

A	B	C	D	E	F	G	H	I	J	K	L	M
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
13	14	15	16	17	18	19	20	21	22	23	24	25

the numerical equivalents of letters, rather than the letters themselves. The following are some typical character ciphers.

- (1) Caesar<sup>4</sup> cipher: A simple *Caesar cipher* uses the following substitution transformation:

$$f_3 = E_3(m) \equiv m + 3 \pmod{26}, \quad 0 \leq m \in \mathcal{M} \leq 25, \quad (3.41)$$

and

$$f_3^{-1} = D_3(c) \equiv c - 3 \pmod{26}, \quad 0 \leq c \in \mathcal{C} \leq 25, \quad (3.42)$$

where 3 is the key for both encryption and decryption. Clearly, the corresponding letters of the Caesar cipher will be obtained from those in Table 3.2 by moving three letters forward, as described in Table 3.3. Mathematically, in encryption we just perform a mapping  $m \mapsto m + 3 \pmod{26}$  on the plaintext, whereas in decryption a mapping  $c \mapsto c - 3 \pmod{26}$  on the ciphertext.

**Table 3.3.** The corresponding letters of the Caesar cipher

$\mathcal{M}$	A	B	C	D	E	F	G	H	I	J	K	L	M
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Shift	3	4	5	6	7	8	9	10	11	12	13	14	15
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
$\mathcal{C}$	D	E	F	G	H	I	J	K	L	M	N	O	P
$\mathcal{M}$	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Shift	16	17	18	19	20	21	22	23	24	25	0	1	2
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
$\mathcal{C}$	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

<sup>4</sup>



Julius Caesar (100–44 BC) was a celebrated Roman general, statesman, orator and reformer. The Caesar cipher, involving in replacing each letter of the alphabet with the letter standing three places further down the alphabet, was apparently used by Caesar (he used the cipher in both his domestic and military efforts), but he was also supposed to have invented the cipher himself. Although the Caesar cipher was a simple cipher and particularly simple to crack, it is a useful vehicle for explaining cryptographic principles. (Photo by courtesy of Dr. Singh.)

- (2) Shift transformations: Slightly more general transformations are the following so-called *shift transformations*:

$$f_k = E_k(m) \equiv m + k \pmod{26}, \quad 0 \leq k, m \leq 25, \quad (3.43)$$

and

$$f_k^{-1} = D_k(c) \equiv c - k \pmod{26}, \quad 0 \leq k, c \leq 25, \quad (3.44)$$

- (3) Affine transformations: More general transformations are the following so-called *affine transformations*:

$$f_{(a,b)} = E_{(a,b)}(m) \equiv am + b \pmod{26}, \quad (3.45)$$

with  $a, b \in \mathbb{Z}$  the key,  $0 \leq a, b, m \leq 26$  and  $\gcd(a, 26) = 1$ , together with

$$f_{(a,b)}^{-1} = D_{(a,b)}(c) \equiv a^{-1}(c - b) \pmod{26}, \quad (3.46)$$

where  $a^{-1}$  is the multiplicative inverse of  $a$  modulo 26 (even more generally, the modulus 26 could be any number greater than 26, but normally chosen to be a prime number).

**Example 3.3.1.** In character ciphers, we have

$$\begin{aligned} E_3(\text{IBM}) &= \text{LEP}, \\ E_4(\text{NIST}) &= \text{QLVW}, \\ E_7(\text{ENCRYPTION}) &= \text{LUJXFWAPYU}. \\ D_4(\text{YWHEBKNJEW}) &= \text{CALIFORNIA}, \\ D_5(\text{ZIBGVITY}) &= \text{ENGLAND}, \\ D_6(\text{XYWLSJNCIH}) &= \text{DECRYPTION}. \end{aligned}$$

**Exercise 3.3.1.** Decrypt the following character ciphertexts:

$$\begin{aligned} D_7(\text{VHFFNGBVTMBHG}), \\ D_9(\text{JVTILIZKP}). \end{aligned}$$

**Example 3.3.2.** Use the following affine transformations

$$f_{(7,21)} \equiv 7m + 21 \pmod{26}$$

and

$$f_{(7,21)}^{-1} \equiv 7^{-1}(c - 21) \pmod{26}$$

to encrypt the message SECURITY and decrypt the message VLXIJH. To encrypt the message, we have

$S = 18,$	$7 \cdot 18 + 21 \pmod{26} = 17,$	$S \Rightarrow R,$
$E = 4,$	$7 \cdot 4 + 21 \pmod{26} = 23,$	$E \Rightarrow X,$
$C = 2,$	$7 \cdot 2 + 21 \pmod{26} = 9,$	$C \Rightarrow J,$
$U = 20,$	$7 \cdot 20 + 21 \pmod{26} = 5,$	$U \Rightarrow F,$
$R = 17,$	$7 \cdot 17 + 21 \pmod{26} = 10,$	$R \Rightarrow K,$
$I = 8,$	$7 \cdot 8 + 21 \pmod{26} = 25,$	$I \Rightarrow Z,$
$T = 19,$	$7 \cdot 19 + 21 \pmod{26} = 24,$	$T \Rightarrow Y,$
$Y = 24,$	$7 \cdot 24 + 21 \pmod{26} = 7,$	$Y \Rightarrow H.$

Thus,  $E_{(7,21)}(\text{SECURITY}) = \text{RXJFKZYH}$ . Similarly, to decrypt the message VLXIJH, we have

$$\begin{array}{lll} V = 21, & 7^{-1} \cdot (21 - 21) \bmod 26 = 0, & V \Rightarrow A, \\ L = 11, & 7^{-1} \cdot (11 - 21) \bmod 26 = 6, & L \Rightarrow G, \\ X = 23, & 7^{-1} \cdot (13 - 21) \bmod 26 = 4, & X \Rightarrow E, \\ I = 8, & 7^{-1} \cdot (8 - 21) \bmod 26 = 13, & I \Rightarrow N, \\ J = 9, & 7^{-1} \cdot (9 - 21) \bmod 26 = 2, & J \Rightarrow C, \\ H = 7, & 7^{-1} \cdot (7 - 21) \bmod 26 = 24, & H \Rightarrow Y. \end{array}$$

Thus,  $D_{(7,21)}(\text{VLXIJH}) = \text{AGENCY}$ .

**Exercise 3.3.2.** Use the affine transformation

$$f_{(11,23)} = 11m + 23 \pmod{26}$$

to encrypt the message THE NATIONAL SECURITY AGENCY. Use also the inverse transformation

$$f_{(11,23)}^{-1} = 11^{-1}(c - 23) \pmod{26}$$

to verify your result.

**(III) Polygraphic (Block) Ciphers.** Monographic ciphers can be made more secure by splitting the plaintext into groups of letters (rather than a single letter) and then performing the encryption and decryption on these groups of letters. This block technique is called *block ciphering*. Block cipher is also called a *polygraphic cipher*. Block ciphers may be described as follows:

- (1) Split the message  $M$  into blocks of  $n$ -letters (when  $n = 2$  it is called a *digraphic cipher*)  $M_1, M_2, \dots, M_j$ ; each block  $M_i$  for  $1 \leq i \leq j$  is a block consisting of  $n$  letters.
- (2) Translate the letters into their numerical equivalents and form the ciphertext:

$$\mathbf{C}_i \equiv \mathbf{A}\mathbf{M}_i + \mathbf{B} \pmod{N}, \quad i = 1, 2, \dots, j \quad (3.47)$$

where  $(\mathbf{A}, \mathbf{B})$  is the key,  $\mathbf{A}$  is an invertible  $n \times n$  matrix with  $\gcd(\det(\mathbf{A}), N) = 1$ ,  $\mathbf{B} = (B_1, B_2, \dots, B_n)^T$ ,  $\mathbf{C}_i = (c_1, c_2, \dots, c_n)^T$  and  $\mathbf{M}_i = (m_1, m_2, \dots, m_n)^T$ . For simplicity, we just consider

$$\mathbf{C}_i \equiv \mathbf{A}\mathbf{M}_i \pmod{26}. \quad (3.48)$$

- (3) For decryption, we perform

$$\mathbf{M}_i \equiv \mathbf{A}^{-1}(\mathbf{C}_i - \mathbf{B}) \pmod{N}, \quad (3.49)$$

where  $\mathbf{A}^{-1}$  is the inverse matrix of  $\mathbf{A}$ . Again, for simplicity, we just consider

$$\mathbf{M}_i \equiv \mathbf{A}^{-1}\mathbf{C}_i \pmod{26}. \quad (3.50)$$

**Example 3.3.3.** Let

$$M = \text{YOUR PIN NO IS FOUR ONE TWO SIX}$$

be the plaintext and  $n = 3$ . Let also the encryption matrix be

$$\mathbf{A} = \begin{pmatrix} 11 & 2 & 19 \\ 5 & 23 & 25 \\ 20 & 7 & 17 \end{pmatrix}.$$

Then the encryption and decryption of the message can be described as follows:

- (1) Split the message  $M$  into blocks of 3-letters and translate these letters into their numerical equivalents:

Y	O	U	R	P	I	N	N	O	I	S	F
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
24	14	20	17	15	8	13	13	14	8	18	5
O	U	R	O	N	E	T	W	O	S	I	X
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
14	20	17	14	13	4	19	22	14	18	8	23

- (2) Encrypt these nine blocks in the following way:

$$\mathbf{C}_1 = \mathbf{A} \begin{pmatrix} 24 \\ 14 \\ 20 \end{pmatrix} = \begin{pmatrix} 22 \\ 6 \\ 8 \end{pmatrix}, \quad \mathbf{C}_2 = \mathbf{A} \begin{pmatrix} 17 \\ 15 \\ 8 \end{pmatrix} = \begin{pmatrix} 5 \\ 6 \\ 9 \end{pmatrix},$$

$$\mathbf{C}_3 = \mathbf{A} \begin{pmatrix} 13 \\ 13 \\ 14 \end{pmatrix} = \begin{pmatrix} 19 \\ 12 \\ 17 \end{pmatrix}, \quad \mathbf{C}_4 = \mathbf{A} \begin{pmatrix} 8 \\ 18 \\ 5 \end{pmatrix} = \begin{pmatrix} 11 \\ 7 \\ 7 \end{pmatrix},$$

$$\mathbf{C}_5 = \mathbf{A} \begin{pmatrix} 14 \\ 20 \\ 17 \end{pmatrix} = \begin{pmatrix} 23 \\ 19 \\ 7 \end{pmatrix}, \quad \mathbf{C}_6 = \mathbf{A} \begin{pmatrix} 14 \\ 13 \\ 4 \end{pmatrix} = \begin{pmatrix} 22 \\ 1 \\ 23 \end{pmatrix},$$

$$\mathbf{C}_7 = \mathbf{A} \begin{pmatrix} 19 \\ 22 \\ 14 \end{pmatrix} = \begin{pmatrix} 25 \\ 15 \\ 18 \end{pmatrix}, \quad \mathbf{C}_8 = \mathbf{A} \begin{pmatrix} 18 \\ 8 \\ 23 \end{pmatrix} = \begin{pmatrix} 1 \\ 17 \\ 1 \end{pmatrix}.$$

(3) Translating these into letters, we get the ciphertext  $C$ :

22	6	8	5	6	9	19	12	17	11	7	7
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
W	G	I	F	G	J	T	M	R	L	H	H
23	19	7	22	1	23	25	15	18	1	17	1
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
X	T	H	W	B	X	Z	P	S	B	R	B

(4) To recover the message  $M$  from  $C$ , we first compute  $A^{-1}$  modulo 26:

$$\mathbf{A}^{-1} = \begin{pmatrix} 11 & 2 & 19 \\ 5 & 23 & 25 \\ 20 & 7 & 17 \end{pmatrix}^{-1} = \begin{pmatrix} 10 & 23 & 7 \\ 15 & 9 & 22 \\ 5 & 9 & 21 \end{pmatrix}.$$

and then perform  $\mathbf{C}_i = \mathbf{A}^{-1}\mathbf{C}_i$  as follows:

$$\mathbf{M}_1 = \mathbf{A}^{-1} \begin{pmatrix} 22 \\ 6 \\ 8 \end{pmatrix} = \begin{pmatrix} 24 \\ 14 \\ 20 \end{pmatrix}, \quad \mathbf{M}_2 = \mathbf{A}^{-1} \begin{pmatrix} 5 \\ 6 \\ 9 \end{pmatrix} = \begin{pmatrix} 17 \\ 15 \\ 8 \end{pmatrix},$$

$$\mathbf{M}_3 = \mathbf{A}^{-1} \begin{pmatrix} 19 \\ 12 \\ 17 \end{pmatrix} = \begin{pmatrix} 13 \\ 13 \\ 14 \end{pmatrix}, \quad \mathbf{M}_4 = \mathbf{A}^{-1} \begin{pmatrix} 11 \\ 7 \\ 7 \end{pmatrix} = \begin{pmatrix} 8 \\ 18 \\ 5 \end{pmatrix},$$

$$\mathbf{M}_5 = \mathbf{A}^{-1} \begin{pmatrix} 23 \\ 19 \\ 7 \end{pmatrix} = \begin{pmatrix} 14 \\ 20 \\ 17 \end{pmatrix}, \quad \mathbf{M}_6 = \mathbf{A}^{-1} \begin{pmatrix} 22 \\ 1 \\ 23 \end{pmatrix} = \begin{pmatrix} 14 \\ 13 \\ 4 \end{pmatrix},$$

$$\mathbf{M}_7 = \mathbf{A}^{-1} \begin{pmatrix} 25 \\ 15 \\ 18 \end{pmatrix} = \begin{pmatrix} 19 \\ 22 \\ 14 \end{pmatrix}, \quad \mathbf{M}_8 = \mathbf{A}^{-1} \begin{pmatrix} 1 \\ 17 \\ 1 \end{pmatrix} = \begin{pmatrix} 18 \\ 8 \\ 23 \end{pmatrix}.$$

So, we have:

24	14	20	17	15	8	13	13	14	8	18	5
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Y	O	U	R	P	I	N	N	O	I	S	F
14	20	17	14	13	4	19	22	14	18	8	23
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
O	U	R	O	N	E	T	W	O	S	I	X

which is the original message.

**Exercise 3.3.3.** Let

$$\mathbf{A} = \begin{pmatrix} 3 & 13 & 21 & 9 \\ 15 & 10 & 6 & 25 \\ 10 & 17 & 4 & 8 \\ 1 & 23 & 7 & 2 \end{pmatrix} \quad \text{and} \quad \mathbf{B} = \begin{pmatrix} 1 \\ 21 \\ 8 \\ 17 \end{pmatrix}.$$

Use the block transformation

$$C_i \equiv \mathbf{A}M_i + \mathbf{B} \pmod{26}$$

to encrypt the following message

PLEASE SEND ME THE BOOK, MY CREDIT CARD NO IS  
SIX ONE TWO ONE THREE EIGHT SIX ZERO  
ONE SIX EIGHT FOUR NINE SEVEN ZERO TWO.

Use

$$M_i \equiv \mathbf{A}^{-1}(C_i - \mathbf{B}) \pmod{26}$$

to verify your result, where

$$\mathbf{A}^{-1} = \begin{pmatrix} 26 & 13 & 20 & 5 \\ 0 & 10 & 11 & 0 \\ 9 & 11 & 15 & 22 \\ 9 & 22 & 6 & 25 \end{pmatrix}.$$

**(IV) Exponentiation Ciphers.** The exponentiation cipher, invented by Pohlig and Hellman in 1976, may be described as follows. Let  $p$  be a prime number,  $M$  the numerical equivalent of the plaintext, where each letter of the plaintext is replaced by its two digit equivalent, as defined in Table 3.4. Subdivide  $M$  into blocks  $M_i$  such that  $0 < M_i < p$ . Let  $k$  be an integer with  $0 < k < p$  and  $\gcd(k, p-1) = 1$ . Then the encryption transformation for  $M_i$  is defined by

$$C_i = E_k(M_i) \equiv M_i^k \pmod{p}, \quad (3.51)$$

**Table 3.4.** Two digit equivalents of letters

□	A	B	C	D	E	F	G	H	I	J	K	L	M
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
00	01	02	03	04	05	06	07	08	09	10	11	12	13
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	
14	15	16	17	18	19	20	21	22	23	24	25	26	

and the decryption transformation by

$$M_i = D_{k^{-1}}(C_i) \equiv C_i^{k^{-1}} \equiv (M_i^k)^{k^{-1}} \equiv M_i \pmod{p}, \quad (3.52)$$

where  $k \cdot k^{-1} \equiv 1 \pmod{p-1}$ .

**Example 3.3.4.** Let  $p = 7951$  and  $k = 91$  such that  $\gcd(7951-1, 91) = 1$ . Suppose we wish to encrypt the message

$M = \text{ENCRYPTION REGULATION MOVES TO A STEP CLOSER}$

using the exponentiation cipher. Firstly, we convert all the letters in the message to their numerical equivalents via Table 3.4

05 14 03 18 25 16 20 09 15 14 00 18 05 07 21 12 01 20 09 15 14 00  
13 15 22 05 19 00 20 15 00 01 00 19 20 05 16 00 03 12 15 19 05 18

and group them into blocks with four digits

0514 0318 2516 2009 1514 0018 0507 2112 0120 0915 1400  
1315 2205 1900 2015 0001 0019 2005 1600 0312 1519 0518

Then we perform the following computation

$$\begin{aligned}
 C_1 &= 0514^{91} \pmod{7951} = 2174 & C_2 &= 0318^{91} \pmod{7951} = 4468 \\
 C_3 &= 2516^{91} \pmod{7951} = 7889 & C_4 &= 2009^{91} \pmod{7951} = 6582 \\
 C_5 &= 1514^{91} \pmod{7951} = 924 & C_6 &= 0018^{91} \pmod{7951} = 5460 \\
 C_7 &= 0507^{91} \pmod{7951} = 7868 & C_8 &= 2112^{91} \pmod{7951} = 7319 \\
 C_9 &= 0120^{91} \pmod{7951} = 726 & C_{10} &= 915^{91} \pmod{7951} = 2890 \\
 C_{11} &= 1400^{91} \pmod{7951} = 7114 & C_{12} &= 1315^{91} \pmod{7951} = 5463 \\
 C_{13} &= 2205^{91} \pmod{7951} = 5000 & C_{14} &= 1900^{91} \pmod{7951} = 438 \\
 C_{15} &= 2015^{91} \pmod{7951} = 2300 & C_{16} &= 0001^{91} \pmod{7951} = 1 \\
 C_{17} &= 0019^{91} \pmod{7951} = 1607 & C_{18} &= 2005^{91} \pmod{7951} = 3509 \\
 C_{19} &= 1600^{91} \pmod{7951} = 7143 & C_{20} &= 0312^{91} \pmod{7951} = 5648 \\
 C_{21} &= 1519^{91} \pmod{7951} = 3937 & C_{22} &= 0518^{91} \pmod{7951} = 4736.
 \end{aligned}$$

So, the ciphertext of  $M$  is



2174 4468 7889 6582 0924 5460 7868 7319 0726 2890 7114  
 5463 5000 0438 2300 0001 1607 3509 7143 5648 3937 5064.

To decrypt the ciphertext  $C$  back to the plaintext  $M$ , since the secret key  $k = 91$  and the prime modulus  $p = 7951$  are known, we compute the multiplicative inverse  $k^{-1}$  of  $k$  modulo  $p - 1$  as follows:

$$k^{-1} \equiv \frac{1}{k} \pmod{p-1} \equiv \frac{1}{91} \pmod{7950} \equiv 961 \pmod{7950}.$$

Thus, we have

$$\begin{aligned} M_1 &= 2174^{961} \pmod{7951} = 514 & M_2 &= 4468^{961} \pmod{7951} = 318 \\ M_3 &= 7889^{961} \pmod{7951} = 2516 & M_4 &= 6582^{961} \pmod{7951} = 2009 \\ M_5 &= 924^{961} \pmod{7951} = 1514 & M_6 &= 5460^{961} \pmod{7951} = 18 \\ M_7 &= 7868^{961} \pmod{7951} = 507 & M_8 &= 7319^{961} \pmod{7951} = 2112 \\ M_9 &= 726^{961} \pmod{7951} = 120 & M_{10} &= 2890^{961} \pmod{7951} = 915 \\ M_{11} &= 7114^{961} \pmod{7951} = 1400 & M_{12} &= 5463^{961} \pmod{7951} = 1315 \\ M_{13} &= 5000^{961} \pmod{7951} = 2205 & M_{14} &= 438^{961} \pmod{7951} = 1900 \\ M_{15} &= 2300^{961} \pmod{7951} = 2015 & M_{16} &= 1^{961} \pmod{7951} = 1 \\ M_{17} &= 1607^{961} \pmod{7951} = 19 & M_{18} &= 3509^{961} \pmod{7951} = 2005 \\ M_{19} &= 7143^{961} \pmod{7951} = 1600 & M_{20} &= 5648^{961} \pmod{7951} = 312 \\ M_{21} &= 3937^{961} \pmod{7951} = 1519 & M_{22} &= 4736^{961} \pmod{7951} = 518. \end{aligned}$$

Therefore, we have recovered the original message.

**Exercise 3.3.4.** Let  $p = 9137$  and  $k = 73$  so that  $\gcd(p - 1, k) = 1$  and  $k^{-1} \pmod{p-1} = 750$ . Use the exponentiation transformation  $C = M^k \pmod{p}$  to encrypt the following message:

THE CESG IS THE UK NATIONAL TECHNICAL AUTHORITY  
 ON INFORMATION SECURITY.

THE NSA IS THE OFFICIAL INTELLIGENCE-GATHERING  
 ORGANIZATION OF THE UNITED STATES.

Use also  $M = C^{k^{-1}} \pmod{p}$  to verify your result.

**Exercise 3.3.5 (A challenge problem).** The following cryptogram was presented by Édouard Lucas at the 1891 meeting of the French Association for Advancement of Science (see Williams, [257]); it has never been decrypted, and hence is suitable as a challenge to the interested reader.

XSJOD	PEFOC	XCXFM	RDZME
JZCOA	YUMTZ	LTDNJ	HBUSQ
XTFLK	XCBDY	GYJKK	QBSAH
QHXPE	DBMLI	ZOYVQ	PRETL
TPMUK	XGHIV	ARLAH	SPGGP

VBQYH	TVJYJ	NXFFX	BVLCZ
LEFXF	VDMUB	QBIJV	ZGGAJ
TRYQB	AIDEZ	EZEDX	KS

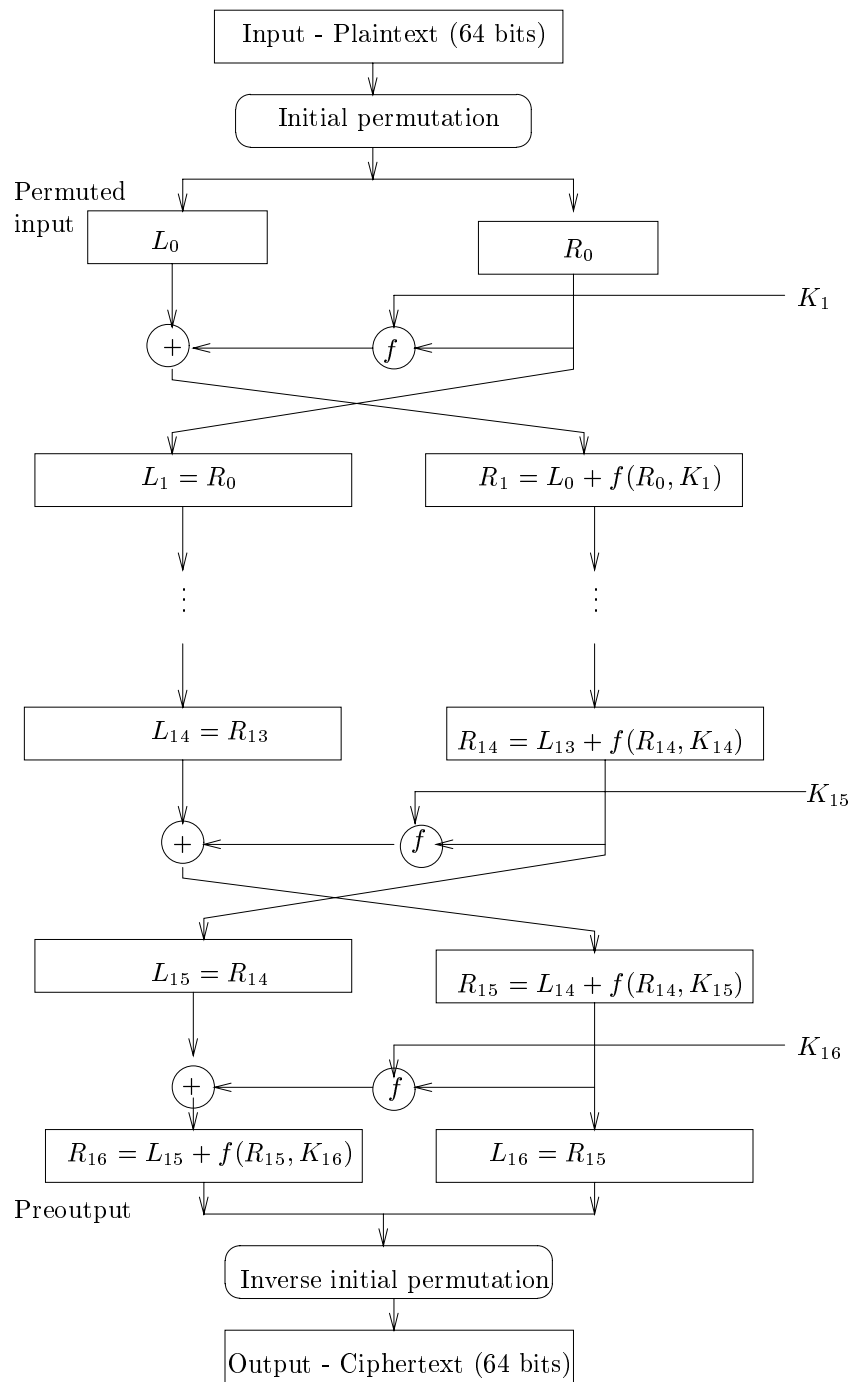
### 3.3.3 Data/Advanced Encryption Standard (DES/AES)

The most popular secret-key cryptographic scheme in use (by both governments and private companies) is the Data Encryption Standard (DES) — DES was designed at IBM and approved in 1977 as a standard by the U.S. National Bureau of Standards (NBS), now called the National Institute of Standards and Technology (NIST). This standard, first issued in 1977 (FIPS 46 – Federal Information Processing Standard 46), is reviewed every five years. It is currently specified in FIPS 46-2. NIST is proposing to replace FIPS 46-2 with FIPS 46-3 to provide for the use of Triple DES (TDES) as specified in the American National Standards Institute (ANSI) X9.52 standard. Comments were sought from industry, government agencies, and the public on the draft of FIPS 46-3 before 15 April 15, 1999.

