CHAPTER 2

Mathematics of Cryptography

Part I: Modular Arithmetic, Congruence, and Matrices

Objectives

This chapter is intended to prepare the reader for the next few chapters in cryptography. The chapter has several objectives:

- ❏ To review integer arithmetic, concentrating on divisibility and finding the greatest common divisor using the Euclidean algorithm
- ❏ To understand how the extended Euclidean algorithm can be used to solve linear Diophantine equations, to solve linear congruent equations, and to find the multiplicative inverses
- ❏ To emphasize the importance of modular arithmetic and the modulo operator, because they are extensively used in cryptography
- ❏ To emphasize and review matrices and operations on residue matrices that are extensively used in cryptography
- ❏ To solve a set of congruent equations using residue matrices

Cryptography is based on some specific areas of mathematics, including number theory, linear algebra, and algebraic structures. In this chapter, we discuss only the topics in the above areas that are needed to understand the contents of the next few chapters. Readers who are familiar with these topics can skip this chapter entirely or partially. Similar chapters are provided throughout the book when needed. Proofs of theorems and algorithms have been omitted, and only their applications are shown. The interested reader can find proofs of the theorems and algorithms in Appendix P.

Proofs of theorems and algorithms discussed in this chapter can be found in Appendix P.

2.1 INTEGER ARITHMETIC

In **integer arithmetic,** we use a set and a few operations. You are familiar with this set and the corresponding operations, but they are reviewed here to create a background for modular arithmetic.

Set of Integers

The **set of integers,** denoted by **Z,** contains all integral numbers (with no fraction) from negative infinity to positive infinity (Figure 2.1).

Figure 2.1 *The set of integers*

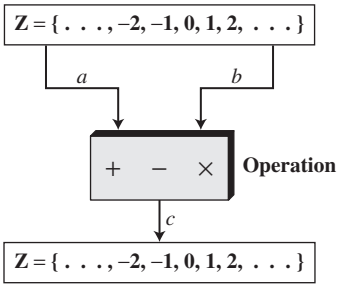
$$\mathbf{Z} = \{ \ldots, -2, -1, 0, 1, 2, \ldots \}$$

Binary Operations

In cryptography, we are interested in three binary operations applied to the set of integers. A **binary operation** takes two inputs and creates one output. Three common binary operations defined for integers are *addition, subtraction,* and *multiplication*. Each of these operations takes two inputs (*a* and *b*) and creates one output (*c*) as shown in Figure 2.2. The two inputs come from the set of integers; the output goes into the set of integers.

Note that *division* does not fit in this category because, as we will see shortly, it produces two outputs instead of one.

Figure 2.2 *Three binary operations for the set of integers*



Example 2.1

The following shows the results of the three binary operations on two integers. Because each input can be either positive or negative, we can have four cases for each operation.

Add:	$5 + 9 = 14$	$(-5) + 9 = 4$	$5 + (-9) = -4$	$(-5) + (-9) = -14$
Subtract:	$5 - 9 = -4$	$(-5) - 9 = -14$	$5 - (-9) = 14$	$(-5) - (-9) = +4$
Multiply:	$5 \times 9 = 45$	$(-5) \times 9 = -45$	$5 \times (-9) = -45$	$(-5) \times (-9) = 45$

Integer Division

In integer arithmetic, if we divide a by n, we can get q and r. The relationship between these four integers can be shown as

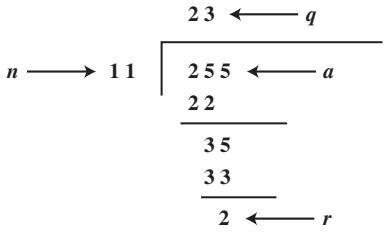
$$a = q \times n + r$$

In this relation, a is called the *dividend; q*, the *quotient; n*, the *divisor;* and r, the *remainder.* Note that this is not an operation, because the result of dividing a by n is two integers, q and r. We can call it *division relation*.

Example 2.2

Assume that $a = 255$ and $n = 11$. We can find $q = 23$ and $r = 2$ using the division algorithm we have learned in arithmetic as shown in Figure 2.3.