The standard (algorithm) uses a product transformation of transpositions, substitutions, and non-linear operations. They are applied for 16 iterations to each block of a message; the message is split into 64-bit message blocks. The key used is composed of 56 bits taken from a 64-bit key which includes 8 parity bits. The algorithm is used in reverse to decrypt each ciphertext block and the same key is used for both encryption and decryption. The algorithm itself is shown schematically in Figure 3.5, where the  $\oplus$  is the “exclusive or” (XOR) operator. The DES algorithm takes as input a 64-bit message (plaintext)  $M$  and a 56-bit key  $K$ , and produces a 64-bit ciphertext  $C$ . DES first applies an initial fixed bit-permutation (IP) to  $M$  to obtain  $M'$ . This permutation has no apparent cryptographic significance. Second, DES divides  $M'$  into a 32-bit left half  $L_0$  and 32-bit right half  $R_0$ . Third, DES executes the following operations for  $i = 1, 2, \dots, 16$  (there are 16 “rounds”):

$$\left. \begin{aligned} L_i &= R_{i-1}, \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i), \end{aligned} \right\} \quad (3.53)$$

where  $f$  is a function that takes a 32-bit right half and a 48-bit “round key” and produces a 32-bit output. Each round key  $K_i$  contains a different subset of the 56-bit key bits. Finally, the pre-ciphertext  $C' = (R_{16}, L_{16})$  is permuted according to  $IP^{-1}$  to obtain the final ciphertext  $C$ . To decrypt, the algorithm is run in reverse: a permutation, 16 XOR rounds using the round key in reverse order, and a final permutation that recovers the plaintext. All of this extensive bit manipulations can be incorporated into the logic of a single



**Figure 3.5.** The Data Encryption Standard (DES) algorithm

special-purpose microchip, so DES can be implemented very efficiently. However, the DES cracking project being undertaken by the Electronic Frontier Foundation is able to break the encryption for 56 bit DES in about 22 hours. As a result, NIST has recommended that businesses use Triple DES<sup>5</sup> (TDES), which involves three different DES encryption and decryption operations. Let  $E_K(M)$  and  $D_K(C)$  represent the DES encryption and decryption of  $M$  and  $C$  using DES key  $K$ , respectively. Each TDES encryption/decryption operation (as specified in ANSI X9.52) is a compound operation of DES encryption and decryption operations. The following operations are used in TDES:

- (1) **TDES encryption operation:** the transformation of a 64-bit block  $M$  into a 64-bit block  $C$  is defined as follows:

$$C = E_{K_3}(D_{K_2}(E_{K_1}(M))). \quad (3.54)$$

- (2) **TDES decryption operation:** the transformation of a 64-bit block  $C$  into a 64-bit block  $M$  is defined as follows:

$$M = D_{K_1}(E_{K_2}(D_{K_3}(C))). \quad (3.55)$$

There are three options for the TDES *key bundle*  $(K_1, K_2, K_3)$ :

- (1)  $K_1, K_2$ , and  $K_3$  are independent keys.
- (2)  $K_1, K_2$  are independent keys and  $K_3 = K_1$ .
- (3)  $K_1 = K_2 = K_3$ .

For example, if option 2 is chosen, then the TDES encryption and decryption are as follows:

$$C = E_{K_1}(D_{K_2}(E_{K_1}(M))), \quad (3.56)$$

$$M = D_{K_1}(E_{K_2}(D_{K_1}(C))). \quad (3.57)$$

Interested readers are suggested to consult the current NIST report FIPS 46-3 [173] for the new standard of the TDES.

It is interesting to note that some experts say DES is still secure when used properly. However, Edward Roback at the NIST said that the DES, which uses 56-bit encryption keys, is no longer sufficiently difficult to decrypt. For example, in February 1998, a team of engineers used a distributed “brute force” decryption program to break a 56-bit DES key in 39 days, about three

---

<sup>5</sup> Triple DES is a type of *multiple encryption*. Multiple encryption is a combination technique aimed to improve the security of a block algorithm. It uses an algorithm to encrypt the same plaintext block multiple times with multiple keys. The simplest multiple encryption is the so-called *double encryption* in which an algorithm is used to encrypt a block twice with two different keys – first encrypt a block with the first key, and then encrypt the resulting ciphertext with the second key:  $C = E_{k_2}(E_{k_1}(M))$ . The decryption is just the reverse process of the encryption:  $M = D_{k_1}(D_{k_2}(C))$ .

times faster than it took another team just the year before, and more recently, the team cracked DES in just over 22 hours earlier this year.

The U.S. Department of Commerce's NIST had issued a formal call on 12 September 1997 for companies, universities, and other organizations to submit algorithm proposals for a new generation encryption standard for protecting sensitive data well into the 21st century. This new Advanced Encryption Standard (AES) will replace the DES and support encryption key size up to 256 bits and must be available royalty-free throughout the world. On 20 August 1998 at the First AES Candidate Conference (AES1), NIST announced fifteen (15) official AES candidate algorithms submitted by researchers from twelve (12) different countries, including the United States, Australia, France, Germany, Japan, Norway and the United Kingdom. Since then, cryptographers have tried to find ways to *attack* the different algorithms, looking for weaknesses that would compromise the encrypted information. Shortly after the Second AES Candidate Conference (AES2) on 22–23 March 1999 in Rome, Italy, NIST announced on 9 August 1999 that the following five (5) contenders had been chosen as finalist for the AES, all are block ciphers:

- (1) **MARS**: Developed by International Business Machines (IBM) Corporation of Armonk, New York, USA.
- (2) **RC6**: Developed by RSA Laboratories of Bedford, Massachusetts, USA.
- (3) **Rijndael**: Developed by Joan Daemen and Vincent Rijmen of Belgium.
- (4) **Serpent**: Developed by Ross Anderson, Eli Biham and Lars Knudsen of the United Kingdom, Israel and Norway, respectively.
- (5) **Twofish**: Developed by Bruce Schneier, John Kelsey, Doug Whiting, David Wagner Chris Hall and Niels Ferguson, of Counterpane Systems, Minneapolis, USA.

These five finalist algorithms had received further analysis during a second, more in-depth review period (August 1999–May 2000) in the selection of the final algorithm for the FIPS (Federal Information Processing Standard) AES. On 2 October 2000, the algorithm Rijndael, developed by Joan Daemen (Proton World International, Belgium) and Vincent Rijmen (Katholieke Universiteit Leuven, Belgium) was finally chosen to be the AES. The strong points of Rijndael are a simple and elegant design, efficient and fast on modern processors, but also compact in hardware and on smartcards. These features make Rijndael suitable for a wide range of applications. It will be used to protect sensitive but 'unclassified' electronic information of the US government. During the last year, a large number of products and applications has been AES-enabled. Therefore, it is very likely to become a worldwide de facto standard in numerous other applications such as Internet security, bank cards and ATMs.

### 3.3.4 Public-Key Cryptography

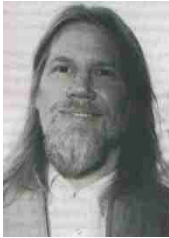
*An obvious requirement of a good cryptographic system is that secret messages should be easy to encrypt and decrypt for legitimate users, and these processes (or, at least, decryption) should be hard for everyone else. Number Theory has turned out to be an excellent source of computational problems that have both easy and (apparently) hard aspects and that can be used as the backbone of several cryptographic systems.*

CARL POMERANCE

Cryptology and Computational Number Theory [191]

In their seminal paper “New Directions in Cryptography” [66], Diffie<sup>6</sup> and Hellman<sup>7</sup>, both in the Department of Electrical Engineering at Stanford University at the time, first proposed the idea and the concept of public-key

6



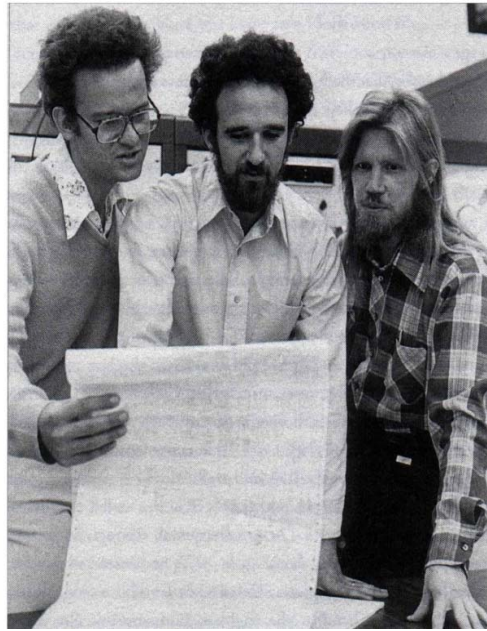
Whitfield Diffie (1944– ), a Distinguished Engineer at Sun Microsystems in Palo Alto, California, is perhaps best known for his 1975 discovery of the concept of public-key cryptography, for which he was awarded a Doctorate in Technical Sciences (Honoris Causa) by the Swiss Federal Institute of Technology in 1992. He received a BSc degree in mathematics from the Massachusetts Institute of Technology in 1965. Prior to becoming interested in cryptography, he worked on the development of the Mathlab symbolic manipulation system — sponsored jointly at Mitre and the MIT Artificial Intelligence Laboratory — and later on proof of correctness of computer programs at Stanford University. Diffie was the recipient of the IEEE Information Theory Society Best Paper Award 1979 for the paper *New Directions in Cryptography* [66], the IEEE Donald E. Fink award 1981 for expository writing for the paper *Privacy and Authentication* [67] (both papers co-authored with Martin Hellman), and the National Computer Systems Security Award for 1996. (Photo by courtesy of Dr. Simon Singh.)

7



Martin E. Hellman (1945– ), the father of modern (public key) cryptography, received his BEng from New York University in 1966, and his MSc and PhD from Stanford University in 1967 and 1969, respectively, all in Electrical Engineering. Hellman was on the research staff at IBM's Watson Research Center from 1968-69 and on the faculty of Electrical Engineering at MIT from 1969-71. He returned to Stanford as a faculty member in 1971, where he served on the regular faculty until becoming Professor Emeritus in 1996. He has authored over 60 technical papers, five U.S. and a number of foreign patents. His work, particularly the invention of public key cryptography, has been covered in the popular media including Scientific American and Time magazine. He was the recipient of an IEEE Centennial Medal (1984). Notice that Diffie, Hellman and Merkle are the three joint inventors of public-key cryptography, with Diffie and Merkle as Hellman's research assistant and PhD student. (Photo by courtesy of Prof. Hellman.)

cryptography as well as digital signatures; they also proposed in the same time a key-exchange protocol, based on the hard *discrete logarithm problem*, for two parties to form a common private key over the insecure channel (see Subsection 3.3.2).



**Figure 3.6.** The DHM crypto years: (Left to right) Merkle, Hellman and Diffie (Photo by courtesy of Dr. Simon Singh)

It should be noted that Ralph Merkle<sup>8</sup>, deserves equal credit with Diffie and Hellman for the invention of public key cryptography. Although his paper *Secure Communication Over Insecure Channels* [158] was published in 1978,

8



Ralph C. Merkle (1952– ) studied Computer Science at the University of California at Berkeley with a B.A. in 1974 and a M.S. in 1977, and obtained his PhD in Electrical Engineering at Stanford University in 1979 with the thesis entitled *Secrecy, Authentication, and Public Key Systems*, with Prof. Martin Hellman as his thesis advisor. Merkle co-invented public-key cryptography, received the 1997 ACM Kanellakis Award (along with Leonard Adleman, Whitfield Diffie, Martin Hellman, Ronald Rivest and Adi Shamir), the 1998 Feynman Prize in Nanotechnology for theory, the 1999 IEEE Kobayashi Award, and the 2000 RSA Award in Mathematics. He is currently a Principal Fellow at Zyvex, working on molecular manufacturing (also known as nanotechnology). (Photo by courtesy of Dr. Merkle.)

two years later than Diffie and Hellman's paper *New Directions in Cryptography*, it was submitted in August 1975. Also, his conception of *public key distribution* occurred in the Fall of 1974, again before Diffie and Hellman conceived of *public key cryptosystems*.

Remarkably enough, just about one or two years later, three MIT computer scientists, Rivest, Shamir, and Adleman, proposed in 1978 a practical public-key cryptosystem based on primality testing and integer factorization, now widely known as RSA cryptosystem (see Subsection 3.3.6). More specifically, they based on their encryption and decryption on mod- $n$  arithmetic, where  $n$  is the product of two large prime numbers  $p$  and  $q$ . A special case based on mod- $p$  arithmetic with  $p$  prime, now known as exponential cipher, had already been studied by Pohlig and Hellman in 1978 [176].

It is interesting to note that in December 1997 the Communication-Electronics Security Group (CESG) of the British Government Communications Headquarters (GCHQ), claimed that public-key cryptography was conceived by Ellis<sup>9</sup> in 1970 and implemented by two of his colleagues Cocks<sup>10</sup>

9



James H. Ellis (1924–1997) was conceived in Britain but was born in Australia. While still a baby, he returned to and grew up in London. He studied Physics at Imperial College, London and worked in the Post Office Research Station at Dollis Hill. In 1965, Ellis, together with the cryptographic division at Dollis Hill, moved to Cheltenham to join the newly formed Communication-Electronics Security Group (CESG), a special section of the GCHQ, devoted to ensuring the security of British communications. Ellis was unpredictable, introverted and a rather quirky worker, he was never put in charge of any of the important CESG research groups, and he even didn't really fit into the day-to-day business of CESG. Nevertheless, he was a foremost British government cryptographer. Ellis had a good reputation as a cryptoguru, and if other researchers found themselves with impossible problems, they would knock his door in the hope that his vast knowledge and originality would provide a solution. It was probably because of this reputation that the British military asked him in the beginning of 1969 to investigate the key distribution problem, that led him to have the idea of the non-secret encryption.

10



Clifford C. Cocks studied mathematics, specialized in number theory, at the University of Cambridge and joined the CESG in September 1973. While as a school student in Manchester Grammar School, he represented Britain at the International Mathematical Olympiad in Moscow in 1968 and won a Silver prize. Before joining CESG he knew very little about encryption and its intimate connection with military and diplomatic communications, so his mentor, Nick Patterson at CESG told him Ellis's idea for public-key cryptography. "Because I had been working in number theory, it was natural to think about one-way functions, something you could do but not undo. Prime numbers and factoring was a natural candidate," explained by Cocks. It did not take him too long to formulate a special case of the RSA public key cryptography.



and Williamson<sup>11</sup> between 1973 and 1976 in CESG, by releasing the following five papers:

- [1] James H. Ellis, *The Possibility of Non-Secret Encryption*, January 1970, 9 pages.
- [2] Clifford C. Cocks, *A Note on Non-Secret Encryption*, 20 November 1973, 2 pages.
- [3] Malcolm J. Williamson, *Non-Secret Encryption Using a Finite Field*, 21 January 1974, 2 pages.
- [4] Malcolm Williamson, *Thoughts on Cheaper Non-Secret Encryption*, 10 August 1976, 3 pages.
- [5] James Ellis, *The Story of Non-Secret Encryption*, 1987, 9 pages.

The US Government's National Security Agency (NSA) also made a similar claim that they had public-key cryptography a decade earlier. It must be pointed out that there are apparently two parallel universes in cryptography, the public and the secret worlds. The CESG and even the NSA people certainly deserve some kind of credit, but according to the “first to publish, not first to keep secret” rule, the *full* credit of the invention of public-key cryptography goes to Diffie, Hellman and Merkle (along with Rivest, Shamir and Adleman for their first practical implementation). It must also be pointed out that Diffie and Hellman [66] in the same time also proposed the marvelous idea of digital signatures, and in implementing their RSA cryptosystem, Rivest, Shmire and Adleman also implemented the idea of digital signatures, whereas none of the CESG released papers showed any evidence that they had any thought of digital signatures, which is half of the Diffie-Hellman-Merkle public-key cryptography invention!

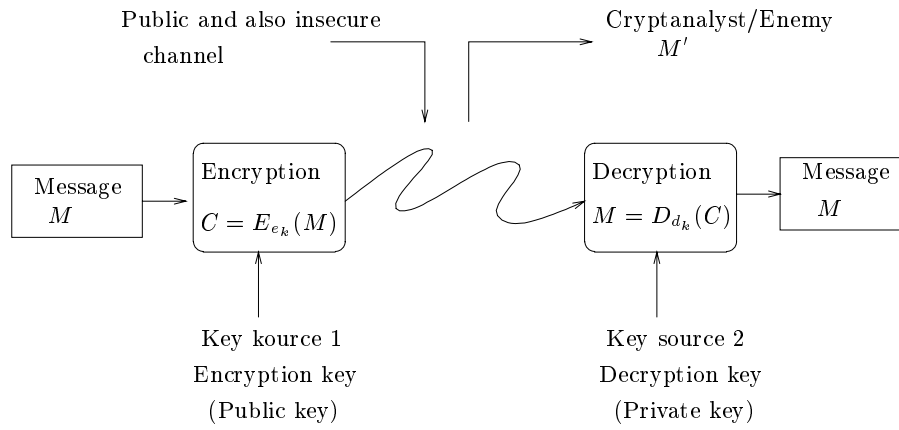
In a public-key (non-secret key) cryptosystem (see Figure 3.7), the encryption key  $e_k$  and decryption key  $d_k$  are different, that is,  $e_k \neq d_k$  (this is why we call public-key cryptosystems *asymmetric key cryptosystems*). Since  $e_k$  is

---

<sup>11</sup>



Malcolm J. Williamson also attended the Manchester Grammar School and studied mathematics at the University of Cambridge, but joined the CESG in September 1974. Same as Clifford Cocks, Malcolm Williamson also represented Britain at the International Mathematical Olympiad in Moscow in 1968 but won a Gold prize. When Cocks first explained his work on public-key cryptography to Williamson, Williamson really didn't believe it and tried to prove that Cocks had made a mistake and that public-key cryptography did not really exist. Remarkably enough, Williamson failed to find a mistake, instead he found another solution to the problem of key distribution, at roughly the same time that Prof. Martin Hellman discovered it. (Photos of Ellis, Cocks and Williamson by courtesy of Dr. Simon Singh.)



**Figure 3.7.** Modern public-key cryptosystems ( $e_k \neq d_k$ )

only used for encryption, it can be made public; only  $d_k$  must be kept a secret for decryption. To distinguish public-key cryptosystems from secret-key cryptosystems,  $e_k$  is called the *public key*, and  $d_k$  the *private key*; only the key used in secret-key cryptosystems is called the *secret key*. The implementation of public-key cryptosystems is based on *trapdoor one-way functions*.

**Definition 3.3.1.** Let  $S$  and  $T$  be finite sets. A one-way function

$$f : S \rightarrow T \quad (3.58)$$

is an invertible function satisfying

- (1)  $f$  is easy to compute, that is, given  $x \in S$ ,  $y = f(x)$  is easy to compute.
- (2)  $f^{-1}$ , the inverse function of  $f$ , is difficult to compute, that is, given  $y \in T$ ,  $x = f^{-1}(y)$  is difficult to compute.
- (3)  $f^{-1}$  is easy to compute when a trapdoor (i.e., a secret string of information associated with the function) becomes available.

A function  $f$  satisfying only the first two conditions is also called a one-to-one one-way function. If  $f$  satisfies further the third condition, it is called a *trapdoor one-way function*.

**Example 3.3.5.** The following functions are one-way functions:

- (1)  $f : pq \mapsto n$  is a one-way function, where  $p$  and  $q$  are prime numbers. The function  $f$  is easy to compute since the multiplication of  $p$  and  $q$  can be done in polynomial time. However, the computation of  $f^{-1}$ , the inverse of  $f$  is an extremely difficult problem (this is the well-known difficult *integer factorization problem*); there is no efficient algorithm to determine  $p$  and  $q$  from their product  $pq$ , in fact, the fastest factoring algorithm NFS runs in subexponential time.

- (2)  $f_{g,N} : x \mapsto g^x \bmod N$  is a one-way function. The function  $f$  is easy to compute since the modular exponentiation  $g^x \bmod N$  can be performed in polynomial time. But the computation of  $f^{-1}$ , the inverse of  $f$  is an extremely difficult problem (this is the well-known difficult *discrete logarithm problem*); there is no efficient method to determine  $x$  from the knowledge of  $g^x \bmod N$  and  $g$  and  $N$ .
- (3)  $f_{k,N} : x \mapsto x^k \bmod N$  is a trapdoor one-way function, where  $N = pq$  with  $p$  and  $q$  primes, and  $kk' \equiv 1 \pmod{\phi(N)}$ . It is obvious that  $f$  is easy to compute since the modular exponentiation  $x^k \bmod N$  can be done in polynomial time, but  $f^{-1}$ , the inverse of  $f$  (i.e., the  $k$ th root of  $x$  modulo  $N$ ) is difficult to compute. However, if  $k'$ , the trapdoor is given,  $f$  can be easily inverted, since  $(x^k)^{k'} = x$ .

**Remark 3.3.1.** The discrete logarithm problem and the integer factorization problem are the most important difficult number-theoretic problems on which to build one-way functions in practice. Of course, there might exist some other problems which can be used to build one-way functions. One such problem is the so-called Quadratic Residuosity Problem (QRP), that can be simply stated as follows (recall that an integer  $a$  is a quadratic residue modulo  $n$  if  $\gcd(a, n) = 1$  and if there exists a solution  $x$  to the congruence  $x^2 \equiv a \pmod{n}$ ):

Given integers  $a$  and  $n$ , decide if  $a$  is a quadratic residue modulo  $n$ .

If  $n = p$  is an odd prime, then by Euler's criterion (Theorem 1.6.26),  $a$  is a quadratic residue of  $p$  if and only if  $a^{(p-1)/2} \equiv 1 \pmod{p}$ . What about if  $n$  is an odd composite? In this case, we know that  $a$  is a quadratic residue of  $n$  if and only if it is quadratic residue modulo every prime dividing  $n$ . It is evident that if  $\left(\frac{a}{n}\right) = -1$ , then  $\left(\frac{a}{p_i}\right) = -1$  for some  $i$ , and  $a$  is a quadratic nonresidue modulo  $n$ . On the other hand, even if  $\left(\frac{a}{n}\right) = 1$ , it may be possible for  $a$  to be a quadratic nonresidue modulo  $n$ . This is precisely the case that is regarded by some researchers as an intractable problem, since the only method we know for determining quadratic residuosity in this case requires that we first factor  $n$ . Because of our inability to solve the quadratic residuosity problem without factoring, several researchers have proposed cryptosystems whose security is based on the difficulty of determining quadratic residuosity. Whether it is in fact intractable (or at least equivalent to factoring in some sense) remains a very interesting question (McCurley [151]). We shall introduce an encryption scheme based the QRP in Section 3.3.7. There are also some analogues such as elliptic curve analogues of discrete logarithms, which can be used to build one-way functions in public-key cryptosystems; we shall introduce these analogues and their cryptosystems in later sections of this chapter.