Figure 2.3 *Example 2.2, finding the quotient and the remainder*

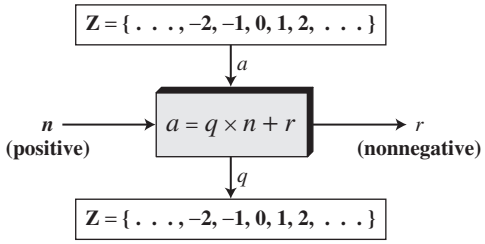


Most computer languages can find the quotient and the remainder using language-specific operators. For example, in the C language, the operator / can find the quotient and the operator % can find the remainder.

Two Restrictions

When we use the above division relationship in cryptography, we impose two restrictions. First, we require that the divisor be a positive integer ($n > 0$). Second, we require that the remainder be a nonnegative integer ($r \geq 0$). Figure 2.4 shows this relationship with the two above-mentioned restrictions.

Figure 2.4 *Division algorithm for integers*



Example 2.3

When we use a computer or a calculator, r and q are negative when a is negative. How can we apply the restriction that r needs to be positive? The solution is simple, we decrement the value of q by 1 and we add the value of n to r to make it positive.

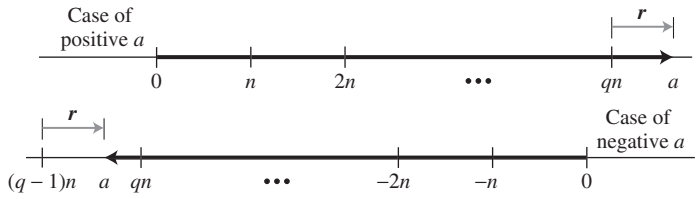
$$-255 = (-\mathbf{23} \times 11) + (-\mathbf{2}) \qquad \leftrightarrow \qquad -255 = (-\mathbf{24} \times 11) + \mathbf{9}$$

We have decremented −23 to become −24 and added 11 to −2 to make it 9. The above relation is still valid.

The Graph of the Relation

We can show the above relation with the two restrictions on n and r using two graphs in Figure 2.5. The first one shows the case when a is positive; the second when a is negative.

Figure 2.5 *Graph of division algorithm*



Starting from zero, the graph shows how we can reach the point representing the integer a on the line. In case of a positive a, we need to move $q \times n$ units to the right and then move extra r units in the same direction. In case of a negative a, we need to move $(q - 1) \times n$ units to the left (q is negative in this case) and then move r units in the opposite direction. In both cases the value of r is positive.

Divisibility

Let us briefly discuss **divisibility,** a topic we often encounter in cryptography. If a is not zero and we let $r = 0$ in the division relation, we get

$$\boldsymbol{a = q \times n}$$

We then say that n divides a (or n is a divisor of a). We can also say that a is divisible by n. When we are not interested in the value of q, we can write the above relationship as $\boldsymbol{a | n}$. If the remainder is not zero, then n does not divide a and we can write the relationship as $\boldsymbol{a \nmid n}$.

Example 2.4

a. The integer 4 divides the integer 32 because $\mathbf{32} = 8 \times \mathbf{4}$. We show this as 4|32.
b. The number 8 does not divide the number 42 because $\mathbf{42} = 5 \times \mathbf{8} + 2$. There is a remainder, the number 2, in the equation. We show this as $8 \nmid 42$.

Example 2.5

a. We have $13|78$, $7|98$, $-6|24$, $4|44$, and $11|(-33)$.

b. We have $13 \nmid 27$, $7 \nmid 50$, $-6 \nmid 23$, $4 \nmid 41$, and $11 \nmid (-32)$.

Properties

Following are several properties of divisibility. The interested reader can check Appendix P for proofs.

Property 1: if $a|1$, then $a = \pm 1$.
Property 2: if $a|b$ and $b|a$, then $a = \pm b$.
Property 3: if $a|b$ and $b|c$, then $a|c$.
Property 4: if $a|b$ and $a|c$, then $a|(m \times b + n \times c)$, where m and n are arbitrary integers.

Example 2.6

a. Since $3|15$ and $15|45$, according to the third property, $3|45$.

b. Since $3|15$ and $3|9$, according to the fourth property, $3|(15 \times 2 + 9 \times 4)$, which means $3|66$.