**Remark 3.3.2.** Public-key cryptosystems have some important advantages over secret-key cryptosystems in the distribution of the keys. However, when a large amount of information has to be communicated, it may be that the use of public-key cryptography would be too slow, whereas the use of secret-key cryptography could be impossible for the lack of a shared secret key. In practice, it is better to combine the secret-key and public-key cryptography into a single cryptosystem for secure communications. Such a combined system is often called a *hybrid cryptosystem*. A hybrid cryptosystem uses a public-key cryptosystem once at the beginning of the communication to share a short piece of information that is then used as the key for encryption and decryption by means of a “conventional” secret-key cryptosystem in later stages. Such a cryptosystem is essentially a secret-key cryptosystem but still enjoys the advantages of the public-key cryptosystems.

### 3.3.5 Discrete Logarithm Based Cryptosystems

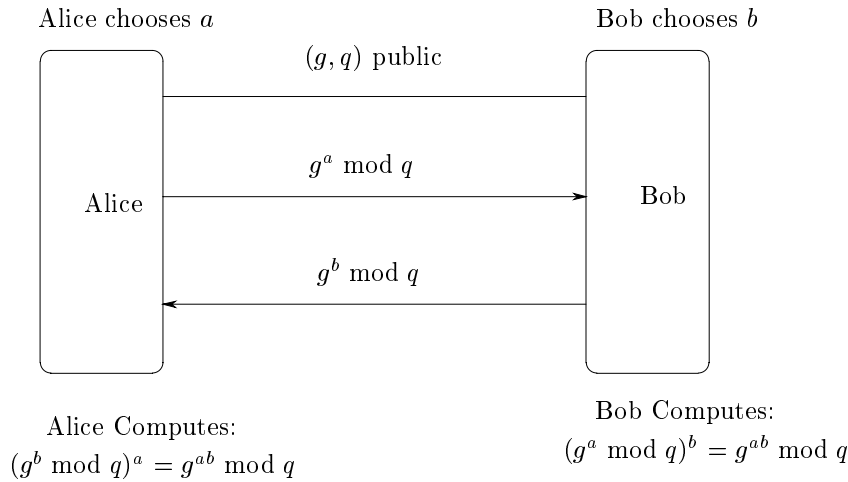
The Diffie-Hellman-Merkle scheme, the first public-key cryptographic scheme, is based on the intractable discrete logarithm problem, which can be described as follows:

$$\begin{array}{ll} \text{Input :} & a, b, n \in \mathbb{N} \\ \text{Output :} & x \in \mathbb{N} \text{ with } a^x \equiv b \pmod{n} \\ & \text{if such a } x \text{ exists} \end{array}$$

The Diffie-Hellman-Merkle scheme has found widespread use in practical cryptosystems, as for example in the optional security features of the NFS file system of SunOS operating system. In this subsection, we shall introduce some discrete logarithm based cryptosystems.

**(I) The Diffie-Hellman-Merkle Key-Exchange Protocol.** Diffie and Hellman [66] in 1976 proposed for the first time a public-key cryptographic scheme based on the difficult discrete logarithm problem. Their scheme was not a public key cryptographic system (first proposed in [66]), but rather a public key distribution system as proposed by Merkle [158]. Such a public key distribution scheme does not send secret messages directly, but rather allows the two parties to agree on a common private key over public networks to be used later in exchanging messages through conventional cryptography. Thus, the Diffie-Hellman-Merkle scheme has the nice property that a very fast scheme such as DES or AES can be used for actual encryption, yet it still enjoys one of the main advantages of public-key cryptography. The Diffie-Hellman-Merkle key-exchange protocol works in the following way (see also Figure 3.8):

- (1) A prime  $q$  and a generator  $g$  are made public (assume all users have agreed upon a finite group over a fixed finite field  $\mathbb{F}_q$ ),
- (2) Alice chooses a random number  $a \in \{1, 2, \dots, q-1\}$  and sends  $g^a \bmod q$  to Bob,
- (3) Bob chooses a random number  $b \in \{1, 2, \dots, q-1\}$  and sends  $g^b \bmod q$  to Alice,
- (4) Alice and Bob both compute  $g^{ab} \bmod q$  and use this as a private key for future communications.



**Figure 3.8.** The Diffie-Hellman-Merkel key-exchange scheme

Clearly, an eavesdropper has  $g$ ,  $q$ ,  $g^a \bmod q$  and  $g^b \bmod q$ , so if he can take discrete logarithms, he can calculate  $g^{ab} \bmod q$  and understand communications. That is, if the eavesdropper can use his knowledge of  $g$ ,  $q$ ,  $g^a \bmod q$  and  $g^b \bmod q$  to recover the integer  $a$ , then he can easily break the Diffie-Hellman-Merkel system. So, the security of the Diffie-Hellman-Merkel system is based on the following assumption:

**Diffie-Hellman-Merkel Assumption:** It is computationally infeasible to compute  $g^{ab}$  from  $g^a$  and  $g^b$ .

In theory, there could be a way to use knowledge of  $g^a$  and  $g^b$  to find  $g^{ab}$ . But at present we simply cannot imagine a way to go from  $g^a$  and  $g^b$  to  $g^{ab}$  without essentially solving the discrete logarithm problem.

**Example 3.3.6.** The following example, taken from McCurley [150], shows how the Diffie-Hellman-Merkel scheme works in a real situation:

- (1) Let  $q = (7^{149} - 1)/6$  and  $p = 2 \cdot 739 \cdot q + 1$ . (It can be shown that both  $p$  and  $q$  are primes.)
- (2) Alice chooses a random number residue  $x$  modulo  $p$ , computes  $7^x \pmod{p}$ , and sends the result to Bob, keeping  $x$  secret.
- (3) B receives
 
$$7^x = 12740218011997394682426924433432284974938204258693162165 \\ 45577352903229146790959986818609788130465951664554581442 \\ 80588076766033781$$
- (4) Bob chooses a random number residue  $y$  modulo  $p$ , computes  $7^y \pmod{p}$ , and sends the result to Alice, keeping  $y$  secret.
- (5) Alice receives
 
$$7^y = 18016228528745310244478283483679989501596704669534669731 \\ 30251217340599537720584759581769106253806921016518486623 \\ 62137934026803049$$
- (6) Now both Alice and Bob can compute the private key  $7^{xy} \pmod{p}$ .

McCurley offered a prize of \$100 in 1989 to the first person to find the private key constructed from the above communication.

**Remark 3.3.3.** McCurley's 129-digit discrete logarithm challenge was actually solved on 25 January 1998 using the NFS method, by two German computer scientists, Damian Weber at the Institut für Techno- und Wirtschaftsmathematik in Kaiserslautern and Thomas F. Denny at the Debis IT Security Services in Bonn.

As we have already mentioned earlier the Diffie-Hellman-Merkle scheme is not intended to be used for actual secure communications, but for key-exchanges. There are, however, several other cryptosystems based on discrete logarithms, that can be used for secure message transmissions.

**(II) The ElGamal Cryptosystem for Secure Communications.** In 1985, ElGamal proposed a public-key cryptosystem based on discrete logarithms:

- (1) A prime  $q$  and a generator  $g \in \mathbb{F}_q^*$  are made public.
- (2) Alice chooses a private integer  $a = a_A \in \{1, 2, \dots, q-1\}$ . This  $a$  is the private decryption key. The public encryption key is  $g^a \in \mathbb{F}_q$ .
- (3) Suppose now Bob wishes to send a message to Alice. He chooses a random number  $b \in \{1, 2, \dots, q-1\}$  and sends Alice the following pair of elements of  $\mathbb{F}_q$ :

$$(g^b, M g^{ab})$$

where  $M$  is the message.

- (4) Since Alice knows the private decryption key  $a$ , she can recover  $M$  from this pair by computing  $g^{ab} \pmod{q}$  and dividing this result into the second element, i.e.,  $Mg^{ab}$ .

**Remark 3.3.4.** Someone who can solve the discrete logarithm problem in  $\mathbb{F}_q$  breaks the cryptosystem by finding the secret decryption key  $a$  from the public encryption key  $g^a$ . In theory, there could be a way to use knowledge of  $g^a$  and  $g^b$  to find  $g^{ab}$  and hence break the cipher without solving the discrete logarithm problem. But as we have already seen in the Diffie-Hellman scheme, there is no known way to go from  $g^a$  and  $g^b$  to  $g^{ab}$  without essentially solving the discrete logarithm problem. So, the ElGamal cryptosystem is equivalent to the Diffie-Hellman key-exchange system.

**(III) The Massey–Omura Cryptosystem for Message Transmissions.** This is another popular cryptosystem based on discrete logarithms; it works in the following way:

- (1) All the users have agreed upon a finite group over a fixed finite field  $\mathbb{F}_q$  with  $q$  a prime power.
- (2) Each user secretly selects a random integer  $e$  between 0 and  $q - 1$  such that  $\gcd(e, q - 1) = 1$ , and computes  $d = e^{-1} \pmod{q - 1}$  by using the extended Euclidean algorithm.
- (3) Now suppose that user Alice wishes to send a secure message  $M$  to user Bob, then they follow the following procedure:
  - (i) Alice first sends  $M^{e_A}$  to Bob,
  - (ii) On receiving Alice's message, Bob sends  $M^{e_A e_B}$  back to Alice (note that at this point, Bob cannot read Alice's message  $M$ ),
  - (iii) Alice sends  $M^{e_A e_B d_A} = M^{e_B}$  to Bob,
  - (iv) Bob then computes  $M^{d_B e_B} = M$ , and hence recovers Alice's original message  $M$ .

### 3.3.6 RSA Public-Key Cryptosystem

In 1978, just shortly after Diffie and Hellman proposed the first public-key exchange protocol at Stanford, three MIT researchers Rivest<sup>12</sup>, Shamir<sup>13</sup> and Adleman<sup>14</sup> proposed the first practical public-key cryptosystem, now widely known as the RSA public-key cryptosystem. The RSA cryptosystem is based on the following assumption:

**RSA Assumption:** It is not so difficult to find two large prime numbers, but it is very difficult to factor a large composite into its prime factorization form.

12



Ronald L. Rivest (1948– ) is currently the Webster Professor of Electrical Engineering and Computer Science in the Department of Electrical Engineering and Computer Science (EECS) at the Massachusetts Institute of Technology (MIT), an Associate Director of the MIT's Laboratory for Computer Science, and a leader of the lab's Cryptography and Information Security Group. He obtained a B.A. in Mathematics from Yale University in 1969, and a Ph.D. in Computer Science from Stanford University in 1974. Professor Rivest is an inventor of the RSA public-key cryptosystem, and a founder of RSA Data Security (now a subsidiary of Security Dynamics). He has worked extensively in the areas of cryptography, computer algorithms, machine learning and VLSI design. (Photo by courtesy of Prof. Rivest.)

13



Adi Shamir (Born 1952) is currently Professor in the Department of Applied Mathematics and Computer Science at the Weizmann Institute of Science, Israel. He obtained his PhD in Computer Science from the Weizmann Institute of Science in 1977, with Prof. Zohar Manna on "Fixedpoints of Recursive Programs", and did his postdoc with Prof. Mike Paterson for a year in Computer Science at Warwick University in England. He participated in developing the RSA public-key cryptosystem, the Fiat-Shamir identification scheme, polynomial secret sharing schemes, visual cryptosystems, lattice attacks on knapsack cryptosystems, differential cryptanalysis, fault attacks on smart cards, algebraic attacks on multivariate cryptosystems and numerous other cryptographic schemes and techniques. (Photo by courtesy of Prof. Shamir.)

14



Leonard Adleman (Born 1945 ) received his BSc in mathematics and PhD in computer science both from the University of California at Berkeley in 1972 and 1976, respectively. He is currently Professor in the Department of Computer Science at the University of Southern California. His main research activities are in theoretical computer science with particular emphasis on the complexity of number theoretic problems. Recently he has also been involved in the development of DNA biological computers. (Photo by courtesy of Prof. Adleman.)



The system works as follows:

$$\left. \begin{aligned} C &\equiv M^e \pmod{N} \\ M &\equiv C^d \pmod{N} \end{aligned} \right\} \quad (3.59)$$

where

- (1)  $M$  is the plaintext.
- (2)  $C$  is the ciphertext.
- (3)  $N = pq$  is the modulus, with  $p$  and  $q$  large and distinct primes.
- (4)  $e$  is the *public* encryption exponent (key) and  $d$  the *private* decryption exponent (key), with  $ed \equiv 1 \pmod{\phi(N)}$ .  $\langle N, e \rangle$  should be made public, but  $d$  (as well as  $\phi(N)$ ) should be kept secret.



**Figure 3.9.** The RSA crypto years: (Left to right) Shamir, Rivest and Adleman (Photo by courtesy of Prof. Adleman)

Clearly, the function  $f : M \rightarrow C$  is a one-way trap-door function, since it is easy to compute by the fast exponentiation method, but its inverse  $f^{-1} : C \rightarrow M$  is difficult to compute, because for those who do not know the private decryption key (the trap-door information)  $d$ , they will have to factor  $n$  and to compute  $\phi(n)$  in order to find  $d$ . However, for those who know  $d$ , then the computation of  $f^{-1}$  is as easy as of  $f$ . This exactly the idea of RSA cryptography.

Suppose now the sender, say, for example, Alice wants to send a message  $M$  to the receiver, say, for example, Bob. Bob will have already chosen a

one-way trapdoor function  $f$  described above, and published his *public-key*  $(e, N)$ , so we can assume that both Alice and any potential adversary know  $(e, N)$ . Alice splits the message  $M$  into blocks of  $\lfloor \log N \rfloor$  bits or less (padded on the right with zeros for the last block), and treats each block as an integer  $x \in \{0, 1, 2, \dots, N-1\}$ . Alice computes

$$y \equiv x^e \pmod{N} \quad (3.60)$$

and transmits  $y$  to Bob. Bob, who knows the private key  $d$ , computes

$$x \equiv y^d \pmod{N} \quad (3.61)$$

where  $ed \equiv 1 \pmod{\phi(N)}$ . An adversary who intercepts the encrypted message should be unable to decrypt it without knowledge of  $d$ . There is no known way of cracking the RSA system without essentially factoring  $N$ , so it is clear that the security of the RSA system depends on the difficulty of factoring  $N$ . Some authors, for example, Woll [259] observed that finding the RSA decryption key  $d$  is random polynomial-time equivalent to factorization. More recently, Pinch [184] showed that an algorithm  $A(N, e)$  for obtaining  $d$  given  $N$  and  $e$  can be turned into an algorithm which obtains  $p$  and  $q$  with positive probability.

**Example 3.3.7.** Suppose the message to be encrypted is “Please wait for me”. Let  $N = 5515596313 = 71593 \cdot 77041$ . Let also  $e = 1757316971$  with  $\gcd(e, N) = 1$ . Then  $d \equiv 1/1757316971 \equiv 2674607171 \pmod{(71593-1)(77041-1)}$ . To encrypt the message, we first translate the message into its numerical equivalent by the letter-digit encoding scheme described in Table 3.4 as follows:

$$M = 1612050119050023010920061518001305.$$

Then we split it into 4 blocks, each with 10 digits, padded on the right with zeros for the last block:

$$M = (M_1, M_2, M_3, M_4) = (1612050119 \ 0500230109 \ 2000061518 \ 0013050000).$$

Now, we have

$$\begin{aligned} C_1 &\equiv 1612050119^{1757316971} \equiv 763222127 \pmod{5515596313} \\ C_2 &\equiv 0500230109^{1757316971} \equiv 1991534528 \pmod{5515596313} \\ C_3 &\equiv 2000061518^{1757316971} \equiv 74882553 \pmod{5515596313} \\ C_4 &\equiv 0013050000^{1757316971} \equiv 3895624854 \pmod{5515596313} \end{aligned}$$

That is,

$$C = (C_1, C_2, C_3, C_4) = (763222127, 1991534528, 74882553, 3895624854).$$

To decrypt the cipher text, we perform:

$$\begin{aligned} M_1 &\equiv 763222127^{2674607171} \equiv 1612050119 \pmod{5515596313} \\ M_2 &\equiv 1991534528^{2674607171} \equiv 500230109 \pmod{5515596313} \\ M_3 &\equiv 74882553^{2674607171} \equiv 2000061518 \pmod{5515596313} \\ M_4 &\equiv 3895624854^{2674607171} \equiv 13050000 \pmod{5515596313} \end{aligned}$$

By padding the necessary zeros on the left of some blocks, we get

$$M = (M_1, M_2, M_3, M_4) = (1612050119 \ 0500230109 \ 2000061518 \ 0013050000)$$

which is “Please wait for me”, the original plaintext message.

**Example 3.3.8.** We now give a reasonably large RSA example. In one of his series of Mathematical Games, Martin Gardner [78] reported an RSA challenge with US\$100 to decrypt the following message  $C$ :

9686961375462206147714092225435588290575999112457\_  
4319874695120930816298225145708356931476622883989\_  
628013391990551829945157815154.

The public key consists of a pair of integers  $(e, N)$ , where  $e = 9007$  and  $N$  is a “random” 129-digit number (called RSA-129):

1143816257578888676692357799761466120102182967212\_  
4236256256184293570693524573389783059712356395870\_  
5058989075147599290026879543541.

The RSA-129 was factored by Derek Atkins, Michael Graff, Arjen K. Lenstra, Paul Leyland et al. on 2 April 1994 to win the \$100 prize offered by RSA in 1977. Its two prime factors are as follows:

3490529510847650949147849619903898133417764638493\_  
387843990820577,  
3276913299326670954996198819083446141317764296799\_  
2942539798288533.

They used the double large prime variation of the Multiple Polynomial Quadratic Sieve (MPQS) factoring method. The sieving step took approximately 5000 mips years, and was carried out in 8 months by about 600 volunteers from more than 20 countries, on all continents except Antarctica. As we have explained in the previous example, to encrypt an RSA-encrypted message, we only need to use the public key  $(N, e)$  to compute

$$x^e \equiv y \pmod{N}.$$

But decrypting an RSA-message requires factorization of  $N$  if one does not know the secret decryption key. This means that if we can factor  $N$ , then we can compute the secret key  $d$ , and get back the original message by calculating

$$y^d \equiv x \pmod{N}.$$

Since now we know the prime factorization of  $N$ , it is trivial to compute the secret key  $d = 1/e \bmod \phi(N)$ , which in fact is

```
1066986143685780244428687713289201547807099066339_
3786280122622449663106312591177447087334016859746_
2306553968544513277109053606095.
```

So we shall be able to compute

$$C^d \equiv M \pmod{N}$$

without any problem. To use the fast exponential method to compute  $C^d \bmod N$ , we first write  $d$  in its binary form  $d_1 d_2 \cdots d_{\text{size}}$  (where size is the number of the bits of  $d$ ) as follows:

```
d = d1d2⋯d426 =
100111011001111110010100110010001000001000001110100111100100110_
01001111010011100000000000001111111010000110101011000101110111_
01010000111110110000001000001110110101010111101010100111110110_
11010000111110100000011110100110001011001011001101001010001100_
100111010110000101110100101011010000011100000001110001110101010_
011011101000111101001110001101011010101010010011101010001001111_
000000100111010011000110111110101100100011001111
```

and perform the following computation:

```
M ← 1
for i from 1 to 426 do
  M ← M2 mod N
  if di = 1 then M ← M · C mod N
print M
```

which gives the plaintext  $M$ :

```
2008050013010709030023151804190001180500191721050_
11309190800151919090618010705
```

and hence the original message:

THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE

via the encoding alphabet  $\sqcup = 00, A = 01, B = 02, \dots, Z = 26$ . Of course, by the public encryption key  $e = 9007$ , we can compute  $M^e \equiv C \pmod{N}$ ; first write  $e$  in the binary form  $e = e_1 e_2 \cdots e_{14} = 10001100101111$ , then perform the following procedure:

```
C ← 1
for i from 1 to 14 do
  C ← C2 mod N
  if ei = 1 then C ← C · M mod N
print C
```

which gives the encrypted text  $C$  at the beginning of this example:

```
9686961375462206147714092225435588290575999112457_
4319874695120930816298225145708356931476622883989_
628013391990551829945157815154.
```

**Remark 3.3.5.** In fact, anyone who can factor the integer RSA-129 can decrypt the message. Thus, decrypting the message is essentially factoring the 129-digit integer. The factorization of RSA-129 implies that it is possible to factor a random 129-digit integer. It should be also noted that on 10 April 1996, Arjen Lenstra et al. also factored the following RSA-130:

```
1807082088687404805951656164405905566278102516769_
4013491701270214500566625402440483873411275908123_
03371781887966563182013214880557
```

which has the following two prime factors:

```
3968599945959745429016112616288378606757644911281_
0064832555157243,
4553449864673597218840368689727440886435630126320_
5069600999044599.
```

This factorization was found using the Number Field Sieve (NFS) factoring algorithm, and beats the above mentioned 129-digit record by the Quadratic Sieve (QS) factoring algorithm. The amount of computer time spent on this 130-digit NFS-record is only a fraction of what was spent on the old 129-digit QS-record. More recently a group led by Peter Montgomery and Herman te Riele found in February 1999 that the RSA-140:

```
2129024631825875754749788201627151749780670396327_
7216278233383215381949984056495911366573853021918_
316783107387995317230889569230873441936471
```

can be written as the product of two 70-digit primes:

```
3398717423028438554530123627613875835633986495969_
597423490929302771479,
6264200187401285096151654948264442219302037178623_
509019111660653946049.
```

This factorization was found using the Number Field Sieve (NFS) factoring algorithm, and beats the 130-digit record that was set in April 1996, also with the help of NFS. The amount of computer time spent on this new 140-digit NFS-record is prudently estimated to be equivalent to 2000 mips years. For the old 130-digit NFS-record, this effort is estimated to be 1000 mips years (Te Riele [205]). Even more recently (August 26, 1999), Herman te Riele and Stefania Cavallar et al. successfully factored (again using NFS) the RSA-155, a number with 155 digits and 512 bits, which can be written as the product of two 78-digit primes:

1026395928297411057720541965739916759007165678080\_  
 38066803341933521790711307779,  
 10660348838016845482092722036001287867920795857598\_  
 9291522270608237193062808643.

So, it follows from the above factorization results that

**Corollary 3.3.1.** The composite number (i.e., the modulus)  $N$  used in the RSA cryptosystem should have more than 155 decimal digits.

**Exercise 3.3.6.** Below is an encrypted message (consisting of two blocks  $C_1$  and  $C_2$ ):

4660 4906 4350 6009 6392 3911 2238 7112  
 0237 3603 9163 4700 8276 8243 4103 8329  
 6685 0734 6202 7217 9820 0029 7925 0670  
 8833 7283 5678 0453 2383 8911 4071 9579  
  
 6506 4096 9385 1106 9741 5283 1334 2475  
 3966 4897 8551 7358 1383 6777 9635 0373  
 8147 2092 8779 3861 7878 7818 9741 5743  
 9185 7183 6081 9612 4160 0934 3883 0158

The public key used to encrypt the message is  $(e, N)$ , where  $e = 9137$  and  $N$  is the following RSA-129:

1143816257578888676692357799761466120102182967212\_  
 4236256256184293570693524573389783059712356395870\_  
 5058989075147599290026879543541.

Decrypt the message. (Note that in the encryption process if  $\gcd(M_i, N) \neq 1$  for  $i = 1, 2$ , some dummy letter may be added to the end of  $M_i$  to make  $\gcd(M_i, N) = 1$ .)