All Divisors

A positive integer can have more than one divisor. For example, the integer 32 has six divisors: 1, 2, 4, 8, 16, and 32. We can mention two interesting facts about divisors of positive integers:

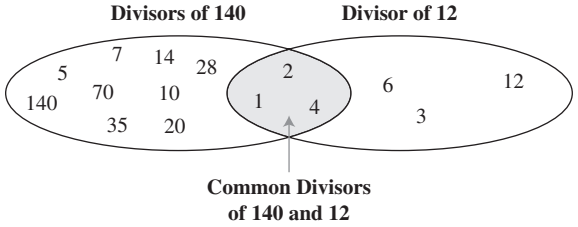
Fact 1: The integer 1 has only one divisor, itself.

Fact 2: Any positive integer has at least two divisors, 1 and itself (but it can have more).

Greatest Common Divisor

One integer often needed in cryptography is the **greatest common divisor** of two positive integers. Two positive integers may have many common divisors, but only one greatest common divisor. For example, the common divisors of 12 and 140 are 1, 2, and 4. However, the greatest common divisor is 4. See Figure 2.6.

Figure 2.6 *Common divisors of two integers*



The greatest common divisor of two positive integers is the largest integer that can divide both integers.

Euclidean Algorithm

Finding the greatest common divisor (gcd) of two positive integers by listing all common divisors is not practical when the two integers are large. Fortunately, more than 2000 years ago a mathematician named Euclid developed an algorithm that can find the greatest common divisor of two positive integers. The **Euclidean algorithm** is based on the following two facts (see Appendix P for the proof):

Fact 1: gcd $(a, 0) = a$

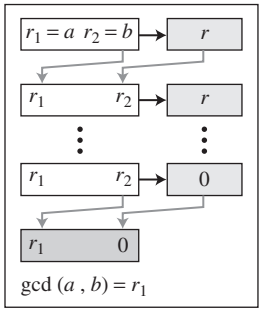
Fact 2: gcd (a, b) = gcd (b, r), where r is the remainder of dividing a by b

The first fact tells us that if the second integer is 0, the greatest common divisor is the first one. The second fact allows us to change the value of a, b until b becomes 0. For example, to calculate the gcd (36, 10), we can use the second fact several times and the first fact once, as shown below.

$$\text{gcd } (36, 10) = \text{gcd } (10, 6) = \text{gcd } (6, 4) = \text{gcd } (4, 2) = \text{gcd } (2, 0) = 2$$

In other words, gcd (36, 10) = 2, gcd (10, 6) = 2, and so on. This means that instead of calculating gcd (36, 10), we can find gcd (2, 0). Figure 2.7 shows how we use the above two facts to calculate gcd (a, b).

Figure 2.7 *Euclidean algorithm*



a. Process

```
r1 = a; r2 = b;   (Initialization)
while (r2 > 0)
 {
  q = r1 / r2;
  r = r1 - q * r2;
  r1 = r2; r2 = r;
 }

 gcd (a, b) = r1
```

b. Algorithm

We use two variables, r_1 and r_2, to hold the changing values during the process of reduction. They are initialized to a and b. In each step, we calculate the remainder of r_1 divided by r_2 and store the result in the variable r. We then replace r_1 by r_2 and r_2 by r. The steps are continued until r_2 becomes 0. At this moment, we stop. The gcd (a, b) is r_1.

When gcd (a, b) = 1, we say that a and b are relatively prime.

Find the greatest common divisor of 2740 and 1760.

Solution

We apply the above procedure using a table. We initialize r_1 to 2740 and r_2 to 1760. We have also shown the value of q in each step. We have gcd (2740, 1760) = 20.

q	r_1	r_2	r
1	2740	1760	980
1	1760	980	780
1	980	780	200
3	780	200	180
1	200	180	20
9	180	20	0
	20	0	

Example 2.7

Find the greatest common divisor of 25 and 60.