Let us now consider a more general and more realistic case of secure communications in a computer network with  $n$  nodes. It is apparent that there are

$$\binom{n}{2} = n(n-1)/2$$

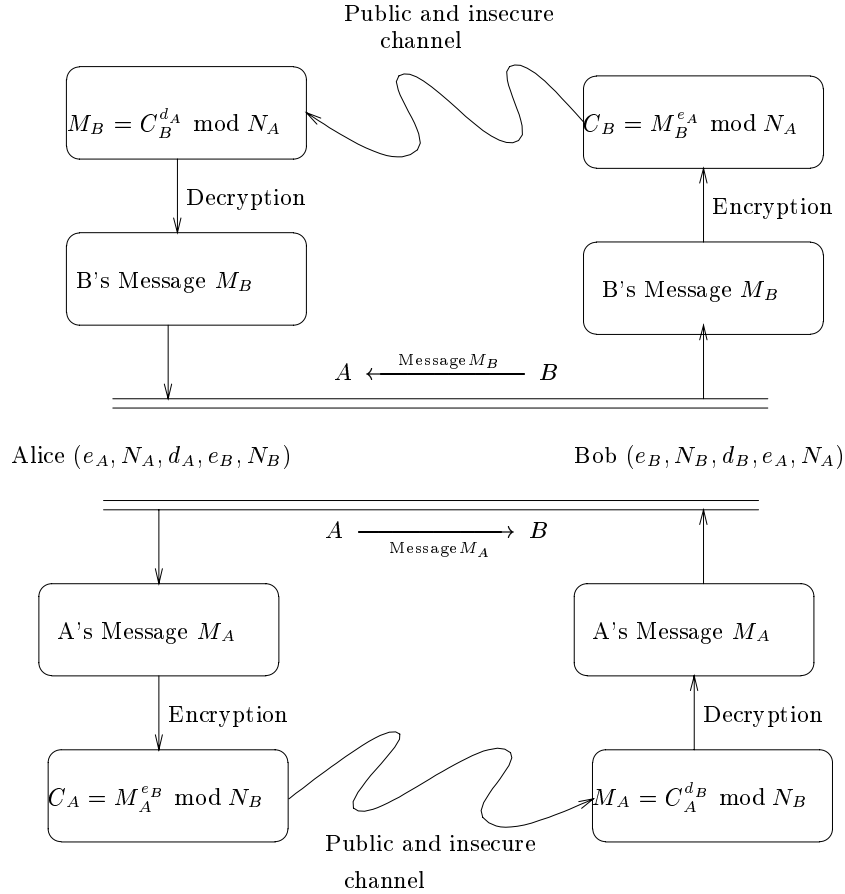
ways of communicating between two nodes in the network. Suppose one of the nodes (users), say, Alice (A), wants to send a secure message  $M$  to another node, say, Bob (B), or vice versa. Then A uses B's encryption key  $e_B$  to encrypt her message  $M_A$

$$C_A = M_A^{e_B} \bmod N_B \quad (3.62)$$

and sends the encrypted message  $C$  to B; on receiving A's message  $M_A$ , B uses his own decryption key  $d_B$  to decrypt A's message  $C$ :

$$M_A = C_A^{d_B} \bmod N_B. \quad (3.63)$$

Since only B has the decryption key  $d_B$ , only B (at least from a theoretical point of view) can recover the original message. B can of course send a secure message M to A in a similar way. Figure 3.10 shows diagrammatically the idea of secure communication between any two parties, say, for example, Alice and Bob.



**Figure 3.10.** The RSA secure communications between two parties

A better example of a trap-door one-way function of the form used in the RSA cryptosystem would use Carmichael's  $\lambda$ -function rather than Euler's  $\phi$ -function, and is as follows:

$$y = f(x) \equiv x^k \pmod{N} \quad (3.64)$$

where

$$\left. \begin{aligned} N &= pq \quad (p \text{ and } q \text{ are two large primes}), \\ k &> 1, \quad \gcd(k, \lambda) = 1, \\ \lambda(N) &= \text{lcm}(p-1, q-1) = \frac{(p-1)(q-1)}{\gcd(p-1, q-1)}. \end{aligned} \right\} \quad (3.65)$$

We assume that  $k$  and  $N$  are publicly known but  $p, q$  and  $\lambda(N)$  are not. The inverse function of  $f(x)$  is defined by

$$x = f^{-1}(y) \equiv y^{k'} \pmod{N} \quad \text{with } kk' \equiv 1 \pmod{\lambda}. \quad (3.66)$$

To show it works, we see

$$\begin{aligned} x &\equiv y^{k'} \equiv (x^k)^{k'} \equiv x^{kk'} \equiv x^{m \cdot \lambda(N) + 1} \\ &\equiv (x^{\lambda(N)})^m \cdot x \equiv 1^m \cdot x \quad (\text{by Carmichael's theorem}) \\ &\equiv x \end{aligned}$$

It should be easy to compute  $f^{-1}(y) \equiv y^{k'} \pmod{N}$  if  $k'$  is known, provided that  $f^{-1}(y)$  exists (note that  $f^{-1}(y)$  may not exist). The assumption underlying the RSA cryptosystem is that it is hard to compute  $f^{-1}(y)$  without knowing  $k'$ . However, the knowledge of  $p, q$  or  $\lambda(N)$  makes it easy to compute  $k'$ .

**Example 3.3.9.** Suppose we wish to encrypt the plaintext message

NATURAL NUMBERS ARE MADE BY GOD.

We first translate all the letters in the message into their numerical equivalents as in Table 3.4. Then we split the message into, for example, four message blocks, each with 15 digits as follows:

$\langle 140020211801120, 014211302051800, 011805001301040, 500022500071504 \rangle$ .

and perform the following computation steps:

- (1) Select two primes  $p$  and  $q$ , compute  $N = pq$  and  $\lambda(N)$ :

$$p = 440334654777631,$$

$$q = 145295143558111$$

$$N = pq = 63978486879527143858831415041$$

$$\lambda(N) = 710872076439183980322589770.$$

- (2) Determine the keys  $k$  and  $k'$ : we try to factorize  $m\lambda(N) + 1$  for  $m = 1, 2, 3, \dots$  until we find a “good” factorization that can be used to obtain suitable  $k$  and  $k'$ :



$\lambda(N) + 1 = 1193 \cdot 2990957 \cdot 209791523 \cdot 17107 \cdot 55511$   
 $2\lambda(N) + 1 = 47 \cdot 131 \cdot 199 \cdot 3322357 \cdot 1716499 \cdot 203474209$   
 $3\lambda(N) + 1 = 674683 \cdot 1696366781 \cdot 297801601 \cdot 6257$   
 $4\lambda(N) + 1 = 17 \cdot 53 \cdot 5605331 \cdot 563022035211575351$   
 $5\lambda(N) + 1 = 17450633 \cdot 13017248387079301 \cdot 15647$   
 $6\lambda(N) + 1 = 1261058128567 \cdot 49864411 \cdot 2293 \cdot 29581$   
 $7\lambda(N) + 1 = 19 \cdot 261900238688120413803059389$   
 $8\lambda(N) + 1 = 15037114930441 \cdot 378195992902921$   
 $9\lambda(N) + 1 = 11 \cdot 13200581 \cdot 8097845885549501 \cdot 5441$   
 $10\lambda(N) + 1 = 7108720764391839803225897701$   
 $11\lambda(N) + 1 = 2131418173 \cdot 7417510211 \cdot 494603657$   
 $12\lambda(N) + 1 = 4425033337657 \cdot 1927774158146113$   
 $13\lambda(N) + 1 = 23 \cdot 6796296973884340591 \cdot 59120027$   
 $14\lambda(N) + 1 = 14785772846857861 \cdot 673093599721$   
 $15\lambda(N) + 1 = 500807 \cdot 647357777401277 \cdot 17579 \cdot 1871.$

Suppose now we wish to use the 15th factorization  $15\lambda(N) + 1$  to obtain

$$\langle k, k' \rangle = \langle 17579, 606580644324919489438469 \rangle$$

such that  $kk' = 1 + 15\lambda(N)$ .

- (3) Encrypt the message  $x \mapsto x^k \bmod N = y$  (using the fast modular exponentiation method, for example, Algorithm 2.1.1):

$140020211801120^{17579} \bmod N = 60379537366647508826042726177$   
 $014211302051800^{17579} \bmod N = 47215464067987497433568498485$   
 $011805001301040^{17579} \bmod N = 20999327573397550148935085516$   
 $500022500071504^{17579} \bmod N = 37746963038639759803119392704.$

- (4) Decrypt the message  $y \mapsto y^{k'} \bmod N = x^{kk'} \bmod N = x$  (again using, for example, Algorithm 2.1.1):

$60379537366647508826042726177^{k'} \bmod N = 140020211801120$   
 $47215464067987497433568498485^{k'} \bmod N = 014211302051800$   
 $20999327573397550148935085516^{k'} \bmod N = 011805001301040$   
 $37746963038639759803119392704^{k'} \bmod N = 500022500071504$

where  $k' = 606580644324919489438469$ .

**Remark 3.3.6.** Compared with the conventional cryptosystems such as the Data Encryption Standard (DES), the RSA system is *very* slow. For example, the DES, when implemented with special-purpose chips, can be run at speeds of tens of millions of bits per second, and even in software on modest size machines can encrypt on the order of  $10^5$  bits per second, whereas the RSA system, when implemented with the best possible special purpose chips, can only encrypt at the rate of  $10^4$  or  $2 \cdot 10^4$  bits per second, and software implementations are limited to something on the order of  $10^2$  bits per second. Thus, the RSA system is about 100 to 1000 times slower than conventional cryptosystems.

Now we are in a position to give a brief discussion of the existence of the inverse function  $f^{-1}(y)$  defined in (3.66) for all  $y$ . Let us first introduce a useful result (Riesel [207]):

**Theorem 3.3.1.** If  $N$  is a product of distinct primes, then for all  $a$ ,

$$a^{\lambda(N)+1} \equiv a \pmod{N}. \quad (3.67)$$

Note that if  $N$  contains multiple prime factors, then (3.67) need no longer be true; say, for example, let  $N = 12 = 2^2 \cdot 3$ , then  $9^{\lambda(12)+1} = 9^3 \equiv 9 \pmod{12}$ , but  $10^{\lambda(12)+1} = 10^3 \equiv 4 \not\equiv 10 \pmod{12}$ . Now, let  $k$  and  $N$  have been chosen suitably as follows:

$$N = pq, \quad \text{with } p, q \text{ distinct primes} \quad (3.68)$$

$$a^{kk'} \equiv a \pmod{N}, \quad \text{for all } a. \quad (3.69)$$

Then, by Theorem 3.3.1, the inverse function  $f^{-1}(y)$ , defined in (3.66), exists for all  $y$ . It follows immediately from (3.67) that

$$a^{m\lambda(N)+1} \equiv a \pmod{N}, \quad (3.70)$$

which is exactly the form needed in a RSA cryptosystem. For an arbitrary integer  $N$  and  $m \geq 1$ , a necessary and sufficient condition for (3.70) to have a solution  $a$  is that (private communications with William Freeman)

$$\gcd(a^2, N) \mid a, \quad (3.71)$$

or equivalently,

$$\gcd(a, N/d) = 1, \quad \text{where } d = \gcd(a, N). \quad (3.72)$$

More generally (private communications with Peter Pleasants and Carl Pomerance), a necessary and sufficient condition for

$$a^{m\lambda(N)+k} \equiv a^k \pmod{N} \quad (3.73)$$

is

$$\gcd(a^{k+1}, N) \mid a^k, \quad (3.74)$$

or equivalently,

$$\gcd(a, N/d) = 1, \quad \text{where } d = \gcd(a^k, N). \quad (3.75)$$

The proof for the more general case is as follows: Let  $p$  be prime and  $p^\alpha \parallel N$ . Let  $\beta$  be such that  $p^\beta \parallel a$ . We assume that  $p \mid N$ , that is  $\alpha > 0$ . There are three cases:

(1)  $\beta = 0$ : we have  $a^{m\lambda(N)+k} \equiv a^k \pmod{p^\alpha}$ , by Euler's theorem,

- (2)  $0 < k\beta < \alpha$ : we have  $a^t \not\equiv a^k \pmod{p^\alpha}$  for all  $t > k$ , obviously,  
 (3)  $k\beta \geq \alpha$ : we have  $a^t \equiv a^k \pmod{p^\alpha}$  for all  $t > k$ , obviously.

We conclude that  $a^{m\lambda(N)+k} \equiv a^k \pmod{N}$  if and only if we are never in the second case for all primes  $p \mid N$ . Never being in the second case is equivalent to the condition  $\gcd(a^{k+1}, N) \mid a^k$ .

Now let us return to the construction of a good trapdoor function (Brent [37]) used in RSA:

**Algorithm 3.3.1 (Construction of trapdoor functions).** This algorithm constructs the trapdoor function and generates both the public and the secret keys suitable for RSA cryptography:

- [1] Use Algorithm 3.3.3 or Algorithm 3.3.2 to find two large primes  $p$  and  $q$ , each with at least 100 digits such that:
  - [1-1]  $|p - q|$  is large;
  - [1-2]  $p \equiv -1 \pmod{12}, q \equiv -1 \pmod{12}$ ;
  - [1-3] The following values of  $p', p'', q'$  and  $q''$  are all primes:
 
$$p' = (p - 1)/2,$$

$$p'' = (p + 1)/12,$$

$$q' = (q - 1)/2,$$

$$q'' = (q + 1)/12.$$
- [2] Compute  $N = pq$  and  $\lambda = 2p'q'$ .
- [3] Choose a random integer  $k$  relatively prime to  $\lambda$  such that  $k - 1$  is not a multiple of  $p'$  or  $q'$ .
- [4] Apply the extended Euclidean algorithm to  $k$  and  $\lambda$  to find  $k'$  and  $\lambda'$  such that  $0 < k' < \lambda$  and
 
$$kk' + \lambda\lambda' = 1.$$
- [5] Destroy all evidence of  $p, q, \lambda$  and  $\lambda'$ .
- [6] Make  $(k, N)$  public but keep  $k'$  secret.

It is clear that the most important task in the construction of RSA cryptosystems is to find two large primes, say each with at least 100 digits. An algorithm for finding two 100 digit primes can be described as follows:

**Algorithm 3.3.2 (Large prime generation).** This algorithm generates prime numbers with 100 digits; it can be modified to generate any length of the required prime numbers:

- [1] (Initialization) Randomly generate an odd integer  $n$  with say, for example, 100 digits;

- [2] (Primality Testing – Probabilistic Method) Use a combination of the Miller–Rabin test and a Lucas test to determine if  $n$  is a probable prime. If it is, goto Step [3], else goto Step [1] to get another 100-digit odd integer.
- [3] (Primality Proving – Elliptic Curve Method) Use the elliptic curve method to verify whether or not  $n$  is indeed a prime. If it is, then report that  $n$  is prime, and save it for later use; or otherwise, goto Step [1] to get another 100-digit odd integer.
- [4] (done?) If you need more primes, goto Step [1], else terminate the algorithm.

How many primes with 100 digits do we have? By Chebyshev's inequality (1.167), if  $N$  is large, then

$$0.92129 \frac{N}{\ln N} < \pi(N) < 1.1056 \frac{N}{\ln N}. \quad (3.76)$$

Hence

$$0.92129 \frac{10^{99}}{\ln 10^{99}} < \pi(10^{99}) < 1.1056 \frac{10^{99}}{\ln 10^{99}},$$

$$0.92129 \frac{10^{100}}{\ln 10^{100}} < \pi(10^{100}) < 1.1056 \frac{10^{100}}{\ln 10^{100}}.$$

The difference  $\pi(10^{100}) - \pi(10^{99})$  will give the number of primes with exactly 100 digits, we have

$$3.596958942 \cdot 10^{97} < \pi(10^{100}) - \pi(10^{99}) < 4.076949099 \cdot 10^{97}.$$

The above algorithm for large prime generation depends on primality testing and proving. However, there are methods which do not rely on primality testing and proving. One such method is based on Pocklington's theorem (Theorem 2.2.19), that can automatically lead to primes, say with 100 digits (Ribenoim [199]). We re-state the theorem in a slightly different way as follows:

**Theorem 3.3.2.** Let  $p$  be an odd prime,  $k$  a natural number such that  $p$  does not divide  $k$  and  $1 < k < 2(p+1)$  and let  $N = 2kp + 1$ . Then the following conditions are equivalent:

- (1)  $N$  is prime.
- (2) There exists a natural number  $a$ ,  $2 \leq a < N$ , such that

$$a^{kp} \equiv -1 \pmod{N}, \text{ and} \quad (3.77)$$

$$\gcd(a^k + 1, N) = 1. \quad (3.78)$$

**Algorithm 3.3.3 (Large prime number generation).** This algorithm, based on Theorem 3.3.2, generates large prime numbers without the use of primality testing:

- [1] Choose, for example, a prime  $p_1$  with  $d_1 = 5$  digits. Find  $k_1 < 2(p_1 + 1)$  such that  $p_2 = 2k_1p_1 + 1$  has  $d_2 = 2d_1 = 10$  digits or  $d_2 = 2d_1 - 1 = 9$  digits and there exists  $a_1 < p_2$  satisfying the conditions  $a_1^{k_1p_1} \equiv -1 \pmod{p_2}$  and  $\gcd(a_1^{k_1} + 1, p_2) = 1$ . By Pocklington's Theorem,  $p_2$  is prime.
- [2] Repeat the same procedure starting from  $p_2$  to obtain the primes  $p_3, p_4, \dots$ . In order to produce a prime with 100 digits, the process must be iterated five times. In the last step,  $k_5$  should be chosen so that  $2k_5p_5 + 1$  has 100 digits.

As pointed out in Ribenboim [199], for all practical purposes, the above algorithm for producing primes of a given size will run in polynomial time, even though this has not yet been supported by a proof.

According to the Prime Number Theorem, the probability that a randomly chosen integer in  $[1, N]$  is prime is  $\sim 1/\ln N$ . Thus, the expected number of random trials required to find  $p$  (or  $p'$ , or  $p''$ ; assume that  $p$ ,  $p'$ , and  $p''$  are independent) is conjectured to be  $\mathcal{O}((\log N)^3)$ . Based on this assumption, the expected time required to construct the above one-way trap-door function is  $\mathcal{O}((\log N)^6)$ .

Finally, in this subsection, we shall give a brief account of some possible attacks on the RSA cryptosystem. We restrict ourselves to the simplified version of RSA system. Let  $N$ , the RSA modulus, be the product of two primes  $p$  and  $q$ . Let also  $e$  and  $d$  be two positive integers satisfying  $ed \equiv 1 \pmod{\phi(N)}$ , where  $\phi(N) = (p-1)(q-1)$  is the order of the multiplicative group  $(\mathbb{Z}/N\mathbb{Z})^*$ . Recall that the RSA system works as follows:

$$\left. \begin{aligned} C &\equiv M^e \pmod{N} \\ M &\equiv C^d \pmod{N} \end{aligned} \right\}$$

where  $\langle N, e \rangle$  is the public key for encryption, and  $\langle N, d \rangle$  the private key for decryption. From an cryptanalytic point of view we would like to know that given the triple  $\langle N, e, C \rangle$ , how hard (or how many ways) an enemy cryptanalyst can break the RSA system. In what follows, we shall present some possible ways of cracking the RSA scheme.

- (1) Factoring  $N$ . The most obvious way of breaking the RSA system is to factor  $N$ , since if an enemy cryptanalyst could factor  $N$ , then he could determine  $\phi(N) = (p-1)(q-1)$  and hence the private key  $d$ . But this is not easy, since integer factorization is a computationally intractable problem.
- (2) Computing  $\phi(N)$  without factoring  $N$ . It is also obvious that if an enemy cryptanalyst could compute  $\phi(N)$  then he could break the system by computing  $d$  as the multiplicative inverse of  $e$  modulo  $\phi(N)$ . However, the knowledge of  $\phi(N)$  can lead to an easy way of factoring  $N$ , since

$$\begin{aligned}
p + q &= n - \phi(N) + 1, \\
(p - q)^2 &= (p + q)^2 - 4n, \\
p &= \frac{1}{2} [(p + q) + (p - q)], \\
q &= \frac{1}{2} [(p + q) - (p - q)].
\end{aligned}$$

Thus, breaking the RSA system by computing  $\phi(N)$  is no easier than breaking the system by factoring  $N$ .

- (3) Determining  $d$  without factoring  $n$  or computing  $\phi(N)$ . If  $N$  is large and  $d$  is chosen from a large set, then a cryptanalyst should not be able to determine  $d$  any easier than he can factor  $N$ . Again, a knowledge of  $d$  enables  $N$  to be factored, since once  $d$  is known,  $ed - 1$  (a multiple of  $\phi(N)$ ) can be calculated;  $N$  can be factored using any multiple of  $\phi(N)$ .
- (4) Computing the  $e^{\text{th}}$  root of  $C$  modulo  $N$ . Clearly, the RSA decryption process is just the computation of the  $e^{\text{th}}$  root of  $C$  modulo  $N$ . That is, the decryption problem is just the *root finding problem*. It is evident that in the following congruence

$$C \equiv M^e \pmod{N},$$

once  $\langle N, e, C \rangle$  is given, we could try substituting  $M = 0, 1, 2, \dots$  until a correct  $M$  is found. In theory, it is possible to enumerate all elements of  $(\mathbb{Z}/N\mathbb{Z})^*$ , since  $(\mathbb{Z}/N\mathbb{Z})^*$  is a finite set, but in practice, it is impossible when  $N$  is large. However, if  $\phi(N)$  is known, then we can compute the  $e^{\text{th}}$  root of  $C$  modulo  $N$  fairly easily (see Algorithm 2.4.8 in Chapter 2).

So, all the above obvious methods of breaking the RSA system are closely related to the integer factorization problem. In fact, Rivest, Shamir and Adleman [209] conjectured that

**Conjecture 3.3.1 (RSA conjecture).** Any method of breaking the RSA cryptosystem must be as difficult as factoring.

There are some other possible attacks on the RSA cryptosystem, which include:

- (1) Wiener's attack [253] on the short RSA private-key. It is important that the private-key  $d$  should be large (nearly as many bits as the modulus  $N$ ); otherwise, there is an attack due to Wiener and based on properties of continued fractions, that can find the private-key  $d$  in time polynomial in the length of the modulus  $N$ , and hence decrypt the message.
- (2) Iterated encryption or fixed-point attack (Meijer [154] and Pinch [184]): Suppose  $e$  has order  $r$  in the multiplicative group modulo  $\lambda(N)$ . Then  $e^r \equiv 1 \pmod{\lambda(N)}$ , so  $M^{e^r} \equiv M \pmod{N}$ . This is just the  $r^{\text{th}}$  iterate of the encryption of  $M$ . So we must ensure that  $r$  is large.

It is interesting to note that the attacks discovered so far mainly illustrate the pitfalls to be avoided when implementing RSA. RSA will be still secure if the parameters such as  $p$ ,  $q$ ,  $e$ , and  $d$  are properly chosen. Readers who wish to know more information about the attacks on the RSA cryptosystem are suggested to consult Boneh's recent paper "Twenty Years of Attacks on the RSA Cryptosystem" [30], as well as an earlier paper by Rivest [208].

### 3.3.7 Quadratic Residuosity Cryptosystems

The RSA cryptosystem discussed in the previous subsection is *deterministic* in the sense that under a fixed public-key, a particular plaintext  $M$  is always encrypted to the same ciphertext  $C$ . Some of the drawbacks of a deterministic scheme are:

- (1) It is not secure for all probability distributions of the message space. For example, in RSA encryption, the messages 0 and 1 always get encrypted to themselves, and hence are easy to detect.
- (2) It is easy to obtain some partial information of the secret key  $(p, q)$  from the public modulus  $n$  (assume that  $n = pq$ ). For example, when the least-significant digit of  $n$  is 3, then it is easy to obtain the partial information that the least-significant digits of  $p$  and  $q$  are either 1 and 3 or 7 and 9, as indicated as follows:

$$\begin{array}{ll} 183 = 3 \cdot 61 & 253 = 11 \cdot 23 \\ 203 = 7 \cdot 29 & 303 = 3 \cdot 101 \\ 213 = 3 \cdot 71 & 323 = 17 \cdot 19. \end{array}$$

- (3) It is sometimes easy to compute partial information about the plaintext  $M$  from the ciphertext  $C$ . For example, given  $(C, e, n)$ , the Jacobi symbol of  $M$  over  $n$  can be easily deduced from  $C$ :

$$\left(\frac{C}{n}\right) = \left(\frac{M^e}{n}\right) \left(\frac{M}{n}\right)^e = \left(\frac{M}{n}\right).$$

- (4) It is easy to detect when the same message is sent twice.

Probabilistic encryption, or randomized encryption, however, utilizes randomness to attain a strong level of security, namely, the *polynomial security* and *semantic security*, defined as follows:

**Definition 3.3.2.** A public-key encryption scheme is said to be *polynomially* secure if no passive adversary can, in expected polynomial time, select two plaintexts  $M_1$  and  $M_2$  and then correctly distinguish between encryptions of  $M_1$  and  $M_2$  with probability significantly greater than  $1/2$ .

**Definition 3.3.3.** A public-key encryption scheme is said to be *semantically secure* if, for all probability distributions over the message space, whatever a passive adversary can compute in expected polynomial time about the plaintext given the ciphertext, it can also be computed in expected polynomial time without the ciphertext.

Intuitively, a public-key encryption scheme is semantically secure if the ciphertext does not leak any partial information whatsoever about the plaintext that can be computed in expected polynomial time. That is, given  $(C, e, n)$ , it should be intractable to recover any information about  $M$ . Clearly, a public-key encryption scheme is semantically secure if and only if it is polynomially secure.

In this subsection, we shall introduce a semantically secure cryptosystem based on the *quadratic residuosity problem*. Recall that an integer  $a$  is a quadratic residue modulo  $n$ , denoted by  $a \in Q_n$ , if  $\gcd(a, n) = 1$  and there exists a solution  $x$  to the congruence  $x^2 \equiv a \pmod{n}$ , otherwise  $a$  is a quadratic nonresidue modulo  $n$ , denoted by  $a \in \overline{Q}_n$ . The Quadratic Residuosity Problem may be stated as:

Given positive integers  $a$  and  $n$ , decide whether or not  $a \in Q_n$ .

It is believed that solving QRP is equivalent to computing the prime factorization of  $n$ , so it is computationally infeasible. We have seen in Subsection 1.6.6 of Chapter 1 that if  $n$  is prime then

$$a \in Q_n \iff \left(\frac{a}{n}\right) = 1, \quad (3.79)$$

and if  $n$  is composite, then

$$a \in Q_n \implies \left(\frac{a}{n}\right) = 1, \quad (3.80)$$

but

$$a \in Q_n \not\Leftarrow \left(\frac{a}{n}\right) = 1, \quad (3.81)$$

however

$$a \in \overline{Q}_n \Leftarrow \left(\frac{a}{n}\right) = -1. \quad (3.82)$$

Let  $J_n = \{a \in (\mathbb{Z}/n\mathbb{Z})^* : \left(\frac{a}{n}\right) = 1\}$ , then  $\tilde{Q}_n = J_n - Q_n$ . Thus,  $\tilde{Q}_n$  is the set of all pseudosquares modulo  $n$ ; it contains those elements of  $J_n$  that do not belong to  $Q_n$ . Readers may wish to compare this result to Fermat's little theorem discussed in Subsection 1.6.3 of Chapter 1 namely (assuming  $\gcd(a, n) = 1$ ),

$$n \text{ is prime} \implies a^{n-1} \equiv 1 \pmod{n}, \quad (3.83)$$

but

$$n \text{ is prime} \not\Leftarrow a^{n-1} \equiv 1 \pmod{n}, \quad (3.84)$$

however



$$n \text{ is composite} \iff a^{n-1} \not\equiv 1 \pmod{n}. \quad (3.85)$$

The Quadratic Residuosity Problem can then be further restricted to:

Given a composite  $n$  and an integer  $a \in J_n$ , decide whether or not  $a \in Q_n$ .

For example, when  $n = 21$ , we have  $J_{21} = \{1, 4, 5, 16, 17, 20\}$  and  $Q_{21} = \{1, 4, 16\}$ , thus  $\tilde{Q}_{21} = \{5, 17, 20\}$ . So, the QRP problem for  $n = 21$  is actually to distinguish squares  $\{1, 4, 16\}$  from pseudosquares  $\{5, 17, 20\}$ . The only method we know for distinguishing squares from pseudosquares is to factor  $n$ ; since integer factorization is computationally infeasible, the QRP problem is computationally infeasible. In what follows, we shall present a cryptosystem whose security is based on the infeasibility of the Quadratic Residuosity Problem; it was first proposed by Goldwasser and Micali [88] in 1984, under the term *probabilistic encryption*.

**Algorithm 3.3.4 (Quadratic residuosity based cryptography).** This algorithm uses the randomized method to encrypt messages and is based on the quadratic residuosity problem (QRP). The algorithm divides into three parts: key generation, message encryption and decryption.

- [1] **Key generation:** Both Alice and Bob should do the following to generate their public and secret keys:
  - [1-1] Select two large distinct primes  $p$  and  $q$ , each with roughly the same size, say, each with  $\beta$  bits.
  - [1-2] Compute  $n = pq$ .
  - [1-3] Select a  $y \in \mathbb{Z}/n\mathbb{Z}$ , such that  $y \in \overline{Q}_n$  and  $\left(\frac{y}{n}\right) = 1$ . ( $y$  is thus a pseudosquare modulo  $n$ ).
  - [1-4] Make  $(n, y)$  public, but keep  $(p, q)$  secret.
- [2] **Encryption:** To send a message to Alice, Bob should do the following:
  - [2-1] Obtain Alice's public-key  $(n, y)$ .
  - [2-2] Represent the message  $m$  as a binary string  $m = m_1m_2 \cdots m_k$  of length  $k$ .
  - [2-3] For  $i$  from 1 to  $k$  do
    - [i] Choose at random an  $x \in (\mathbb{Z}/n\mathbb{Z})^*$  and call it  $x_i$ .
    - [ii] Compute  $c_i$ :

$$c_i = \begin{cases} x_i^2 \bmod n, & \text{if } m_i = 0, \quad (\text{r.s.}) \\ yx_i^2 \bmod n, & \text{if } m_i = 1, \quad (\text{r.p.s.}), \end{cases} \quad (3.86)$$

where r.s. and r.p.s. represent random square and random pseudosquare, respectively.

[iii] Send the  $k$ -tuple  $c = (c_1, c_2, \dots, c_k)$  to Alice. (Note first that each  $c_i$  is an integer with  $1 \leq c_i < n$ . Note also that since  $n$  is a  $2\beta$ -bit integer, it is clear that the ciphertext  $c$  is a much longer string than the original plaintext  $m$ .)

[3] **Decryption:** To decrypt Bob's message, Alice should do the following:

[3-1] For  $i$  from 1 to  $k$  do

[i] Evaluate the Legendre symbols:

$$\left. \begin{aligned} e'_i &= \left( \frac{c_i}{p} \right) \\ e''_i &= \left( \frac{c_i}{q} \right) \end{aligned} \right\} \quad (3.87)$$

[ii] Compute  $m_i$ :

$$m_i = \begin{cases} 0, & \text{if } e'_i = e''_i = 1 \\ 1, & \text{if otherwise.} \end{cases} \quad (3.88)$$

That is,  $m_i = 0$  if  $c_i \in Q_n$ , otherwise,  $m_i = 1$ . otherwise, set  $m_i = 1$ .

[3-2] Finally, get the decrypted message  $m = m_1 m_2 \dots m_k$ .

**Remark 3.3.7.** The above encryption scheme has the following interesting features:

- (1) The encryption is random in the sense that the same bit is transformed into different strings depending on the choice of the random number  $x$ . For this reason, it is called *probabilistic* (or *randomized*) encryption.
- (2) Each bit is encrypted as an integer modulo  $n$ , and hence is transformed into a  $2\beta$ -bit string.
- (3) It is semantically secure against any threat from a polynomially bounded attacker, provided that the QRP is hard.

**Exercise 3.3.7.** Show that Algorithm 3.3.4 takes  $\mathcal{O}(\beta^2)$  time to encrypt each bit and  $\mathcal{O}(\beta^3)$  time to decrypt each bit.

**Example 3.3.10.** In what follows we shall give an example of how Bob can send the message "HELP ME" to Alice using the above cryptographic method. We use the binary equivalents of letters as defined in Table 3.5. Now both Alice and Bob proceed as follows:

[1] **Key Generation:**

[1-1] Alice chooses  $(n, y) = (21, 17)$  as a public key, where  $n = 21 = 3 \cdot 7$  is a composite, and  $y = 17 \in \tilde{Q}_{21}$  (since  $17 \in J_{21}$  but  $17 \notin Q_{21}$ ), so that Bob can use the public key to encrypt his message and send it to Alice.

[1-2] Alice keeps the prime factorization  $(3, 7)$  of 21 as a secret; since  $(3, 7)$  will be used as a private decryption key. (Of course, here we just show an example; in practice, the prime factors  $p$  and  $q$  should be at least 100 digits.)

[2] **Decryption:**

[2-1] Bob converts his plaintext HELP ME to the binary stream  $M = m_1 m_2 \cdots m_{35}$ :

00111 00100 01011 01111 11010 01100 00100

(To save space, we only consider how to encrypt and decrypt  $m_2 = 0$  and  $m_3 = 1$ ; readers are suggested to encrypt and decrypt the whole binary stream).

[2-2] Bob randomly chooses integers  $x_i \in (\mathbb{Z}/21\mathbb{Z})^*$ . Suppose he chooses  $x_2 = 10$  and  $x_3 = 19$  which are elements of  $(\mathbb{Z}/21\mathbb{Z})^*$ .

[2-3] Bob computes the encrypted message  $C = c_1 c_2 \cdots c_k$  from the plaintext  $M = m_1 m_2 \cdots m_k$  using Equation (3.86). To get, for example,  $c_2$  and  $c_3$ , Bob performs:

$$c_2 = x_2^2 \bmod 21 = 10^2 \bmod 21 = 16, \quad \text{since } m_2 = 0,$$

$$c_3 = y \cdot x_3^2 \bmod 21 = 17 \cdot 19^2 \bmod 21 = 5, \quad \text{since } m_3 = 1.$$

(Note that each  $c_i$  is an integer reduced to 21, i.e.,  $m_i$  is a bit, but its corresponding  $c_i$  is not a bit but an integer, which is a string of bits, determined by Table 3.5.)

[2-4] Bob then sends  $c_2$  and  $c_3$  along with all other  $c_i$ 's to Alice.

[3] **Decryption:** To decrypt Bob's message, Alice evaluates the Legendre symbols  $\left(\frac{c_i}{p}\right)$  and  $\left(\frac{c_i}{q}\right)$ . Since Alice knows the prime factorization  $(p, q)$  of  $n$ , it should be easy for her to evaluate these Legendre symbols. For example, for  $c_2$  and  $c_3$ , Alice performs:

**Table 3.5.** The binary equivalents of letters

Letter	Binary Code	Letter	Binary Code	Letter	Binary Code
A	00000	B	00001	C	00010
D	00011	E	00100	F	00101
G	00110	H	00111	I	01000
J	01001	K	01010	L	01011
M	01100	N	01101	O	01110
P	01111	Q	10000	R	10001
S	10010	T	10011	U	10100
V	10101	W	10110	X	10111
Y	11000	Z	11001	□	11010

[3-1] Evaluates the Legendre symbols  $\left(\frac{c_i}{p}\right)$ :

$$\begin{aligned} e'_2 &= \left(\frac{c_2}{p}\right) = \left(\frac{16}{3}\right) = \left(\frac{4^2}{3}\right) = 1, \\ e'_3 &= \left(\frac{c_3}{p}\right) = \left(\frac{5}{3}\right) = \left(\frac{2}{3}\right) = -1. \end{aligned}$$

[3-2] Evaluates the Legendre symbols  $\left(\frac{c_i}{q}\right)$ :

$$\begin{aligned} e''_2 &= \left(\frac{c_2}{q}\right) = \left(\frac{16}{8}\right) = 1, \\ e''_3 &= \left(\frac{c_3}{q}\right) = \left(\frac{5}{7}\right) = -1. \end{aligned}$$

[3-3] Further by Equation (3.88), Alice gets

$$\begin{aligned} m_2 &= 0, & \text{since } e'_2 &= e''_2 = 1, \\ m_3 &= 1, & \text{since } e'_3 &= e''_3 = -1. \end{aligned}$$

**Remark 3.3.8.** The scheme introduced above is a good extension of the public-key idea, but encrypts messages bit by bit. It is completely secure with respect to semantic security as well as bit security<sup>15</sup>. However, a major disadvantage of the scheme is the message expansion by a factor of  $\log n$  bit. To improve the efficiency of the scheme, Blum and Goldwasser [28] proposed another randomized encryption scheme, in which the ciphertext is only longer than the plaintext by a constant number of bits; this scheme is comparable to the RSA scheme, both in terms of speed and message expansion.

**Exercise 3.3.8.** RSA encryption scheme is deterministic and not semantically secure, but it can be made semantically secure by adding randomness to the encryption process (Bellare and Rogaway, [22]). Develop an RSA based probabilistic (randomized) encryption scheme that is semantically secure.

Several other cryptographic schemes, including digital signature schemes and authentication encryption schemes are based on the quadratic residuosity problem (QRP); interested readers are referred to, for example, Chen [47] and Nyang [175] for some recent developments and applications of the quadratic residuosity based cryptosystems.

<sup>15</sup> Bit security is a special case of semantic security. Informally, bit security is concerned with not only that the whole message is not recoverable but also that individual bits of the message are not recoverable. The main drawback of the scheme is that the encrypted message is much longer than its original plaintext.

### 3.3.8 Elliptic Curve Public-Key Cryptosystems

We have discussed some novel applications of elliptic curves in primality testing and integer factorization in Chapter 2. In this subsection, we shall introduce one more novel application of elliptic curves in public-Key cryptography. More specifically, we shall introduce elliptic curve analogues of several well-known public-key cryptosystems, including the Diffie–Hellman key exchange system and the RSA cryptosystem.

**(I) Brief History of Elliptic Curve Cryptography.** Elliptic curves have been extensively studied by number theorists for more than one hundred years, only for their mathematical beauty, not for their applications. However, in the late 1980s and early 1990s many important applications of elliptic curves in both mathematics and computer science were discovered, notably applications of elliptic curves in primality testing (see Kilian [120] and Atkin and Morain [12]) and integer factorization (see Lenstra [140]), both discussed in Chapter 2. Applications of elliptic curves in cryptography were not found until the following two seminal papers were published:

- (1) Victor Miller, “Uses of Elliptic Curves in Cryptography”, 1986. (See [163].)
- (2) Neal Koblitz<sup>16</sup>, “Elliptic Curve Cryptosystems”, 1987. (See [126].)

Since then, elliptic curves have been studied extensively for the purpose of cryptography, and many practically more secure encryption and digital signature schemes have been developed based on elliptic curves. Now elliptic curve cryptography (ECC) is a standard term in the field and there is a textbook by Menezes [155] that is solely devoted to elliptic curve cryptography. There is even a computer company in Canada, called Certicom, which is a leading provider of cryptographic technology based on elliptic curves. In the subsections that follow, we shall discuss the basic ideas and computational methods of elliptic curve cryptography.

---

16



Neal Koblitz received his BSc degree in mathematics from Harvard University in 1969, and his PhD in arithmetic algebraic geometry from Princeton in 1974. From 1979 to the present, he has been at the University of Washington in Seattle, where he is now a professor in mathematics. In recent years his research interests have been centered around the applications of number theory in cryptography. He has published a couple of books in related to number theory and cryptography, two of them are as follows: *A Course in Number Theory and Cryptography* [128], and *Algebraic Aspects of Cryptography* [129]. His other interests include pre-university math education, mathematical development in the Third World, and snorkeling. (Photo by courtesy of Springer-Verlag.)

**(II) Precomputations of Elliptic Curve Cryptography.** To implement elliptic curve cryptography, we need to do the following precomputations:

- [1] Embed Messages on Elliptic Curves: Our aim here is to do cryptography with elliptic curve groups in place of  $\mathbb{F}_q$ . More specifically, we wish to embed plaintext messages as points on an elliptic curve defined over a finite field  $\mathbb{F}_q$ , with  $q = p^r$  and  $p \in \text{Primes}$ . Let our message units  $m$  be integers  $0 \leq m \leq M$ , let also  $\kappa$  be a large enough integer for us to be satisfied with an error probability of  $2^{-\kappa}$  when we attempt to embed a plaintext message  $m$ . In practice,  $30 \leq \kappa \leq 50$ . Now let us take  $\kappa = 30$  and an elliptic curve  $E : y^2 = x^3 + ax + b$  over  $\mathbb{F}_q$ . Given a message number  $m$ , we compute a set of values for  $x$ :

$$x = \{m\kappa + j, j = 0, 1, 2, \dots\} = \{30m, 30m + 1, 30m + 2, \dots\} \quad (3.89)$$

until we find  $x^3 + ax + b$  is a square modulo  $p$ , giving us a point  $(x, \sqrt{x^3 + ax + b})$  on  $E$ . To convert a point  $(x, y)$  on  $E$  back to a message number  $m$ , we just compute  $m = \lfloor x/30 \rfloor$ . Since  $x^3 + ax + b$  is a square for approximately 50% of all  $x$ , there is only about a  $2^{-\kappa}$  probability that this method will fail to produce a point on  $E$  over  $\mathbb{F}_q$ . In what follows, we shall give a simple example of how to embed a message number by a point on an elliptic curve. Let  $E$  be  $y^2 = x^3 + 3x$ ,  $m = 2174$  and  $p = 4177$  (in practice, we select  $p > 30m$ ). Then we calculate  $x = \{30 \cdot 2174 + j, j = 0, 1, 2, \dots\}$  until  $x^3 + 3x$  is a square modulo 4177. We find that when  $j = 15$ :

$$\begin{aligned} x &= 30 \cdot 2174 + 15 \\ &= 65235 \\ x^3 + 3x &= (30 \cdot 2174 + 15)^3 + 3(30 \cdot 2174 + 15) \\ &= 277614407048580 \\ &\equiv 1444 \pmod{4177} \\ &\equiv 38^2 \end{aligned}$$

So we get the message point for  $m = 2174$ :

$$(x, \sqrt{x^3 + ax + b}) = (65235, 38).$$

To convert the message point  $(65235, 38)$  on  $E$  back to its original message number  $m$ , we just compute

$$m = \lfloor 65235/30 \rfloor = \lfloor 2174.5 \rfloor = 2174.$$

- [2] Multiply Points on Elliptic Curves over  $\mathbb{F}_q$ : We have discussed the calculation of  $kP \in E$  over  $\mathbb{Z}/N\mathbb{Z}$ . In elliptic curve public-key cryptography, we are now interested in the calculation of  $kP \in E$  over  $\mathbb{F}_q$ , which can be done in  $\mathcal{O}(\log k(\log q)^3)$  bit operations by the *repeated doubling method*.

If we happen to know  $N$ , the number of points on our elliptic curve  $E$  and if  $k > N$ , then the coordinates of  $kP$  on  $E$  can be computed in  $\mathcal{O}(\log q)^4$  bit operations [128]; recall that the number  $N$  of points on  $E$  satisfies  $N \leq q + 1 + 2\sqrt{q} = \mathcal{O}(q)$  and can be computed by René Schoof's algorithm in  $\mathcal{O}(\log q)^8$  bit operations.

- [3] **Compute Discrete Logarithms on Elliptic Curves:** Let  $E$  be an elliptic curve over  $\mathbb{F}_q$ , and  $B$  a point on  $E$ . Then the *discrete logarithm* on  $E$  is the problem: given a point  $P \in E$ , find an integer  $x \in \mathbb{Z}$  such that  $xB = P$  if such an integer  $x$  exists. It is likely that the discrete logarithm problem on elliptic curves over  $\mathbb{F}_q$  is more intractable than the discrete logarithm problem in  $\mathbb{F}_q$ . It is this feature that makes cryptographic systems based on elliptic curves even more secure than that based on the discrete logarithm problem. In the rest of this subsection, we shall discuss elliptic curve analogues for some of the important public-key cryptosystems.

### (III) Elliptic Curve Analogues of Some Public-Key Cryptosystems.

In what follows, we shall introduce elliptic curve analogues of four widely used public-key cryptosystems, namely the Diffie–Hellman key exchange system, the Massey–Omura, the ElGamal and the RSA public-key cryptosystems.

#### (1) Analogue of the Diffie–Hellman Key Exchange System:

- [1] Alice and Bob publicly choose a finite field  $\mathbb{F}_q$  with  $q = p^r$  and  $p \in \text{Primes}$ , an elliptic curve  $E$  over  $\mathbb{F}_q$ , and a random *base* point  $P \in E$  such that  $P$  generates a large subgroup of  $E$ , preferably of the same size as that of  $E$  itself.
- [2] To agree on a secret key, Alice and Bob choose two secret random integers  $a$  and  $b$ . Alice computes  $aP \in E$  and sends  $aP$  to Bob; Bob computes  $bP \in E$  and sends  $bP$  to Alice. Both  $aP$  and  $bP$  are, of course, public but  $a$  and  $b$  are not.
- [3] Now both Alice and Bob compute the secret key  $abP \in E$ , and use it for further secure communications.

There is no known fast way to compute  $abP$  if one only knows  $P$ ,  $aP$  and  $bP$  – this is the discrete logarithm problem on  $E$ .

#### (2) Analogue of the Massey–Omura Cryptosystem:

- [1] Alice and Bob publicly choose an elliptic curve  $E$  over  $\mathbb{F}_q$  with  $q$  large, and we suppose also that the number of points (denoted by  $N$ ) is publicly known.
- [2] Alice chooses a secret pair of numbers  $(e_A, d_A)$  such that  $d_A e_A \equiv 1 \pmod{N}$ . Similarly, Bob chooses  $(e_B, d_B)$ .

- [3] If Alice wants to send a secret message-point  $P \in E$  to Bob, the procedure is as follows:
- [3-1] Alice sends  $e_AP$  to Bob,
  - [3-2] Bob sends  $e_Be_AP$  to Alice,
  - [3-3] Alice sends  $d_Ae_Be_AP = e_BP$  to Bob,
  - [3-4] Bob computes  $d_Be_BP = P$ .

Note that an eavesdropper would know  $e_AP$ ,  $e_Be_AP$ , and  $e_BP$ . So if he could solve the discrete logarithm problem on  $E$ , he could determine  $e_B$  from the first two points and then compute  $d_B = e_B^{-1} \bmod N$  and hence get  $P = d_B(e_BP)$ .

### (3) Analogue of the ElGamal Cryptosystem:

- [1] Alice and Bob publicly choose an elliptic curve  $E$  over  $\mathbb{F}_q$  with  $q = p^r$  and  $p \in \text{Primes}$ , and a random *base* point  $P \in E$ .
- [2] Alice chooses a random integer  $r_a$  and computes  $r_aP$ ; Bob also chooses a random integer  $r_b$  and computes  $r_bP$ .
- [3] To send a message-point  $M$  to Bob, Alice chooses a random integer  $k$  and sends the pair of points  $(kP, M + k(r_bP))$ .
- [4] To read  $M$ , Bob computes

$$M + k(r_bP) - r_b(kP) = M. \quad (3.90)$$

An eavesdropper who can solve the discrete logarithm problem on  $E$  can, of course, determine  $r_b$  from the publicly known information  $P$  and  $r_bP$ . But as everybody knows, there is no efficient way to compute discrete logarithms, so the system is secure.

### (4) Analogue of the RSA Cryptosystem:

RSA, the most popular cryptosystem in use, also has the following elliptic curve analogue:

- [1]  $N = pq$  is a public key which is the product of the two large secret primes  $p$  and  $q$ .
- [2] Choose two random integers  $a$  and  $b$  such that  $E : y^2 = x^3 + ax + b$  defines an elliptic curve both mod  $p$  and mod  $q$ .
- [3] To encrypt a message-point  $P$ , just perform  $eP \bmod N$ , where  $e$  is the public (encryption) key. To decrypt, one needs to know the number of points on  $E$  modulo both  $p$  and  $q$ .

The above are some elliptic curve analogues of certain public-key cryptosystems. It should be noted that almost every public-key cryptosystem has an elliptic curve analogue; it is of course possible to develop new elliptic curve cryptosystems which do not rely on the existing cryptosystems.



**Exercise 3.3.9.** Work back from the descriptions of the elliptic curve analogues of the ElGamal and the Massey–Omura cryptosystems discussed above, to give complete algorithmic descriptions of the original ElGamal and the original Massey–Omura public-key cryptosystems.

**(IV) Menezes-Vanstone Elliptic Curve Cryptosystem.** A serious problem with the above mentioned elliptic curve cryptosystems is that the plaintext message units  $m$  lie on the elliptic curve  $E$ , and there is no convenient method known of deterministically generating such points on  $E$ . Fortunately, Menezes<sup>17</sup> and Vanstone<sup>18</sup> had discovered a more efficient variation [156]; in this variation which we shall describe below, the elliptic curve is used for “masking”, and the plaintext and ciphertext pairs are allowed to be in  $\mathbb{F}_p^* \times \mathbb{F}_p^*$  rather than on the elliptic curve.

[1] Preparation: Alice and Bob publicly choose an elliptic curve  $E$  over  $\mathbb{F}_p$  with  $p > 3$  is prime and a random *base* point  $P \in E(\mathbb{F}_p)$  such that  $P$  generates a large subgroup  $H$  of  $E(\mathbb{F}_p)$ , preferably of the same size as that of  $E(\mathbb{F}_p)$  itself. Assume that randomly chosen  $k \in \mathbb{Z}_{|H|}$  and  $a \in \mathbb{N}$  are secret.

[2] Encryption: Suppose now Alice wants to sent message

$$m = (m_1, m_2) \in (\mathbb{Z}/p\mathbb{Z})^* \times (\mathbb{Z}/p\mathbb{Z})^* \quad (3.91)$$

to Bob, then she does the following:

[2-1]  $\beta = aP$ , where  $P$  and  $\beta$  are public.