Solution

We chose this particular example to show that it does not matter if the first number is smaller than the second number. We immediately get our correct ordering. We have gcd (25, 65) = 5.

q	r_1	r_2	r
0	25	60	25
2	60	25	10
2	25	10	5
2	10	5	0
	5	0	

The Extended Euclidean Algorithm

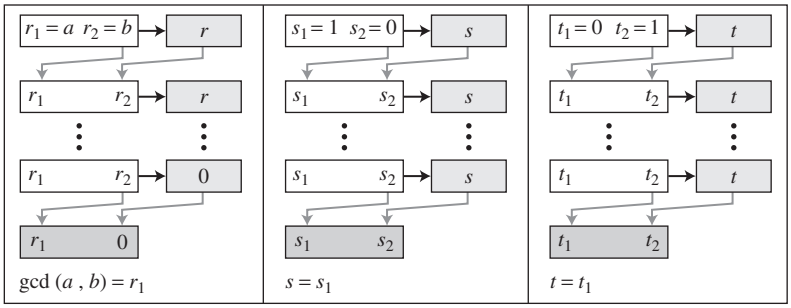
Given two integers a and b, we often need to find other two integers, s and t, such that

$$s \times a + t \times b = \gcd(a, b)$$

The **extended Euclidean algorithm** can calculate the gcd (a, b) and at the same time calculate the value of s and t. The algorithm and the process is shown in Figure 2.8.

As shown in Figure 2.8, the extended Euclidean algorithm uses the same number of steps as the Euclidean algorithm. However, in each step, we use three sets of calculations and exchanges instead of one. The algorithm uses three sets of variables, r's, s's, and t's.

Figure 2.8 *Extended Euclidean algorithm*



a. Process

```
r1 = a;  r2 = b;
s1 = 1;  s2 = 0;  (Initialization)
t1 = 0;  t2 = 1;
while (r2 > 0)
 {
  q = r1 / r2;
  r = r1 - q * r2;
  r1 = r2;  r2 = r;   (Updating r's)
  s = s1 - q * s2;
  s1 = s2;  s2 = s;   (Updating s's)
  t = t1 - q * t2;
  t1 = t2;  t2 = t;   (Updating t's)
 }
  gcd (a , b) = r1   s = s1   t = t1
```

b. Algorithm

In each step, r_1, r_2, and r have the same values in the Euclidean algorithm. The variables r_1 and r_2 are initialized to the values of a and b, respectively. The variables s_1 and s_2 are initialized to 1 and 0, respectively. The variables t_1 and t_2 are initialized to 0 and 1, respectively. The calculations of r, s, and t are similar, with one warning. Although r is the remainder of dividing r_1 by r_2, there is no such relationship between the other two sets. There is only one quotient, q, which is calculated as $r_1 | r_2$ and used for the other two calculations.

Example 2.8

Given $a = 161$ and $b = 28$, find gcd (a, b) and the values of s and t.

Solution

$$r = r_1 - q \times r_2 \qquad s = s_1 - q \times s_2 \qquad t = t_1 - q \times t_2$$

We use a table to follow the algorithm.

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
5	161	28	21	1	0	1	0	1	−5
1	28	21	7	0	1	−1	1	−5	6
3	21	7	0	1	−1	4	−5	6	−23
	7	0		**−1**	4		**6**	−23	

We get gcd (161, 28) = 7, $s = -1$ and $t = 6$. The answers can be tested because we have

$$(-1) \times 161 + 6 \times 28 = 7$$

Example 2.9

Given $a = 17$ and $b = 0$, find gcd (a, b) and the values of s and t.

Solution
We use a table to follow the algorithm.

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
	17	0		**1**	**0**		**0**	**1**	

Note that we need no calculation for q, r, and s. The first value of r_2 meets our termination condition. We get gcd (17, 0) = 17, $s = 1$, and $t = 0$. This indicates why we should initialize s_1 to 1 and t_1 to 0. The answers can be tested as shown below:

$$(1 \times 17) + (0 \times 0) = 17$$

Example 2.10

Given $a = 0$ and $b = 45$, find gcd (a, b) and the values of s and t.

Solution
We use a table to follow the algorithm.