17



Alfred J. Menezes is a professor of mathematics in the Department of Combinatorics and Optimization at the University of Waterloo, where he teaches courses in cryptography, coding theory, finite fields, and discrete mathematics. He is actively involved in cryptographic research, and consults on a regular basis for Certicom Corp., He completed the Bachelor of Mathematics and M.Math degrees in 1987 and 1989 respectively, and a Ph.D. in Mathematics from the University of Waterloo (Canada) in 1992.

18



Scott A. Vanstone is one of the founders of Certicom, the first company to develop elliptic curve cryptography commercially. He devotes much of his research to the efficient implementation of the elliptic curve cryptography for the provision of information security services in hand-held computers, smart cards, wireless devices, and integrated circuits. Vanstone has published more than 150 research papers and several books on topics such as cryptography, coding theory, finite fields, finite geometry, and combinatorial designs. Recently, he was elected a Fellow of the Royal Society of Canada. Vanstone received a Ph.D. in mathematics from the University of Waterloo in 1974.

$$[2-2] \quad (y_1, y_2) = k\beta$$

$$[2-3] \quad c_0 = kP.$$

$$[2-4] \quad c_j \equiv y_j m_j \pmod{p} \text{ for } j = 1, 2.$$

[2-5] Alice sends the encrypted message  $c$  of  $m$  to Bob:

$$c = (c_0, c_1, c_2). \quad (3.92)$$

[3] Decryption: Upon receiving Alice's encrypted message  $c$ , Bob calculates the following to recover  $m$ :

$$[3-1] \quad ac_0 = (y_1, y_2).$$

$$[3-1] \quad m = (c_1 y_1^{-1} \pmod{p}, c_2 y_2^{-1} \pmod{p}).$$

**Example 3.3.11.** The following is a nice example of Menezes-Vanstone cryptosystem, taken from [165].

[1] Key generation: Let  $E$  be the elliptic curve given by  $y^2 = x^3 + 4x + 4$  over  $\mathbb{F}_{13}$ , and  $P = (1, 3)$  be a point on  $E$ . Choose  $E(\mathbb{F}_{13}) = H$  which is cyclic of order 15, generated by  $P$ . Let also the private keys  $k = 5$  and  $a = 2$ , and the plaintext  $m = (12, 7) = (m_1, m_2)$ .

[2] Encryption: Alice computes:

$$\beta = aP = 2(1, 3) = (12, 8)$$

$$(y_1, y_2) = k\beta = 5(12, 8) = (10, 11)$$

$$c_0 = kP = 5(1, 3) = (10, 2)$$

$$c_1 \equiv y_1 m_1 \equiv 10 \cdot 2 \equiv 3 \pmod{13}$$

$$c_2 \equiv y_2 m_2 \equiv 11 \cdot 7 \equiv 12 \pmod{13}.$$

Then Alice sends

$$c = (c_0, c_1, c_2) = ((10, 2), 3, 12)$$

to Bob.

[3] Decryption: Upon receiving Alice's message, Bob computes:

$$ac_0 = 2(10, 2) = (10, 11) = (y_1, y_2)$$

$$m_1 \equiv c_1 y_1^{-1} \equiv 12 \pmod{13}$$

$$m_2 \equiv c_2 y_2^{-1} \equiv 7 \pmod{13}.$$

Thus, Bob recovers the message  $m = (12, 7)$ .

We have introduced so far the most popular public-key cryptosystems, such as Diffie-Hellman-Merkle, RSA, Elliptic curve and probabilistic cryptosystems. There are, of course, many other types of public-key cryptosystems in use, such as Rabin, McEliece and Knapsack cryptosystems. Readers who are interested in the cryptosystems which are not covered in this book are suggested to consult Menezes et al. [157].

### 3.3.9 Digital Signatures

The idea of public-key cryptography (suppose we are using the RSA public-key scheme) can also be used to obtain digital signatures. Recall that in public-key cryptography, we perform

$$C = E_{e_k}(M), \quad (3.93)$$

where  $M$  is the message to be encrypted, for message encryption, and

$$M = D_{d_k}(C), \quad (3.94)$$

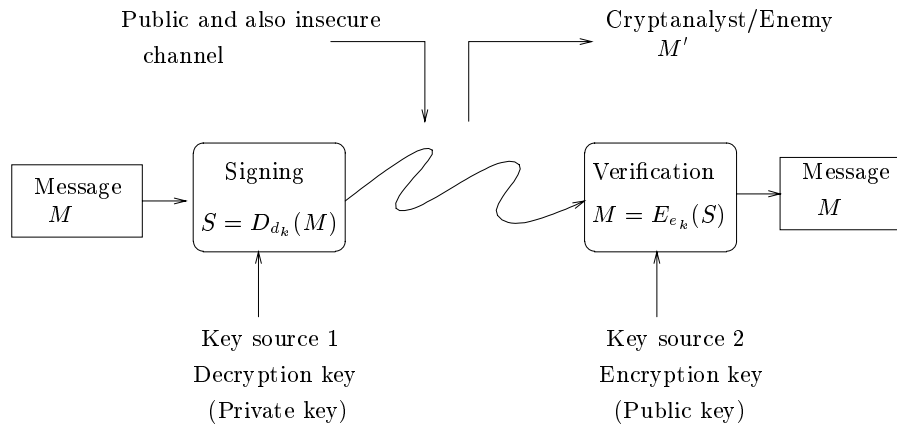
where  $C$  is the encrypted message needed to be decrypted, for decryption. In digital signatures, we perform the operations in exactly the opposite direction. That is, we perform (see also Figure 3.11)

$$S = D_{d_k}(M), \quad (3.95)$$

where  $M$  is the message to be signed, for signature generation, and

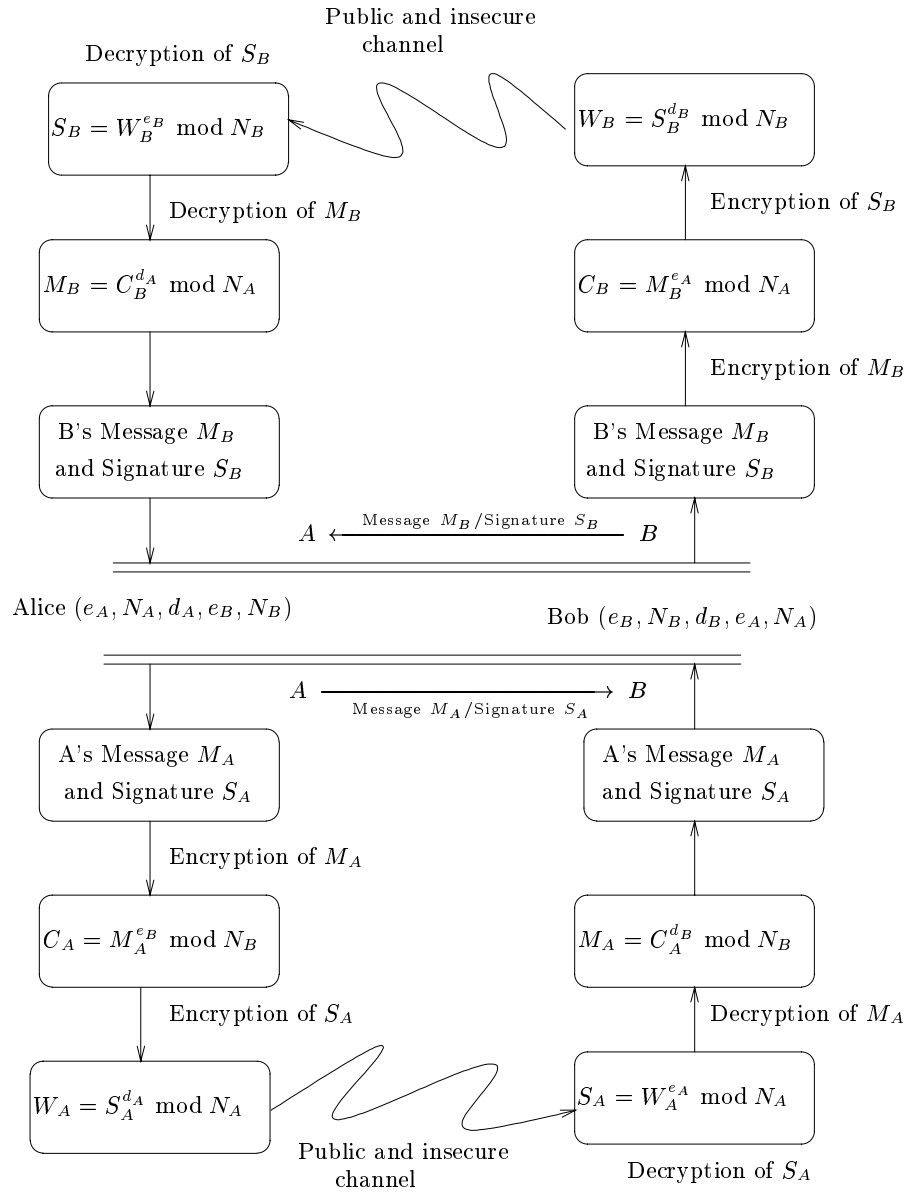
$$M = E_{e_k}(S), \quad (3.96)$$

where  $S$  is the signed message needed to be verified, for signature verification. Suppose now Alice wishes to send Bob a secure message as well as a digital



**Figure 3.11.** Digital signatures

signature. Alice first uses Bob's public key to encrypt her message, and then, she uses her private key to encrypt her signature, and finally sends out her



**Figure 3.12.** Sending encrypted messages and signatures using the RSA scheme (the encrypted message and the signature are two different texts)

message and signature to Bob. At the other end, Bob first uses Alice's public key to decrypt Alice's signature, and then uses his private key to decrypt Alice's message. Figure 3.12 shows how A (Alice) and B (Bob) can send secure message/signature to each other over the insecure channel.

**Example 3.3.12 (Digital Signature).** To verify that the \$100 offer in Example 3.3.8 actually came from RSA, the following signature was added:

$S = 167178611503808442460152713891683982454369010323583112178\_$   
 $350384469290626554487922371144905095786086556624965779748\_$   
 $40004057020373.$

It was encrypted by  $S = M^d \pmod{N}$ , where  $d$  is the secret key, as in Example 3.3.8. To decrypt the signature, we use  $M = S^e \pmod{N}$  by performing the following procedure (also the same as in Example 3.3.8):

```

 $C \leftarrow 1$ 
 $e = (10001100101111)_2$ 
for  $i$  from 1 to 14 do
     $C \leftarrow C^2 \pmod{N}$ 
    if  $e[i] = 0$  then  $C \leftarrow C * M \pmod{N}$ 
print  $C$ 

```

which gives the following decrypted text:

6091819200019151222051800230914190015140500082114\_  
 041805040004151212011819.

It translates to

FIRST SOLVER WINS ONE HUNDRED DOLLARS

Since this signature was encrypted by RSA's secret key, it cannot be forged by an eavesdropper or even by RSA people themselves.

In Example 3.3.12, the signature is a different text from the message, and usually is appended to the encrypted message. We can, of course directly sign the signature on the message. This can be done in the following way. Suppose A (Alice) wants to send B (Bob) a signed message. Suppose also that

- [1] Alice (A) has her own public and secret keys  $(e_A, N_A; d_A)$  as well as B's public key  $e_B$  and  $N_B$  from a public domain;
- [2] Bob (B) has his own public and secret keys  $(e_B, N_B; d_B)$  as well as A's public key  $e_A$  and  $N_A$  from a public domain.

To send a signed message from A to B:

- [1] Alice uses B's public key  $e_B$  and  $N_B$  to encrypt her message  $M_A$ :

$$C_A = M_A^{e_B} \pmod{N_B}. \quad (3.97)$$

- [2] Alice signs the message using her own secret key  $d_A$  directly on the encrypted message:

$$S_A = C_A^{d_A} \bmod N_A, \quad (3.98)$$

and sends this signed message to B over the network.

Upon receiving A's signed message,

- [1] B uses A's public key  $e_A$  to decrypt A's signature:

$$C_A = S_A^{e_A} \bmod N_A. \quad (3.99)$$

- [2] B further uses his own secret key  $d_B$  to decrypt A's encrypted message:

$$M_A = C_A^{d_B} \bmod N_B. \quad (3.100)$$

In this way, Bob can make sure that the message he has just received indeed comes from A, since the signature of A's message is encrypted by A's own secret key, which is only known to A. Once the message is sent out, A cannot deny the message. Similarly, Bob can send a signed message to Alice. The above process is shown in Figure 3.13.

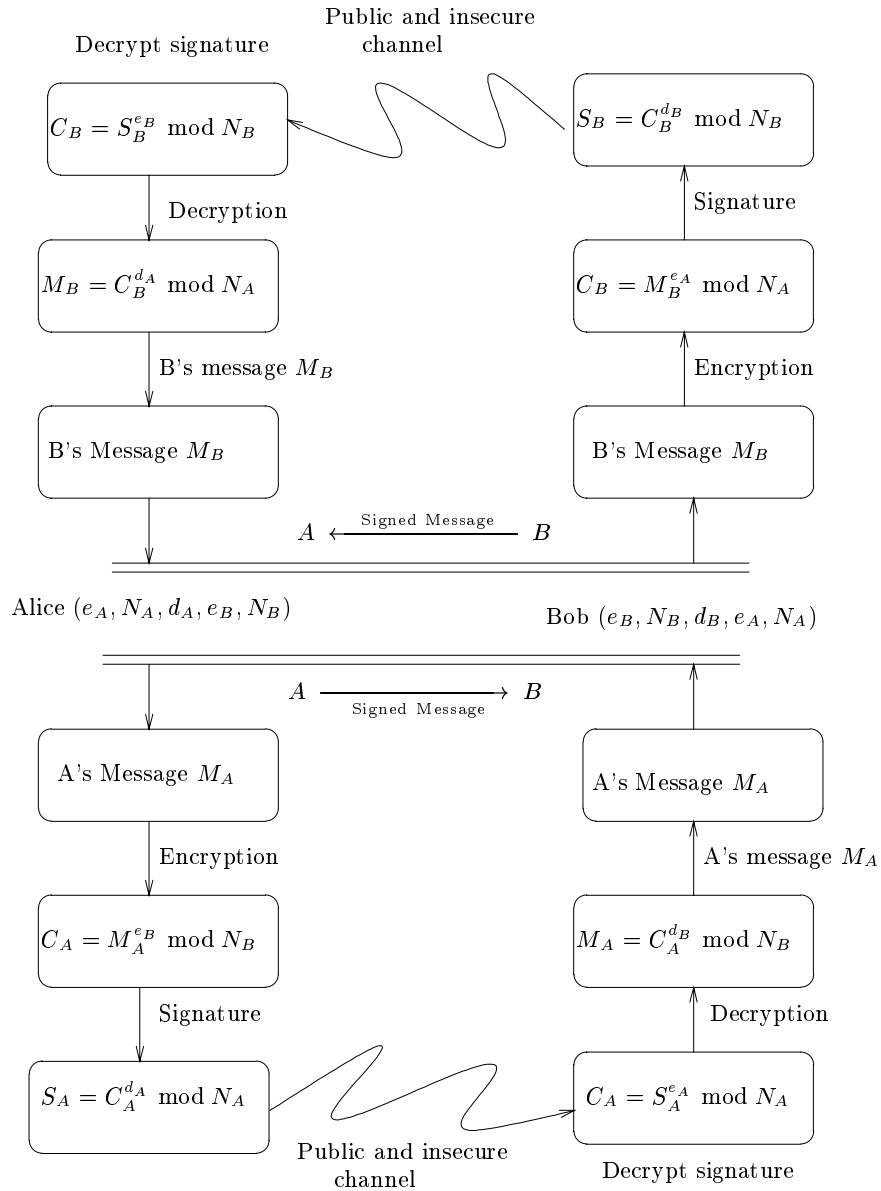
**Example 3.3.13 (Digital Signature).** Suppose now Alice wants to send Bob the signed message "Number Theory is the Queen of Mathematics". The process can be as follows:

- [1] Suppose Alice has the following information at hand:

$$\begin{aligned} M_A &= 1421130205180020080515182500091900200805001721050514001\_ \\ &\quad 506001301200805130120090319 \\ N_A &= 1807082088687404805951656164405905566278102516769401349\_ \\ &\quad 1701270214500566625402440483873411275908123033717818879\_ \\ &\quad 66563182013214880557 \text{ (130 digits)} \\ &= 3968599945959745429016112616288378606757644911281006483\_ \\ &\quad 2555157243 \cdot 45534498646735972188403686897274408864356301\_ \\ &\quad 263205069600999044599 \\ e_A &= 2617 \\ d_A &= 9646517683975179648125577614348681987353875490740747744\_ \\ &\quad 7102309852757971788848801635711139144032242624779107574\_ \\ &\quad 0923050236448593109 \end{aligned}$$

and suppose Bob has the following information at hand:

$$\begin{aligned} N_B &= 1143816257578888676692357799761466120102182967212423625\_ \\ &\quad 6256184293570693524573389783059712356395870505898907514\_ \\ &\quad 7599290026879543541 \text{ (129 digits)} \\ &= 3490529510847650949147849619903898133417764638493387843\_ \\ &\quad 990820577 \cdot 327691329932667095499619881908344614131776429\_ \\ &\quad 67992942539798288533 \end{aligned}$$



**Figure 3.13.** Sending encrypted and signed messages using the RSA scheme (the signatures are directly made on the encrypted message)

$e_B = 9007$   
 $d_B = 1066986143685780244428687713289201547807099066339378628\_$   
 $0122622449663106312591177447087334016859746230655396854\_$   
 $4513277109053606095$

[2] Alice first encrypts the message  $M_A$  using  $e_B$  and  $N_B$  to get

$$C_A = M_A^{e_B} \bmod N_B$$

by the following process:

```

 $C_A \leftarrow 1$ 
 $e_B \leftarrow (10001100101111)_2$ 
for  $i$  from 1 to 14 do
     $C_A \leftarrow C_A^2 \bmod N_B$ 
    if  $e_B[i] = 1$  then  $C_A \leftarrow C_A \cdot M_A \bmod N_B$ 
Save  $C_A$ 

```

[3] Alice then signs the message  $C_A$  using  $d_A$  and  $N_A$  to get  $S_A = C_A^{d_A} \bmod N_A$  via the following process:

```

 $S_A \leftarrow 1$ 
 $d_A \leftarrow (10110010 \dots 11010101)_2$ 
for  $i$  from 1 to 429 do
     $S_A \leftarrow S_A^2 \bmod N_A$ 
    if  $d_A[1, i] = 1$  then  $S_A \leftarrow S_A \cdot C_A \bmod N_A$ 
Send  $S_A$ 

```

[4] Upon receiving Alice's message, Bob first decrypts Alice's signature using  $e_A$  and  $N_A$  to get  $C_A = S_A^{e_A} \bmod N_A$  via the following process:

```

 $C_A \leftarrow 1$ 
 $e_A \leftarrow (101000111001)_2$ 
for  $i$  from 1 to 12 do
     $C_A \leftarrow C_A^2 \bmod N_A$ 
    if  $e_A[i] = 1$  then  $C_A \leftarrow C_A \cdot S_A \bmod N_A$ 
Save  $C_A$ 

```

[5] Bob then decrypts Alice's message using  $d_B$  and  $N_B$  to get  $M_A = C_A^{d_B} \bmod N_B$  via the following process:

```

 $M_A \leftarrow 1$ 
 $d_B := (1001110110 \dots 1001111)_2$ 
for  $i$  from 1 to 426 do
     $M_A \leftarrow M_A^2 \bmod N_B$ 
    if  $d_B[i] = 0$  then  $M_A \leftarrow M_A \cdot C_A \bmod N_B$ 
print  $M_A$ 

```



**Remark 3.3.9.** Suppose Bob is sending an encrypted message to Alice. Normally, the encrypted message consists of a number of blocks; one of the blocks is Bob's signature. Alice can easily identify which block is the *signature block*, since the ordinary decryption procedure for that block yields gibberish. In practice, there are two ways for constructing Bob's encrypted signature (Denson [60]), depending on the values of the moduli  $n_B$  and  $n_A$ :

(1) If  $n_A < n_B$ , then

$$S_B = (M^{d_B} \bmod n_B)^{e_A} \bmod n_A, \quad M_B = (S^{d_A} \bmod n_A)^{e_B} \bmod n_B.$$

The inequality  $n_A < n_B$  ensures that the expression in the parentheses is not too large to be encrypted by Alice's encryption key.

(2) If  $n_A > n_B$ , then

$$S_B = (M^{e_A} \bmod n_A)^{d_B} \bmod n_B, \quad M_B = (S^{e_B} \bmod n_B)^{d_A} \bmod n_A.$$

The inequality  $n_A > n_B$  ensures that the expression in the parentheses is not too large to be encrypted by Bob's decryption key.

The above mentioned signature scheme is based on RSA cryptosystem. Of course, a signature scheme can be based on other cryptosystem. In what follows, we shall introduce a very influential signature scheme based of ElGamal's cryptosystem [69]; the security of such a signature scheme depends on the intractability of discrete logarithms over a finite field.

**Algorithm 3.3.5 (ElGamal Signature Scheme).** This algorithm tries to generate digital signature  $S = (a, b)$  for message  $m$ . Suppose that Alice wishes to send a signed message to Bob.

[1] [ElGamal key generation] Alice does the following:

[1-1] Choose a prime  $p$  and two random integers  $g$  and  $x$ , such that both  $g$  and  $x$  are less than  $p$ .

[1-2] Compute  $y \equiv g^x \pmod{p}$ .

[1-3] Make  $(y, g, p)$  public (both  $g$  and  $p$  can be shared among a group of users), but keep  $x$  as a secret.

[2] [ElGamal signature generation] Alice does the following:

[2-1] Choose at random an integers  $k$  such that  $\gcd(k, p-1) = 1$ .

[2-2] Compute

$$\left. \begin{aligned} a &\equiv g^x \pmod{p}, \\ b &\equiv k^{-1}(m - xa) \pmod{(p-1)}. \end{aligned} \right\} \quad (3.101)$$

Now Alice has generated the signature  $(a, b)$ . She must keep the random integer,  $k$ , as secret.

- [3] [ElGamal signature verification] To verify Alice's signature, Bob confirms that

$$y^a a^b \equiv g^m \pmod{p}. \quad (3.102)$$

### 3.3.10 Digital Signature Standard (DSS)

In August 1991, the U.S. government's National Institute of Standards and Technology (NIST) proposed an algorithm for digital Signatures. The algorithm is known as DSA, for Digital Signature Algorithm. The DSA has become the U.S. Federal Information Processing Standard 186 (FIPS 186). It is called the Digital Signature Standard (DSS), and is the first digital signature scheme recognized by any government. The role of DSA/DSS is expected to be analogous to that of the Data Encryption Standard (DES). The DSA/DSS is similar to a signature scheme proposed by Schnorr [220]; it is also similar to a signature scheme of ElGamal [69]. The DSA is intended for use in electronic mail, electronic funds transfer, electronic data interchange, software distribution, data storage, and other applications which require data integrity assurance and data authentication. The DSA/DSS consists of two main processes:

- (1) Signature generation (using the private key),
- (2) Signature verification (using the public key).

A one-way hash function is used in the signature generation process to obtain a condensed version of data, called a message digest. The message digest is then signed. The digital signature is sent to the intended receiver along with the signed data (often called the message). The receiver of the message and the signature verifies the signature by using the sender's public key. The same hash function must also be used in the verification process. In what follows, we shall give the formal specifications of the DSA/DSS.

**Algorithm 3.3.6 (Digital Signature Algorithm, DSA).** This is a variation of ElGamal signature scheme. It generates a signature  $S = (r, s)$  for the message  $m$ .

- [1] [DSA key generation] To generate the DSA key, the sender performs the following:
- [1-1] Find a 512-bit prime  $p$  (which will be public).
  - [1-2] Find a 160-bit prime  $q$  dividing evenly into  $p - 1$  (which will be public).

[1-3] Generate an element  $g \in \mathbb{Z}/p\mathbb{Z}$  whose multiplicative order is  $q$ , i.e.,  $g^q \equiv 1 \pmod{p}$ .

[1-4] Find a one-way function  $H$  mapping messages into 160-bit values.

[1-5] Choose a secret key  $x$ , with  $0 < x < q$ .

[1-6] Choose a public key  $y$ , where  $y \equiv g^x \pmod{p}$ .

Clearly, the secret  $x$  is the discrete logarithm of  $y$ , modulo  $p$ , to the base  $g$ .

[2] [DSA Signature Generation] To sign the message  $m$ , the sender produces his signature as  $(r, s)$ , by selecting a random integer  $k \in \mathbb{Z}/q\mathbb{Z}$  and computing

$$\left. \begin{aligned} r &\equiv (g^k \pmod{p}) \pmod{q}, \\ s &\equiv k^{-1}(H(m) + xr) \pmod{q}. \end{aligned} \right\} \quad (3.103)$$

[3] [DSA Signature Verification] To verify the signature  $(r, s)$  for the message  $m$  from the sender, the receiver first computes:

$$t \equiv s^{-1} \pmod{q}, \quad (3.104)$$

and then accepts the signature as valid if the following congruence holds:

$$r \equiv \left( g^{H(m)t} y^{rt} \pmod{p} \right) \pmod{q}. \quad (3.105)$$

If the congruence (3.105) does not hold, then the message either may have been incorrectly signed, or may have been signed by an impostor. In this case, the message is considered to be invalid.

There are, however, many responses solicited by the (US) Association of Computing Machinery (ACM) [45], positive and negative, to the NIST's DSA. Some positive aspects of the DSA include:

- (1) The U.S. government has finally recognized the utility and the usefulness of public-key cryptography. In fact, the DSA is the only signature algorithm that has been publicly proposed by any government.
- (2) The DSA is based on reasonable familiar number-theoretic concepts, and it is especially useful to the financial services industry.
- (3) Signatures in DSA are relatively short (only 320 bits), and the key generation process can be performed very efficiently.
- (4) When signing, the computation of  $r$  can be done even before the message  $m$  is available, in a "precomputation" step.

Whilst some negative aspects of the DSA include:

- (1) The DSA does not include key exchanges, and cannot be used for key distribution and encryption.

- (2) The key size in DSA is too short; it is restricted to a 512-bit modulus or key size, which is too short and should be increased to at least 1024 bits.
- (3) The DSA is not compatible with existing international standards; for example, the international standards organizations such as ISO, CCITT and SWIFT all have accepted the RSA as a standard.

Nevertheless, the DSA is the only one publicly known government digital signature standard.

We have already noted that almost every public-key cryptosystem has an elliptic curve analogue. It should also be noted that digital signature schemes can also be represented by elliptic curves over  $\mathbb{F}_q$  with  $q$  a prime power or over  $\mathbb{Z}/n\mathbb{Z}$  with  $n = pq$  and  $p, q \in \text{Primes}$ . In exactly the same way as that for public-key cryptography, several elliptic curve analogues of digital signature schemes have already been proposed (see, for example, Meyer and Müller [160]). In what follows we shall describe an elliptic curve analogue of the DSA/DSS, called ECDSA.

**Algorithm 3.3.7 (Elliptic Curve Digital Signature Algorithm).** Let  $E$  be an elliptic curve over  $\mathbb{F}_p$  with  $p$  prime, and let  $P$  be a point of prime order  $q$  (note that the  $q$  here is just a prime number, not a prime power) in  $E(\mathbb{F}_p)$ . Suppose Alice wishes to send a signed message to Bob.

[1] [ECDSA key generation] Alice does the following:

[1-1] select a random integer  $x \in [1, q - 1]$ ,

[1-2] compute  $Q = xP$ ,

[1-3] make  $Q$  public, but keep  $x$  as a secret.

Now Alice has generated the public key  $Q$  and the private key  $x$ .

[2] [ECDSA signature generation] To sign a message  $m$ , Alice does the following:

[2-1] select a random integer  $k \in [1, q - 1]$ ,

[2-2] compute  $kP = (x_1, y_1)$ , and  $r \equiv x_1 \pmod{q}$ . If  $r = 0$ , go to step [2-1].

[2-3] compute  $k^{-1} \pmod{q}$ .

[2-4] compute  $s \equiv k^{-1}(H(m) + xr) \pmod{q}$ , where  $H(m)$  is the hash value of the message. If  $s = 0$ , go to step [2-1].

The signature for the message  $m$  is the pair of integers  $(r, s)$ .

[3] [ECDSA signature verification] To verify Alice's signature  $(r, s)$  of the message  $m$ , Bob should do the following:

[3-1] obtain an authenticated copy of Alice's public key  $Q$ ;

- [3-2] verify that  $(r, s)$  are integers in the interval  $[1, q - 1]$ , computes  $kP = (x_1, y_1)$ , and  $r \equiv x_1 \pmod{q}$ .
- [3-3] compute  $w \equiv s^{-1} \pmod{q}$  and  $H(m)$ .
- [3-4] compute  $u_1 \equiv H(m)w \pmod{q}$  and  $u_2 \equiv rw \pmod{q}$ .
- [3-5] compute  $u_1P + u_2Q = (x_0, y_0)$  and  $v \equiv x_0 \pmod{q}$ .
- [3-6] accept the signature if and only if  $v = r$ .

**Exercise 3.3.10.** Try to develop an elliptic curve analogue of an existing signature scheme that you are familiar with for obtaining and checking digital signatures.

### 3.3.11 Database Security

Databases pose a special challenge to the designer of secure information systems. Databases are meant to be shared. The sharing is often complex. In many organizations, there are many “rules” concerning the access to different *fields* (or parts) of a database. For example, the payroll department may have access to the name, address and salary fields, while the insurance office may have access to the health field of an individual. In this subsection, we shall introduce a method for database protection; it encrypts the entire database but the individual fields may be decrypted and read without affecting the security of other fields in the database.

Let

$$D = \langle F_1, F_2, \dots, F_n \rangle \quad (3.106)$$

where  $D$  is the database and each  $F_i$  is an individual file (or record). As in RSA encryption, each file in  $D$  can be regarded as an integer. To encrypt  $D$ , we first select  $n$  distinct primes  $m_1, m_2, \dots, m_n$ , where  $m_i > F_i$ , for  $i = 1, 2, \dots, n$ . Then by solving the following system of congruences:

$$\left. \begin{array}{l} C \equiv F_1 \pmod{m_1}, \\ C \equiv F_2 \pmod{m_2}, \\ \dots\dots\dots \\ C \equiv F_n \pmod{m_n}, \end{array} \right\} \quad (3.107)$$

we get  $C$ , the encrypted text of  $D$ . According to the Chinese Remainder Theorem, such a  $C$  always exists and can be found. Let

$$\left. \begin{array}{l} M = m_1 m_2 \dots m_n, \\ M_i = M / m_i, \\ e_i = M_i [M_i^{-1} \pmod{m_i}], \end{array} \right\} \quad (3.108)$$

for  $i = 1, 2, \dots, n$ . Then  $C$  can be obtained as follows:

$$C = \sum_{j=1}^n e_j F_j \pmod{M}, \quad 0 \leq C < M. \quad (3.109)$$

The integers  $e_1, e_2, \dots, e_n$  are used as the *write-keys*. To retrieve the  $i$ -th file  $F_i$  from the encrypted text  $C$  of  $D$ , we simply perform the following operation:

$$F_i \equiv C \pmod{m_i}, \quad 0 \leq F_i < m_i. \quad (3.110)$$

The moduli  $m_1, m_2, \dots, m_n$  are called the *read-keys*. Only people knowing the read-key  $m_i$  can read file  $F_i$ , but not other files. To read other files, for example,  $F_{i+2}$ , it is necessary to know a read-key other than  $m_i$ . We present in the following an algorithm for database encryption and decryption.

**Algorithm 3.3.8 (Database protection).** Given  $D = \langle F_1, F_2, \dots, F_n \rangle$ , this algorithm will first encrypt the database  $D$  into its encrypted text  $C$ . To retrieve information from the encrypted database  $C$ , the user uses the appropriate read-key  $m_i$  to read file  $F_i$ :

Part I: Database Encryption. The database administrators (DBA) perform the following operations to encrypt the database  $D$ :

- [1] Select  $n$  distinct primes  $m_1, m_2, \dots, m_n$  with  $m_i > F_i$ , for  $i = 1, 2, \dots, n$ .
- [2] Use the Chinese Remainder Theorem to solve the following system of congruences:

$$\left. \begin{array}{l} C \equiv F_1 \pmod{m_1}, \\ C \equiv F_2 \pmod{m_2}, \\ \dots\dots\dots \\ C \equiv F_n \pmod{m_n}, \end{array} \right\} \quad (3.111)$$

and get

$$C = \sum_{j=1}^n e_j F_j \pmod{M}, \quad 0 \leq C < M \quad (3.112)$$

where

$$\begin{aligned} M &= m_1 m_2 \dots m_n, \\ M_i &= M / m_i, \\ e_i &= M_i [M_i^{-1} \pmod{m_i}], \end{aligned}$$

for  $i = 1, 2, \dots, n$ .

- [3] Distribute the read-key  $m_i$  to the appropriate database user  $U_i$ .

Part II: Database Decryption. At this stage, the database user  $U_i$  is supposed to have access to the encrypted database  $C$  as well as to have the read-key  $m_i$ , so he performs the following operation:

$$F_i \equiv C \pmod{m_i}, \quad 0 \leq F_i < m_i. \quad (3.113)$$

The required file  $F_i$  should be now readable by user  $U_i$ .

**Example 3.3.14 (Database Encryption and Decryption).** Let

$$\begin{aligned} D &= \langle F_1, F_2, F_3, F_4, F_5 \rangle \\ &= \langle 198753, 217926, 357918, 377761, 391028 \rangle. \end{aligned}$$

Choose five primes  $m_1, m_2, m_3, m_4$  and  $m_5$  as follows:

$$\begin{aligned} m_1 &= 350377 > F_1 = 198753, \\ m_2 &= 364423 > F_2 = 217926, \\ m_3 &= 376127 > F_3 = 357918, \\ m_4 &= 389219 > F_4 = 377761, \\ m_5 &= 391939 > F_5 = 391028. \end{aligned}$$

According to (3.111), we have:

$$\begin{aligned} C &\equiv F_1 \pmod{m_1} \implies C \equiv 198753 \pmod{350377} \\ C &\equiv F_2 \pmod{m_2} \implies C \equiv 217926 \pmod{364423} \\ C &\equiv F_3 \pmod{m_3} \implies C \equiv 357918 \pmod{376127} \\ C &\equiv F_4 \pmod{m_4} \implies C \equiv 377761 \pmod{389219} \\ C &\equiv F_5 \pmod{m_5} \implies C \equiv 391028 \pmod{391939}. \end{aligned}$$

Using the Chinese Remainder Theorem to solve the above system of congruences, we get

$$C = 5826262707691801601352277219.$$

Since  $0 \leq C < M$  with

$$\begin{aligned} M &= 350377 \cdot 364423 \cdot 376127 \cdot 389219 \cdot 391939 \\ &= 7326362302832726883024522697, \end{aligned}$$

$C$  is the required encrypted text of  $D$ . Now suppose user  $U_2$  has the read-key  $m_2 = 364423$ . Then he can simply perform the following computation and get  $F_2$ :

$$F_2 \equiv C \pmod{m_i}.$$

Now

$$\begin{aligned} C \pmod{m_2} &= 5826262707691801601352277219 \bmod 364423 \\ &= 217926 \\ &= F_2, \end{aligned}$$

which is exactly what the user  $U_2$  wanted. Similarly, a user can read  $F_5$  if he knows  $m_5$ , since

$$\begin{aligned} C \pmod{m_5} &= 5826262707691801601352277219 \pmod{391939} \\ &= 391028 \\ &= F_5. \end{aligned}$$

**Remark 3.3.10.** In Example 3.3.14, we have not explicitly given the computing processes for the write keys  $e_i$  and the encrypted text  $C$ ; we give now the detailed computing processes as follows:

$$\begin{aligned} e_1 &= M_1 \cdot (M_1^{-1} \pmod{m_1}) \\ &= 20909940729079611056161 \cdot (20909940729079611056161^{-1} \pmod{350377}) \\ &= 3040577211237653482509539493 \\ e_2 &= M_2 \cdot (M_2^{-1} \pmod{m_2}) \\ &= 20104006341072673467439 \cdot (20104006341072673467439^{-1} \pmod{364423}) \\ &= 2830382740740598479460334493 \quad e_3 = M_3 \cdot (M_3^{-1} \pmod{m_3}) \\ &= 19478426975018349873911 \cdot (19478426975018349873911^{-1} \pmod{376127}) \\ &= 1991883420892351476456012771 \\ e_4 &= M_4 \cdot (M_4^{-1} \pmod{m_4}) \\ &= 18823239109171769320163 \cdot (18823239109171769320163^{-1} \pmod{389219}) \\ &= 6068028768384594103971626147 \\ e_5 &= M_5 \cdot (M_5^{-1} \pmod{m_5}) \\ &= 18692608550903908217923 \cdot (18692608550903908217923^{-1} \pmod{391939}) \\ &= 721852464410256223651532491. \end{aligned}$$

So

$$\begin{aligned} C &= (e_1 F_1 + e_2 F_2 + e_3 F_3 + e_4 F_4 + e_5 F_5) \pmod{M} \\ &= (3040577211237653482509539493 \cdot 198753 \\ &\quad + 2830382740740598479460334493 \cdot 217926 \\ &\quad + 1991883420892351476456012771 \cdot 357918 \\ &\quad + 6068028768384594103971626147 \cdot 377761 \\ &\quad + 721852464410256223651532491 \cdot 391028) \\ &\quad \pmod{7326362302832726883024522697} \\ &= 5826262707691801601352277219. \end{aligned}$$

**Exercise 3.3.11.** Let the database  $D$  be

$$\begin{aligned} D &= \langle F_1, F_2, F_3, F_4 \rangle \\ &= \langle 9853, 6792, 3761, 5102 \rangle. \end{aligned}$$

and the four read keys be



$$\begin{aligned}
m_1 &= 9901 > F_1 = 9853, \\
m_2 &= 7937 > F_2 = 6792, \\
m_3 &= 5279 > F_3 = 3761, \\
m_4 &= 6997 > F_4 = 5102.
\end{aligned}$$

- (1) What are the four write keys  $e_1$ ,  $e_2$ ,  $e_3$  and  $e_4$  used in the encryption process?
- (2) What is the encrypted text  $C$  corresponding to  $D$ ?
- (3) If  $F_1$  is changed from  $F_1 = 9853$  to  $F_1 = 9123$ , what is the new value of the encrypted text  $C$ ?

To protect a database, we can encrypt it by using encryption keys. To protect encryption keys, however, we will need some different methods. In the next subsection, we shall introduce a method for protecting the cryptographic keys.

### 3.3.12 Secret Sharing

Liu [145] considers the following problem: eleven scientists are working on a secret project. They wish to lock up the documents in a cabinet such that the cabinet can be opened if and only if six or more of the scientists are present. What is the smallest number of locks needed? What is the smallest number of keys to the locks each scientist must carry? The minimal solution uses 462 locks and 252 keys. It is clear that these numbers are impractical, and they become exponentially worse when the number of scientists increases. In this section, we shall introduce an interesting method to solve similar problems. It is called secret sharing and was first proposed by Shamir in 1979 (see Mignotte [161] and Shamir [225]). The method can be very useful in the management of cryptographic keys and the keys for accessing the password file in a computer system.

**Definition 3.3.4.** A  $(k, n)$ -threshold scheme is a method for  $n$  people (or parties)  $P_1, P_2, \dots, P_n$  to share a secret  $S$  in such a way that the following properties hold:

- (1)  $k < n$ ,
- (2) each  $P_i$  has some information  $I_i$ ,
- (3) knowledge of any  $k$  of the  $\{I_1, I_2, \dots, I_n\}$  enables one to find  $S$  easily,
- (4) knowledge of less than  $k$  of the  $\{I_1, I_2, \dots, I_n\}$  does not enable one to find  $S$  easily.

Of course, there might be several ways to construct such a threshold scheme, but perhaps the simplest is the one based on congruence theory and the Chinese Remainder Theorem. It can be shown (Krause [134]) by the Chinese Remainder Theorem that:

**Theorem 3.3.3.** *For all  $2 \leq k \leq n$ , there exists a  $(k, n)$ -threshold scheme.*

In what follows, we shall introduce an algorithm for constructing a  $(k, n)$ -threshold scheme.

**Algorithm 3.3.9 (Secret sharing).** This algorithm is divided into two parts: the first part aims to construct a secret set  $\{I_1, I_2, \dots, I_n\}$ , whereas the second part aims to find out the secret  $S$  by any  $k$  of the  $\{I_1, I_2, \dots, I_n\}$ . Throughout the algorithm,  $S$  denotes the secret.

Part I: Construction of the secret set  $\{I_1, I_2, \dots, I_n\}$ .

- [1] Let the threshold sequence  $m_1, m_2, \dots, m_n$  be positive integers  $> 1$  such that  $\gcd(m_i, m_j) = 1$  for  $i \neq j$  and

$$m_1 m_2 \cdots m_k > m_n m_{n-1} \cdots m_{n-k+2}. \quad (3.114)$$

- [2] Determine the secret  $S$  in such a way that

$$\max(k-1) < S < \min(k) \quad (3.115)$$

where

$$\left. \begin{aligned} \min(k) &= m_1 m_2 \cdots m_k, \\ \max(k-1) &= m_n m_{n-1} \cdots m_{n-k+2}. \end{aligned} \right\} \quad (3.116)$$

- [3] Compute  $\{I_1, I_2, \dots, I_n\}$  in the following way:

$$\left. \begin{aligned} S &\equiv I_1 \pmod{m_1}, \\ S &\equiv I_2 \pmod{m_2}, \\ &\dots\dots\dots \\ S &\equiv I_n \pmod{m_n}. \end{aligned} \right\} \quad (3.117)$$

- [4] Compute  $M = m_1 m_2 \cdots m_n$ .

- [5] Send  $I_i$  and  $(m_i, M)$  to each  $P_i$ .

Part II: Recovering  $S$  from any  $k$  of these  $I_1, I_2, \dots, I_n$ : Suppose now parties  $\{P_{i_1}, P_{i_2}, \dots, P_{i_k}\}$  want to combine their knowledge  $\{I_{i_1}, I_{i_2}, \dots, I_{i_k}\}$  to find out  $S$ . (Each  $P_{i_j}, j = 1, 2, \dots, n$  has the triple  $(I_{i_j}, m_{i_j}, M)$  at hand).

- [1] Each  $P_{i_j}, j = 1, 2, \dots, k$  computes his own secret recovering key  $S_{i_j}$  as follows:

$$\left. \begin{aligned} M_{i_j} &= M/m_{i_j}, \\ N_{i_j} &= M_{i_j}^{-1} \pmod{m_{i_j}}, \\ S_{i_j} &= I_{i_j} M_{i_j} N_{i_j}. \end{aligned} \right\} \quad (3.118)$$

[2] Combine all the  $S_{i_j}$  to get the secret  $S$ :

$$S = \sum_{j=1}^k S_{i_j} \left( \text{mod } \prod_{j=1}^k m_{i_j} \right). \quad (3.119)$$

(By the Chinese Remainder Theorem, this computed  $S$  will be the required secret).

**Example 3.3.15.** Suppose we wish to construct a  $(k, n)$ -threshold scheme with  $k = 3$  and  $n = 5$ . The scheme administrator of a security agency first defines the following threshold sequence  $m_i$ :

$$\begin{aligned} m_1 &= 97, \\ m_2 &= 98, \\ m_3 &= 99, \\ m_4 &= 101, \\ m_5 &= 103, \end{aligned}$$

and computes:

$$\begin{aligned} M &= m_1 m_2 m_3 m_4 m_5 = 9790200882 \\ \min(k) &= m_1 m_2 m_3 = 941094 \\ \max(k-1) &= m_4 m_5 = 10403. \end{aligned}$$

He then defines the secret  $S$  to be in the range

$$10403 < S = 671875 < 941094$$

and calculates each  $I_i$  for each  $P_i$ :

$$\begin{aligned} S &\equiv I_1 \pmod{m_1} \implies I_1 = 53 \\ S &\equiv I_2 \pmod{m_2} \implies I_2 = 85 \\ S &\equiv I_3 \pmod{m_3} \implies I_3 = 61 \\ S &\equiv I_4 \pmod{m_4} \implies I_4 = 23 \\ S &\equiv I_5 \pmod{m_5} \implies I_5 = 6. \end{aligned}$$

Finally he distributes each  $I_i$  as well as  $m_i$  and  $M$  to each  $P_i$ , so that each  $P_i$  who shares the secret  $S$  has the triple  $(I_i, m_i, M)$ .

Suppose now  $P_1, P_2$  and  $P_3$  want to combine their knowledge  $\{I_1, I_2, I_3\}$  to find out  $S$ . They first individually compute:

$$\begin{aligned} M_1 &= M/m_1 = 100929906 \\ M_2 &= M/m_2 = 99900009 \\ M_3 &= M/m_3 = 98890918 \end{aligned}$$

and

$$\begin{aligned} N_1 &\equiv M_1^{-1} \pmod{m_1} \implies N_1 = 95 \\ N_2 &\equiv M_2^{-1} \pmod{m_2} \implies N_2 = 13 \\ N_3 &\equiv M_3^{-1} \pmod{m_3} \implies N_3 = 31. \end{aligned}$$

Hence, they get

$$\begin{aligned}
 S &\equiv I_1 \cdot M_1 \cdot N_1 + I_2 \cdot M_2 \cdot N_2 + I_3 \cdot M_3 \cdot N_3 \pmod{m_1 \cdot m_2 \cdot m_3} \\
 &\equiv 53 \cdot 100929906 \cdot 95 + 85 \cdot 99900009 \cdot 13 + 61 \cdot 98890918 \cdot 31 \\
 &\quad \pmod{97 \cdot 98 \cdot 99} \\
 &\equiv 805574312593 \pmod{941094} \\
 &= 671875.
 \end{aligned}$$

Suppose, alternatively,  $P_1, P_4$  and  $P_5$  wish to combine their knowledge  $\{I_1, I_4, I_5\}$  to find out  $S$ . They do the similar computations as follows:

$$\begin{aligned}
 M_1 &= M/m_1 = 100929906 \\
 M_4 &= M/m_4 = 96932682 \\
 M_5 &= M/m_5 = 95050494
 \end{aligned}$$

and

$$\begin{aligned}
 N_1 &\equiv M_1^{-1} \pmod{m_1} \implies N_1 = 95 \\
 N_4 &\equiv M_4^{-1} \pmod{m_4} \implies N_4 = 61 \\
 N_5 &\equiv M_5^{-1} \pmod{m_5} \implies N_5 = 100.
 \end{aligned}$$

Therefore,

$$\begin{aligned}
 S &\equiv I_1 \cdot M_1 \cdot N_1 + I_4 \cdot M_4 \cdot N_4 + I_5 \cdot M_5 \cdot N_5 \pmod{m_1 \cdot m_4 \cdot m_5} \\
 &\equiv 53 \cdot 100929906 \cdot 95 + 23 \cdot 96932682 \cdot 61 + 6 \cdot 95050494 \cdot 100 \\
 &\quad \pmod{97 \cdot 101 \cdot 103} \\
 &\equiv 701208925956 \pmod{1009091} \\
 &= 671875.
 \end{aligned}$$

However, knowledge of less than 3 of these  $I_1, I_2, I_3, I_4, I_5$  is insufficient to find out  $S$ . For example, you cannot expect to find out  $S$  just by combining  $I_1$  and  $I_4$ :

$$\begin{aligned}
 S' &\equiv I_1 \cdot M_1 \cdot N_1 + I_4 \cdot M_4 \cdot N_4 \pmod{m_1 \cdot m_4} \\
 &\equiv 53 \cdot 100929906 \cdot 95 + 23 \cdot 96932682 \cdot 61 \pmod{97 \cdot 101} \\
 &\equiv 644178629556 \pmod{9791} \\
 &= 5679.
 \end{aligned}$$

Clearly, this is not the correct value of  $S$ . Of course, you can find out  $S$  by any 3 or more of the  $I_1, I_2, I_3, I_4, I_5$ .

**Exercise 3.3.12.** In the above context, find out  $S$  if  $P_1, P_3, P_4, P_5$  wish to combine their knowledge  $\{I_1, I_3, I_4, I_5\}$  to find out  $S$ .

**Exercise 3.3.13.** Suppose a security agency defines a  $(5, 7)$ -threshold scheme and sends each triple  $(I_i, m_i, M)$  defined as follows to each person  $P_i$  for  $i = 1, 2, \dots, 7$ , who shares the secret  $S$ :

$$\begin{aligned}
(I_1, m_1) &= (824, 1501) \\
(I_2, m_2) &= (1242, 1617) \\
(I_3, m_3) &= (1602, 1931) \\
(I_4, m_4) &= (1417, 5573) \\
(I_5, m_5) &= (3090, 6191) \\
(I_6, m_6) &= (281, 7537) \\
(I_7, m_7) &= (6261, 9513) \\
M &= 1501 \cdot 1617 \cdot 1917 \cdot 3533 \cdot 9657 \cdot 10361 \cdot 53113 \\
&= 11594148137520792605086941
\end{aligned}$$

Now suppose parties  $P_1, P_3, P_5, P_6, P_7$  wish to combine their knowledge  $\{I_1, I_3, I_5, I_6, I_7\}$  to find out  $S$ . What is the  $S$ ? Suppose also parties  $P_2, P_3, P_4, P_5, P_6$  wish to combine their knowledge  $\{I_2, I_3, I_4, I_5, I_6\}$  to find out  $S$ . What is the  $S$  then? (The two  $S$ 's should be the same.)