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
0	0	45	0	1	0	1	0	1	0
	45	0		0	1		**1**	0	

We get gcd (0, 45) = 45, $s = 0$, and $t = 1$. This indicates why we should initialize s_2 to 0 and t_2 to 1. The answer can be tested as shown below:

$$(0 \times 0) + (1 \times 45) = 45$$

Linear Diophantine Equations

Although we will see a very important application of the extended Euclidean algorithm in the next section, one immediate application is to find the solutions to the **linear Diophantine equations** of two variables, an equation of type $ax + by = c$. We need to find integer values for x and y that satisfy the equation. This type of equation has either no solution or an infinite number of solutions. Let $d = \gcd(a, b)$. If $d \nmid c$, then the equation has no solution. If $d \mid c$, then we have an infinite number of solutions. One of them is called the particular; the rest, general.

A linear Diophantine equation of two variables is $ax + by = c$.

Particular Solution

If d|c, a particular solution to the above equation can be found using the following steps:

1. Reduce the equation to $a_1x + b_1y = c_1$ by dividing both sides of the equation by d. This is possible because d divides a, b, and c by the assumption.
2. Solve for s and t in the relation $a_1s + b_1t = 1$ using the extended Euclidean algorithm.
3. The particular solution can be found:

***Particular solution:* $x_0 = (c/d)s$ *and* $y_0 = (c/d)t$**

General Solutions

After finding the particular solution, the general solutions can be found:

***General solutions:* $x = x_0 + k\,(b/d)$ *and* $y = y_0 - k\,(a/d)$ where $k = 0, 1, 2, \ldots$**

Example 2.11

Find the particular and general solutions to the equation $21x + 14y = 35$.

Solution

We have $d = \gcd(21, 7) = 7$. Since $7 \mid 35$, the equation has an infinite number of solutions. We can divide both sides by 7 to find the equation $3x + 2y = 5$. Using the extended Euclidean algorithm, we find s and t such as $3s + 2t = 1$. We have $s = 1$ and $t = -1$. The solutions are

Particular: $x_0 = 5 \times 1 = 5$ and $y_0 = 5 \times (-1) = -5$	since 35/7 = 5
General: $x = 5 + k \times 2$ and $y = -5 - k \times 3$	where $k = 0, 1, 2, \ldots$

Therefore, the solutions are $(5, -5)$, $(7, -8)$, $(9, -11)$, . . . We can easily test that each of these solutions satisfies the original equation.

Example 2.12

A very interesting application in real life is when we want to find different combinations of objects having different values. For example, imagine we want to cash a $100 check and get some $20 and some $5 bills. We have many choices, which we can find by solving the corresponding Diophantine equation $20x + 5y = 100$. Since $d = \gcd(20, 5) = 5$ and $5 \mid 100$, the equation

has an infinite number of solutions, but only a few of them are acceptable in this case (only answers in which both x and y are nonnegative integers). We divide both sides by 5 to get $4x + y = 20$. We then solve the equation $4s + t = 1$. We can find $s = 0$ and $t = 1$ using the extended Euclidean algorithm. The particular solutions are $x_0 = 0 \times 20 = 0$ and $y_0 = 1 \times 20 = 20$. The general solutions with x and y nonnegative are (0, 20), (1, 16), (2, 12), (3, 8), (4, 4), (5, 0). The rest of the solutions are not acceptable because y becomes negative. The teller at the bank needs to ask which of the above combinations we want. The first has no $20 bills; the last has no $5 bills.

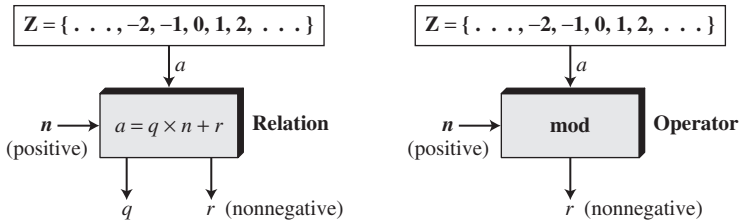
2.2 MODULAR ARITHMETIC

The division relationship ($a = q \times n + r$) discussed in the previous section has two inputs (a and n) and two outputs (q and r). In **modular arithmetic,** we are interested in only one of the outputs, the remainder r. We don't care about the quotient q. In other words, we want to know what is the value of r when we divide a by n. This implies that we can change the above relation into a binary operator with two inputs a and n and one output r.