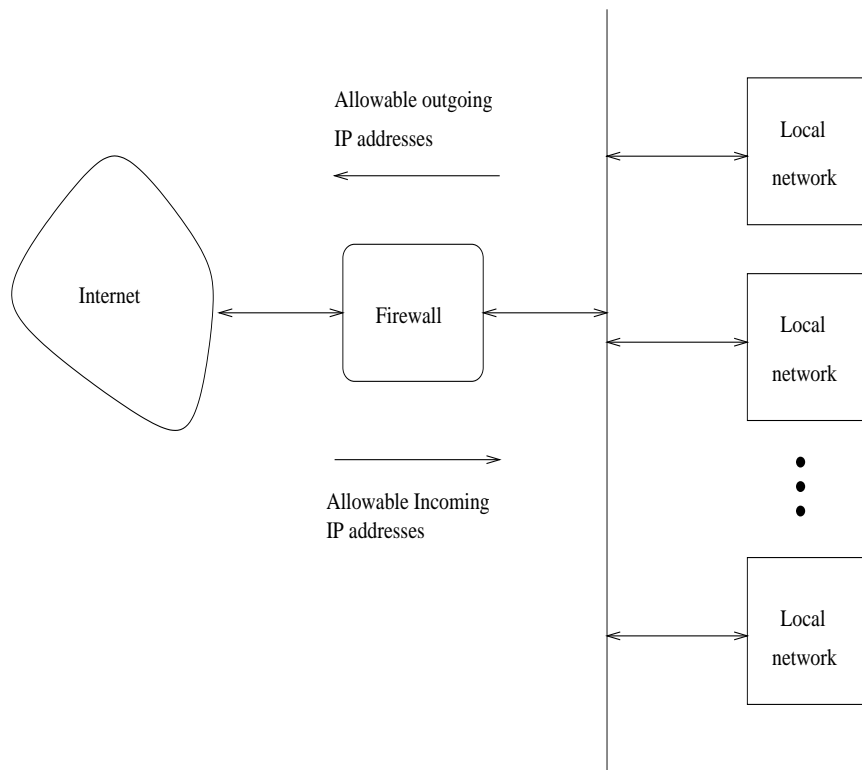
### 3.3.13 Internet/Web Security and Electronic Commerce

It is easy to run a secure computer system. You merely have to disconnect all dial-up connections and permit only direct-wired terminals, put the machine and its terminals in a shielded room, and post a guard at the door.

GRAMPP AND MORRIS  
UNIX Operating System Security [91]

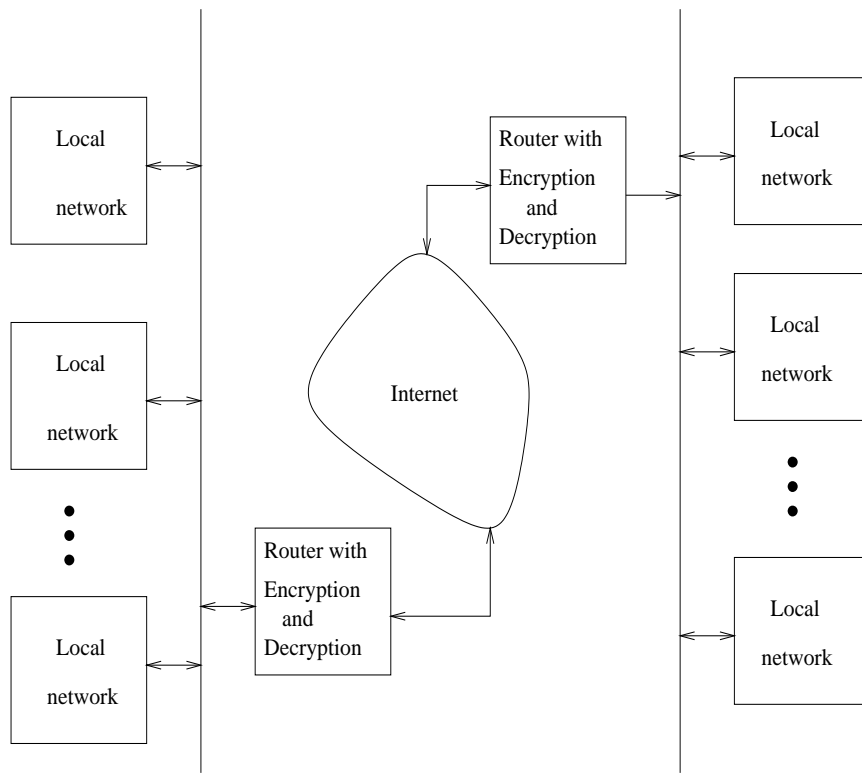
The security mentioned in the above quotation is unfortunately not what we need, though it is easy to achieve; an isolated and disconnected computer system is essentially a useless system in modern days. We would like such a (local network) system which is fully connected to the Internet but still be as secure as a disconnected system. How can we achieve such a goal? The first method to secure the local system is to introduce a firewall (security gateway) to protect a local system against intrusion from outside sources. An Internet firewall serves the same purpose as firewalls in buildings: to protect a certain area from the spread of fire and a potentially catastrophic explosion. It is used to examine the Internet addresses on packets or ports requested on incoming connections to decide what traffic is allowed into the local network. The simplest form of a firewall is the packet filter, as shown in Figure 3.14. It basically keeps a record of allowable sources and destination IP addresses and deletes all packets which do not have these addresses. Unfortunately, this firewalling technique suffers from the fact that IP addresses<sup>19</sup> can be easily forged. For example, a “hacker” might determine the list of good source addresses and then add one of these addresses to any packets which are addressed into the local network. Although some extra layers of security can

<sup>19</sup> An Internet Protocol address (IP address), or just Internet address, is a unique 32-bit binary number assigned to a host and used for all communication with the host.



**Figure 3.14.** Packet filter firewalls

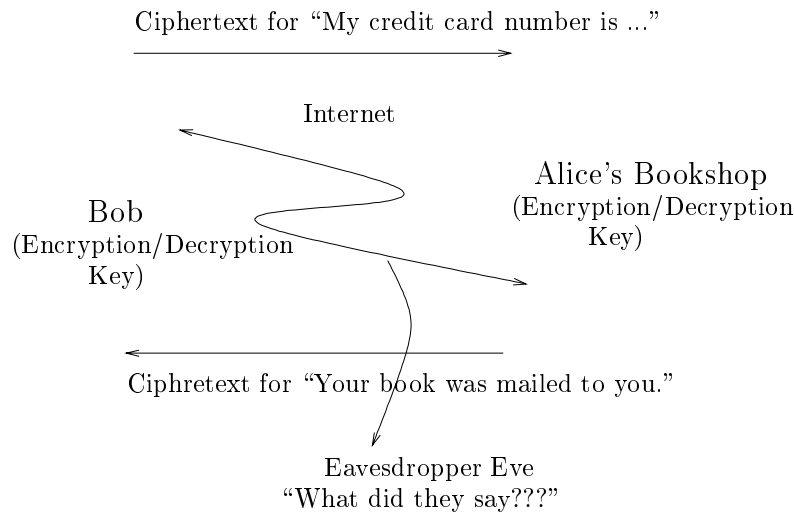
be added into a firewall, it is generally still not powerful enough to protect a local system against intrusion from outside unfriendly users in the Internet. It is worthwhile pointing out that all networked systems have holes in them by which someone else can slip into. For example, recently the U.S. Federal Bureau of Investigation (FBI) estimated that \$7.5 billion are lost annually to electronic attack and the U.S. Department of Defence (DOD) says that in 96% of the cases where the crackers got in, they went undetected. The best method of protection for a local network system is to encrypt all the information stored in the local system and to decrypt it whenever an authorized user wants to use the information. This method has an an important application in secure communications – to encrypt the data leaving the local network and then to decrypt it on the remote site; only friendly sites will have the required encryption/decryption key to receive or to send data, and only the routers which connect to the Internet require to encrypt/decrypt. This technique is known as the cryptographic tunnels (see Figure 3.15), which has the extra advantage that data cannot be easily *tapped-into* (Buchanan [42]). A further



**Figure 3.15.** Cryptographic tunnels

development of the cryptographic tunnels is the Virtual Private Networks technologies [264], which use tunneling to create a private network so as to keep communication private.

Cryptographic tunnels have important applications in secure communications and digital payments, or more generally, the electronic commerce over the insecure Internet/World Wide Web. For example, if Bob wants to order a book from Alice's bookshop (see Figure 3.16), he uses the secure tunnel to send Alice his credit card number; on receiving Bob's credit card number, Alice sends Bob the required book. It is worthwhile pointing out that a great deal of effort has been put into commercial cryptographic-based Internet/Web security in recent years. Generally speaking, there are two categories of commercial cryptographic systems used for securing the Internet/Web communications. The first group are programs and protocols that are used for encryption of e-mail messages. These programs take a plaintext message, encrypt it and either store the encrypted message on a local machine or transmit it



**Figure 3.16.** Electronic book ordering

to another user over the Internet. Some popular systems that fall into this category include the following:

- (1) Pretty Good Privacy (PGP): PGP is a program created by Philip Zimmermann to encrypt e-mails using public-key cryptography. PGP was electronically published as free software in 1991. It has now become the worldwide de facto standard for e-mail encryption.
- (2) Secure/Multipurpose Internet Mail Extensions (S/MIME): S/MIME is a security enhancement to the MIME Internet e-mail format standard, based on technology from RSA Data Security. Although both PGP and S/MIME are on an IETF (Internet Engineering Task Force) standards track, it appears likely that S/MIME will emerge as the industry standard for commercial and organizational use, while PGP will remain the choice for personal e-mail security for many users.

The second category of cryptographic systems are network protocols used for providing confidentiality, authentication, integrity, and nonrepudiation in a networked environment. These systems require real-time interplay between a client and a server to work properly. Listed below are some systems falling into this category:

- (1) Secure Sockets Layer protocol (SSL): SSL is developed by Netscape Communications, and supported by Netscape and Microsoft browsers. It provides a secure channel between client and server which ensures privacy of data, authentication of the session partners and message integrity.
- (2) Private Communication Technology protocol (PCT): PCT, proposed by Microsoft, is a slightly modified version of SSL. The Internet Engineering



Task Force (IETF) is in the process of creating a Transport Secure Layer (TSL) to merge the SSL and PCT.

- (3) Secure HyperText Transport Protocol (S-HTTP): S-HTTP is developed by Enterprise Integration Technologies (EIT). It uses a modified version of HTTP clients and the server to allow negotiation of privacy, authentication and integrity characteristics.
- (4) Secure Transaction Technology Protocol (STT): STT is a standard developed jointly by Microsoft and Visa International to enable secure credit card payment and authorisation over the web.
- (5) Secure Electronic Payment Protocol (SEPP): SEPP is another electronic payments scheme, sponsored by MasterCard and developed in association with IBM, Netscape, CyberCash and GTE. Both STT and SEPP have been superseded by SET (Secure Electronic Transactions), proposed jointly by MasterCard and Visa.

**Exercise 3.3.14.** Try to order a copy of a book, e.g., the present book, from Springer-Verlag by using your SSL-aware web browser to create an encrypted connection to the Springer-Verlag web server:

`https://www.springer.de`

Now we are in a position to discuss a real-world commercial cryptographic protocol, the SET protocol for secure credit card payment over the insecure Internet. It is a simplified version of the SET, based on a description given in [87].

**Algorithm 3.3.10 (SET protocol).** This algorithm describes a cryptographic protocol for credit card payment over the Internet. Suppose that Alice wants to purchase a book from Bob (an Internet bookshop) using the credit card issued by Lisa (a bank), but Alice does not want Bob to see her credit card number, however she wants Bob to send her the book and Lisa to send Bob the payment. And of course, Alice also wants that the communications between Bob, Lisa and herself is kept confidential even if someone is eavesdropping over the Internet.

- [1] Alice first prepares two documents: a purchase order  $O$  stating she wants to order a book from Bob, and a payment slip  $P$ , providing Lisa the card number to be used in the transaction, and the amount to be charged. Then she computes the digests:

$$\left. \begin{array}{l} o = H(O) \\ p = H(P) \end{array} \right\} \quad (3.120)$$

and produces a digital signature  $S$  for the digest of the concatenation of  $o$  and  $p$ :

$$S = D_A(H(o \parallel p)) = D_A(H(H(O) \parallel H(P))) \quad (3.121)$$

where  $D_A$  is the function used by Alice to sign, based on her private key. Alice encrypts the concatenation of  $o$ ,  $P$  and  $S$  with Lisa's public key, which yields the ciphertext:

$$C_L = E_L(o \parallel P \parallel S). \quad (3.122)$$

She also encrypts with Bob's public key the concatenation of  $O$ ,  $p$  and  $S$  and gets the ciphertext:

$$C_B = E_B(O \parallel p \parallel S). \quad (3.123)$$

She then sends  $C_L$  and  $C_B$  to Bob.

- [2] Bob retrieves  $O$ ,  $p$  and  $S$  by decrypting  $C_B$  with his private key. He verifies the authenticity of the purchase order  $O$  with Alice's public key by checking that

$$E_A(S) = H(H(O \parallel p)) \quad (3.124)$$

and forwards  $C_L$  to Lisa.

- [3] Lisa retrieves  $o$ ,  $P$  and  $S$  by decrypting  $C_L$  with private key. She verifies the authenticity of the payment slip  $P$  with Alice's public key by checking that

$$E_A(S) = H(o \parallel H(P)) \quad (3.125)$$

and verifies that  $P$  indicates a payment to Bob. She then creates an authorization message  $M$  that consists of a transaction number, Alice's name, and the amount she agreed to pay. Lisa computes the signature  $T$  of  $M$ , encrypts the pair  $(M, T)$  with Bob's public key to get the ciphertext:

$$C_M = E_B(M \parallel T) \quad (3.126)$$

and sends it to Bob.

- [4] Bob retrieves  $M$  and  $T$  by decrypting  $C_M$  and verifies the authenticity of the authorization message  $M$  with Lisa's public key, by checking that

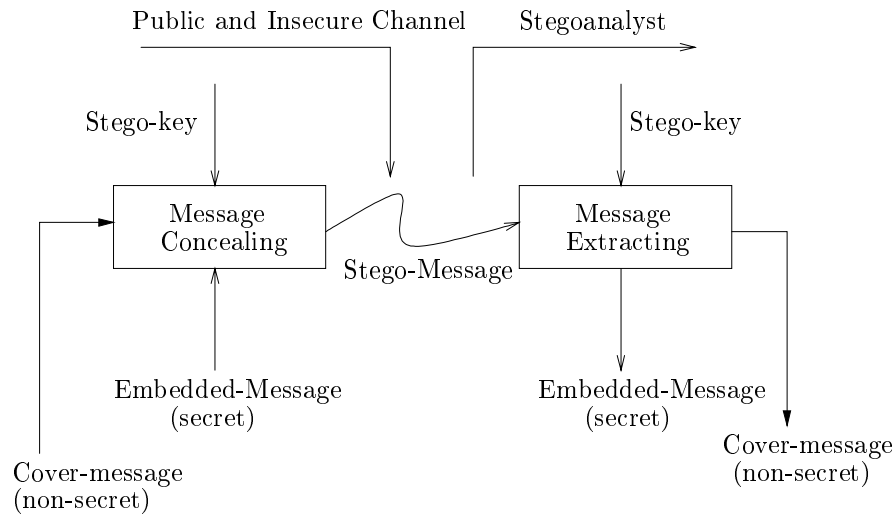
$$E_L(T) = M. \quad (3.127)$$

He verifies that the name in  $M$  is Alice's, and that the amount is the correct price of the book. He fulfills the order by sending the book to Alice and requests the payment from Lisa by sending her the transaction number encrypted with Lisa's public key.

- [5] Lisa pays Bob and charges Alice's credit card account.

### 3.3.14 Steganography

Cryptography means “secret writing”. A closely related area to cryptography is *steganography*, which literally means *covered writing* as derived from Greek and deals with the hiding of messages so that the potential monitors do not even know that a message is being sent. It is different from cryptography where they know that a secret message is being sent. Figure 3.17 shows a schematic diagram of a typical steganography system. Generally, the sender



**Figure 3.17.** A steganographic system

performs the following operations:

- (1) write a non-secret cover-message,
- (2) produce a stego-message by concealing a secret embedded message on the cover-message by using a stego-key,
- (3) send the stego-message over the insecure channel to the receiver.

At the other end, on receiving the stego-message, the intended receiver extracts the secret embedded message from the stego-message by using a pre-agreed stego-key (often the same key as used in the message concealing). Historical tricks include invisible inks, tiny pin punctures on selected characters, minute differences between handwritten characters, etc. For example, Kahn tells of a classical Chinese practice of embedding a code ideogram at a prearranged place in a dispatch (Kahn [117]). More recently, people have hidden secret messages in graphic images by replacing the least significant bits of the image with a secret message (Schneier [218]).

Note that the procedures of message concealing and message extracting in steganography are more or less the same as the message encryption and message decryption in cryptography. It is this reason that steganography is often used together with cryptography. For example, an encrypted message may be written using invisible ink. Note also that a steganographic system can either be secret or public. In a public-key steganographic system, different keys are used for message concealing and message extracting. Readers interested in steganography are suggested to consult the workshop proceedings on *Information Hiding* (Anderson [9] and Aucsmith [13]).

### 3.3.15 Quantum Cryptography

In Chapter 2, we introduced some quantum algorithms for factoring large integers and computing discrete logarithms. It is evident that if a quantum computer is available, then all the public-key cryptographic systems based on the difficulty of integer factorization and discrete logarithms will be insecure. However, the cryptographic systems based on quantum mechanics will still be secure even if a quantum computer is available. To make this book as complete as possible, we shall introduce in this subsection some basic ideas of quantum cryptography. More specifically, we shall introduce a quantum analog of the Diffie-Hellman key exchange/distribution system, proposed by Bennett and Brassard in 1984.

First let us define four *polarizations* as follows:

$$\{0^\circ, 45^\circ, 90^\circ, 135^\circ\} \stackrel{\text{def}}{=} \{\rightarrow, \nearrow, \uparrow, \nwarrow\}. \quad (3.128)$$

The quantum system consists of a transmitter, a receiver, and a quantum channel through which polarized photons can be sent [25]. By the law of quantum mechanics, the receiver can either distinguish between the *rectilinear polarizations*  $\{\rightarrow, \uparrow\}$ , or reconfigure to discriminate between the diagonal polarizations  $\{\nearrow, \nwarrow\}$ , but in any case, he cannot distinguish both types. The system works in the following way:

- [1] Alice uses the transmitter to send Bob a sequence of photons, each of them should be in one of the four polarizations  $\{\rightarrow, \nearrow, \uparrow, \nwarrow\}$ . For instance, Alice could choose, at random, the following photons

$$\uparrow \quad \nearrow \quad \rightarrow \quad \nwarrow \quad \rightarrow \quad \rightarrow \quad \nearrow \quad \uparrow \quad \uparrow$$

to be sent to Bob.

- [2] Bob then uses the receiver to measure the polarizations. For each photon received from Alice, Bob chooses, at random, the following type of measurements  $\{+, \times\}$ :

+      +      ×      ×      +      ×      ×      ×      +

[3] Bob records the result of his measurements but keeps it secret:

↑      →      ↗      ↖      →      ↗      ↗      ↗      ↑

[4] Bob publicly announces the type of measurements he made, and Alice tells him which measurements were of correct type:

✓                      ✓      ✓                      ✓                      ✓

[5] Alice and Bob keep all cases in which Bob measured the correct type. These cases are then translated into bits  $\{0, 1\}$  and thereby become the key:

↑	↖	→	↗	↑
1	1	0	0	1

[6] Using this secret key formed by the quantum channel, Bob and Alice can now encrypt and send their ordinary messages via the classic public-key channel.

An eavesdropper is free to try to measure the photons in the quantum channel, but, according to the law of quantum mechanics, he cannot in general do this without disturbing them, and hence, the key formed by the quantum channel is secure.

### 3.4 Bibliographic Notes and Further Reading

We interpret *applied number theory* in this book as the application of number theory to computing and information technology, and thus this chapter is mainly concerned with these applications of number theory. Even with this restriction, we argue that it is impossible to discuss all the computing related applications of number theory in a single book. We have, in fact, only discussed the applications of number theory to the design of computer systems and cryptosystems.

Our first application of number theory in computing is the design of computer systems; these include residue number systems and residue computers, complementary arithmetic and fast adders, error detections and corrections, the construction of hash functions (particularly minimal perfect hash functions), and the generation of random numbers/bits. Our aim was to show the applicability of number theory in computer systems design rather than the actual design of the computer (hardware or software) systems. There are

plenty of books available on computer arithmetic (including residue number systems and complementary arithmetic) and fast computer architectures, but those by Koren [132], McClellan and Radar [149], Soderstrand et al. [243], and Szabo and Tanaka [247] are highly recommended. A standard reference that contains many applications of number theory in computer arithmetic, random number generation and hash functions (and many more) is Knuth's three volumes of *The Art of Computer Programming* [122], [123], and [124]. For error detection and correction codes, see, for example, Gallian [77], Hill [104], and Welsh [252].

Cryptography, particularly public-key cryptography, is an area that heavily depends on ideas and methods from number theory; of course, number theory is also useful in information systems security, including communication network security. In this chapter, we have provided a mathematical foundation for cryptography and information security. Those who desire a more detailed exposition in the field are invited to consult Bauer [20], Koblitz [128] and [129], and Pinch [184]; for elliptic curve public-key cryptography, see Menezes [155]. Readers may also find the following books useful in cryptography and computer security: Jackson [112], Kaufman et al. [118], Pfleeger [182], Salomaa [215], Smith [242], Stinson [246] and Welsh [252]. The books edited by Pomerance [190] and [44] contain a number of excellent survey papers on cryptology and random number generation.

The series of conferences proceedings entitled *Advances in Cryptology* published in Lecture Notes in Computer Science by Springer-Verlag is an important source for new developments in cryptography and information security.

There is a special section on *computer and network security* in the *Scientific American*, **279**, 4(1998), 69–89; it contains the following articles:

- [1] C. P. Meinel, “How Hackers Break in ... and How They Are Caught”, pp 70–77.
- [2] “How Computer Security Works”,
  - [i] W. Cheswick and S. M. Bellovin, “Firewalls”, pp 78–79.
  - [ii] W. Ford, “Digital Certificates”, page 80.
  - [iii] J. Gosling, “The Java Sandbox”, page 81.
- [3] P. R. Zimmermann, “Cryptography for the Internet”, pp 82–87.
- [4] R. L. Rivest, “The Case Against Regulating Encryption Technology”, pp 88–89.

An issue of the IEEE journal *Computer*, **31**, 9(1998), also has a special report on *computer and network security*, which contains the following six papers:

- [1] P. W. Dowd and J. T. McHenry, “Network Security: It’s Time to Take It Seriously”, pp 24–28.
- [2] B. Schneier, “Cryptographic Design Vulnerabilities”, pp 29–33.

- [3] A. D. Rubin and D. E. Geer Jr, “A Survey on Web Security”, pp 34-42.
- [4] R. Oppliger, “Security at the Internet Layer”, pp 43-47.
- [5] W. A. Arbaugh, et al., “Security for Virtual Private Intranets”, pp 48-56.
- [6] T. D. Tarman, et al., “Algorithm-Agile Encryption in ATM Networks”, pp 57-64.

Note that the paper by Rubin and Geer [213] also discussed some interesting issues in mobile code security. All the above mentioned papers are easy to read and hence suitable for beginners in the field.

As by-products to cryptography, we have also introduced some basic concepts of steganography and quantum cryptography. There has been an increasing number of references in these two fields in recent years; interested readers are referred to, for example, Anderson [9], Aucsmith [13], Hughes [106], Inamori [110] and Lo [146], and the references therein.

In addition to computing and cryptography, number theory has also been successfully applied to many other areas such as physics, chemistry, acoustics, biology, engineering, dynamical systems, digital communications, digital signal processing, graphics design, self-similarity, and even music. For more information about these applications, readers are invited to consult Burr [44], Schroeder [222] and Waldschmidt, Moussa, Luck and Itzykson [250].



<http://www.springer.com/978-3-540-43072-8>

Number Theory for Computing

Yan, S.Y.

2002, XXII, 435 p., Hardcover

ISBN: 978-3-540-43072-8