Modulo Operator

The above-mentioned binary operator is called the **modulo operator** and is shown as *mod*. The second input (n) is called the **modulus.** The output r is called the **residue.** Figure 2.9 shows the division relation compared with the modulo operator.

Figure 2.9 *Division relation and modulo operator*



As Figure 2.9 shows, the modulo operator (**mod**) takes an integer (a) from the set **Z** and a positive modulus (n). The operator creates a nonnegative residue (r). We can say

$$a \bmod n = r$$

Example 2.13

Find the result of the following operations:

a. 27 mod 5
b. 36 mod 12
c. −18 mod 14
d. −7 mod 10

Solution

We are looking for the residue *r*. We can divide the *a* by *n* and find *q* and *r*. We can then disregard *q* and keep *r*.

a. Dividing 27 by 5 results in $r = 2$. This means that $27 \bmod 5 = 2$.
b. Dividing 36 by 12 results in $r = 0$. This means that $36 \bmod 12 = 0$.
c. Dividing −18 by 14 results in $r = -4$. However, we need to add the modulus (14) to make it nonnegative. We have $r = -4 + 14 = 10$. This means that $-18 \bmod 14 = 10$.
d. Dividing −7 by 10 results in $r = -7$. After adding the modulus to −7, we have $r = 3$. This means that $-7 \bmod 10 = 3$.

Set of Residues: $\mathbf{Z}_n$

The result of the modulo operation with modulus *n* is always an integer between 0 and $n - 1$. In other words, the result of *a* mod *n* is always a nonnegative integer less than *n*. We can say that the modulo operation creates a set, which in modular arithmetic is referred to as the **set of least residues modulo *n***, or $\mathbf{Z}_n$. However, we need to remember that although we have only one set of integers ($\mathbf{Z}$), we have infinite instances of the set of residues ($\mathbf{Z}_n$), one for each value of *n*. Figure 2.10 shows the set $\mathbf{Z}_n$ and three instances, $\mathbf{Z}_2$, $\mathbf{Z}_6$, and $\mathbf{Z}_{11}$.

Figure 2.10 *Some $\mathbf{Z}_n$ sets*

$$\mathbf{Z}_n = \{ 0, 1, 2, 3, \ldots, (n-1) \}$$

$$\mathbf{Z}_2 = \{ 0, 1 \} \qquad \mathbf{Z}_6 = \{ 0, 1, 2, 3, 4, 5 \} \qquad \mathbf{Z}_{11} = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \}$$

Congruence

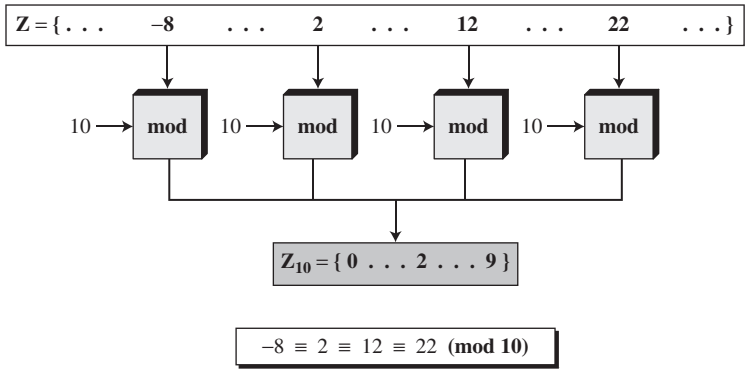
In cryptography, we often used the concept of **congruence** instead of equality. Mapping from $\mathbf{Z}$ to $\mathbf{Z}_n$ is not one-to-one. Infinite members of $\mathbf{Z}$ can map to one member of $\mathbf{Z}_n$. For example, the result of $2 \bmod 10 = 2$, $12 \bmod 10 = 2$, $22 \bmod 2 = 2$, and so on. In modular arithmetic, integers like 2, 12, and 22 are called congruent mod 10. To show that two integers are congruent, we use the **congruence operator** (≡). We add the phrase (mod *n*) to the right side of the congruence to define the value of modulus that makes the relationship valid. For example, we write:

$2 \equiv 12 \pmod{10}$	$13 \equiv 23 \pmod{10}$	$34 \equiv 24 \pmod{10}$	$-8 \equiv 12 \pmod{10}$
$3 \equiv 8 \pmod{5}$	$8 \equiv 13 \pmod{5}$	$23 \equiv 33 \pmod{5}$	$-8 \equiv 2 \pmod{5}$

Figure 2.11 shows the idea of congruence. We need to explain several points.

a. The congruence operator looks like the equality operator, but there are differences. First, an equality operator maps a member of $\mathbf{Z}$ to itself; the congruence operator maps a member from $\mathbf{Z}$ to a member of $\mathbf{Z}_n$. Second, the equality operator is one-to-one; the congruence operator is many-to-one.

Figure 2.11 *Concept of congruence*



b. The phrase (mod n) that we insert at the right-hand side of the congruence operator is just an indication of the destination set ($\mathbf{Z}_n$). We need to add this phrase to show what modulus is used in the mapping. The symbol *mod* used here does not have the same meaning as the binary operator. In other words, the symbol *mod* in 12 mod 10 is an operator; the phrase (mod 10) in $2 \equiv 12 \pmod{10}$ means that the destination set is $\mathbf{Z}_{10}$.

Residue Classes

A **residue class** $[a]$ or $[a]_n$ is the set of integers congruent modulo n. In other words, it is the set of all integers such that $x = a \pmod{n}$. For example, if $n = 5$, we have five sets [0], [1], [2], [3], and [4] as shown below:

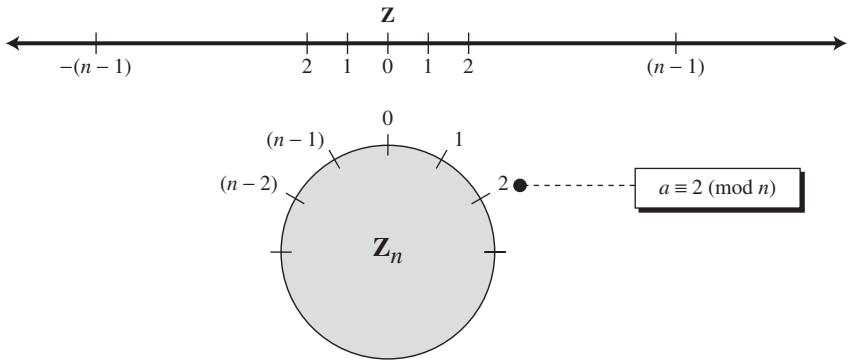
$$
\begin{aligned}
[0] &= \{\ldots, -15, -10, -5, 0, 5, 10, 15, \ldots\} \\
[1] &= \{\ldots, -14, -9, -4, 1, 6, 11, 16, \ldots\} \\
[2] &= \{\ldots, -13, -8, -3, 2, 7, 12, 17, \ldots\} \\
[3] &= \{\ldots, -12, -7, -5, 3, 8, 13, 18, \ldots\} \\
[4] &= \{\ldots, -11, -6, -1, 4, 9, 14, 19, \ldots\}
\end{aligned}
$$

The integers in the set [0] are all reduced to 0 when we apply the modulo 5 operation on them. The integers in the set [1] are all reduced to 1 when we apply the modulo 5 operation, and so on. In each set, there is one element called the least (nonnegative) residue. In the set [0], this element is 0; in the set [1], this element is 1; and so on. The set of all of these least residues is what we have shown as $\mathbf{Z}_5 = \{0, 1, 2, 3, 4\}$. In other words, the set $\mathbf{Z}_n$ is the set of all **least residue** modulo n.

Circular Notation

The concept of congruence can be better understood with the use of a circle. Just as we use a line to show the distribution of integers in $\mathbf{Z}$, we can use a circle to show the

Figure 2.12 *Comparison of* $\mathbf{Z}$ *and* $\mathbf{Z}_n$ *using graphs*



distribution of integers in $\mathbf{Z}_n$. Figure 2.12 shows the comparison between the two. Integers 0 to $n - 1$ are spaced evenly around a circle. All congruent integers modulo n occupy the same point on the circle. Positive and negative integers from $\mathbf{Z}$ are mapped to the circle in such a way that there is a symmetry between them.

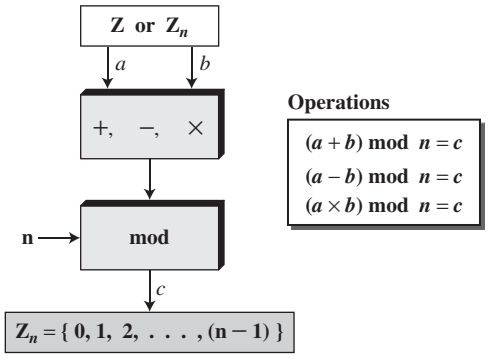
Example 2.14

We use modular arithmetic in our daily life; for example, we use a clock to measure time. Our clock system uses modulo 12 arithmetic. However, instead of a 0 we use the number 12. So our clock system starts with 0 (or 12) and goes until 11. Because our days last 24 hours, we navigate around the circle two times and denote the first revolution as A.M. and the second as P.M.

Operations in $\mathbf{Z}_n$

The three binary operations (*addition, subtraction,* and *multiplication*) that we discussed for the set $\mathbf{Z}$ can also be defined for the set $\mathbf{Z}_n$. The result may need to be mapped to $\mathbf{Z}_n$ using the mod operator as shown in Figure 2.13.

Figure 2.13 *Binary operations in* $\mathbf{Z}_n$



Actually, two sets of operators are used here. The first set is one of the binary operators (+, −, ×); the second is the mod operator. We need to use parentheses to emphasize the order of operations. As Figure 2.13 shows, the inputs (a and b) can be members of $\mathbf{Z}_n$ or $\mathbf{Z}$.

Example 2.15

Perform the following operations (the inputs come from $\mathbf{Z}_n$):

a. Add 7 to 14 in $\mathbf{Z}_{15}$.
b. Subtract 11 from 7 in $\mathbf{Z}_{13}$.
c. Multiply 11 by 7 in $\mathbf{Z}_{20}$.

Solution

The following shows the two steps involved in each case:

$$
\begin{aligned}
&(14 + 7) \bmod 15 \quad \rightarrow \quad (21) \bmod 15 = 6 \\
&(7 - 11) \bmod 13 \quad \rightarrow \quad (-4) \bmod 13 = 9 \\
&(7 \times 11) \bmod 20 \quad \rightarrow \quad (77) \bmod 20 = 17
\end{aligned}
$$

Example 2.16

Perform the following operations (the inputs come from either $\mathbf{Z}$ or $\mathbf{Z}_n$):

a. Add 17 to 27 in $\mathbf{Z}_{14}$.
b. Subtract 34 from 12 in $\mathbf{Z}_{13}$.
c. Multiply 123 by −10 in $\mathbf{Z}_{19}$.

Solution

The following shows the two steps involved in each case:

$$
\begin{aligned}
&(17 + 27) \bmod 14 \quad \rightarrow \quad (44) \bmod 14 = 2 \\
&(12 - 43) \bmod 13 \quad \rightarrow \quad (-31) \bmod 13 = 8 \\
&(123 \times (-10)) \bmod 19 \quad \rightarrow \quad (-1230) \bmod 19 = 5
\end{aligned}
$$

Properties

We mentioned that the two inputs to the three binary operations in the modular arithmetic can come from $\mathbf{Z}$ or $\mathbf{Z}_n$. The following properties allow us to first map the two inputs to $\mathbf{Z}_n$ (if they are coming from $\mathbf{Z}$) before applying the three binary operations (+, −, ×). Interested readers can find proofs for these properties in Appendix P.

First Property:	$(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$
Second Property:	$(a - b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$
Third Property:	$(a \times b) \bmod n = [(a \bmod n) \times (b \bmod n)] \bmod n$

Figure 2.14 shows the process before and after applying the above properties. Although the figure shows that the process is longer if we apply the above properties, we should remember that in cryptography we are dealing with very large integers. For example, if we multiply a very large integer by another very large integer, we