CHAPTER 9

Mathematics of Cryptography

Part III: Primes and Related Congruence Equations

Objectives

This chapter has several objectives:

- ❏ To introduce prime numbers and their applications in cryptography.
- ❏ To discuss some primality test algorithms and their efficiencies.
- ❏ To discuss factorization algorithms and their applications in cryptography.
- ❏ To describe the Chinese remainder theorem and its application.
- ❏ To introduce quadratic congruence.
- ❏ To introduce modular exponentiation and logarithm.

Asymmetric-key cryptography, which we will discuss in Chapter 10, is based on some topics in number theory, including theories related to primes, factorization of composites into primes, modular exponentiation and logarithm, quadratic residues, and the Chinese remainder theorem. These issues are discussed in this chapter to make Chapter 10 easier to understand.

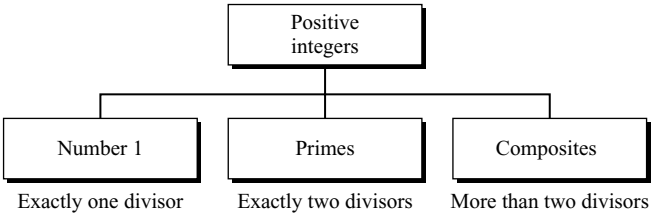
9.1 PRIMES

Asymmetric-key cryptography uses primes extensively. The topic of primes is a large part of any book on number theory. This section discusses only a few concepts and facts to pave the way for Chapter 10.

Definition

The positive integers can be divided into three groups: the number 1, primes, and composites as shown in Figure 9.1.

Figure 9.1 *Three groups of positive integers*



A positive integer is a **prime** *if and only if* it is exactly divisible by two integers, 1 and itself. A **composite** is a positive integer with more than two divisors.

A prime is divisible only by itself and 1.

Example 9.1

What is the smallest prime?

Solution

The smallest prime is 2, which is divisible by 2 (itself) and 1. Note that the integer 1 is not a prime according to the definition, because a prime must be divisible by two different integers, no more, no less. The integer 1 is divisible only by itself; it is not a prime.

Example 9.2

List the primes smaller than 10.

Solution

There are four primes less than 10: 2, 3, 5, and 7. It is interesting to note that the percentage of primes in the range 1 to 10 is 40%. The percentage decreases as the range increases.

Coprimes

Two positive integers, a and b, are **relatively prime,** or **coprime,** if gcd (a, b) = 1. Note that the number 1 is relatively prime with any integer. If p is a prime, then all integers 1 to $p - 1$ are relatively prime to p. In Chapter 2, we discussed set $\mathbf{Z}_n^*$ whose members are all relatively prime to n. Set $\mathbf{Z}_p^*$ is the same except that modulus (p) is a prime.

Cardinality of Primes

After the concept of primes has been defined, two questions naturally arise: Is there a finite number of primes or is the list infinite? Given a number n, how many primes are smaller than or equal to n?

Infinite Number of Primes

The number of primes is infinite. Here is an informal proof: Suppose that the set of primes is finite (limited), with p as the largest prime. Multiply the set of primes and call the result $P = 2 \times 3 \times \cdots \times p$. The integer (P + 1) cannot have a factor $q \leq p$. We know that q divides P. If q also divides (P + 1), then q divides (P + 1) – P = 1 The only number that divides 1 is 1, which is not a prime. Therefore, q is larger than p.

There is an infinite number of primes.

Example 9.3

As a trivial example, assume that the only primes are in the set {2, 3, 5, 7, 11, 13, 17}. Here P = 510510 and P + 1 = 510511. However, $510511 = 19 \times 97 \times 277$; none of these primes were in the original list. Therefore, there are three primes greater than 17.

Number of Primes

To answer the second question, a function called $\pi(n)$ is defined that finds the number of primes smaller than or equal to n. The following shows the values of this function for different n's.

$$\pi(1) = 0 \quad \pi(2) = 1 \quad \pi(3) = 2 \quad \pi(10) = 4 \quad \pi(20) = 8 \quad \pi(50) = 15 \quad \pi(100) = 25$$

But if n is very large, how can we calculate $\pi(n)$? The answer is that we can only use approximation. It has been shown that

$$[n / (\ln n)] \quad < \quad \pi(n) \quad < \quad [n/(\ln n - 1.08366)]$$

Gauss discovered the upper limit; Lagrange discovered the lower limit.

Example 9.4

Find the number of primes less than 1,000,000.

Solution

The approximation gives the range 72,383 to 78,543. The actual number of primes is 78,498.

Checking for Primeness

The next question that comes to mind is this: Given a number n, how can we determine if n is a prime? The answer is that we need to see if the number is divisible by all primes less than $\sqrt{n}$. We know that this method is inefficient, but it is a good start.

Example 9.5

Is 97 a prime?

Solution

The floor of $\sqrt{97} = 9$. The primes less than 9 are 2, 3, 5, and 7. We need to see if 97 is divisible by any of these numbers. It is not, so 97 is a prime.

Example 9.6

Is 301 a prime?

Solution

The floor of $\sqrt{301} = 17$. We need to check 2, 3, 5, 7, 11, 13, and 17. The numbers 2, 3, and 5 do not divide 301, but 7 does. Therefore 301 is not a prime.

Sieve of Eratosthenes

The Greek mathematician Eratosthenes devised a method to find all primes less than *n*. The method is called the **sieve of Eratosthenes.** Suppose we want to find all prime less than 100. We write down all the numbers between 2 and 100. Because $\sqrt{100} = 10$, we need to see if any number less than 100 is divisible by 2, 3, 5, and 7. Table 9.1 shows the result.

Table 9.1 *Sieve of Eratosthenes*

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

The following shows the process:

1. Cross out all numbers divisible by 2 (except 2 itself).
2. Cross out all numbers divisible by 3 (except 3 itself).
3. Cross out all numbers divisible by 5 (except 5 itself).
4. Cross out all numbers divisible by 7 (except 7 itself).
5. The numbers left over are primes.

Euler's Phi-Function

Euler's phi-function, $\phi(n)$, which is sometimes called the **Euler's totient function** plays a very important role in cryptography. The function finds the number of integers that are both smaller than *n* and relatively prime to *n*. Recall from Chapter 2 that the set $\mathbf{Z}_n^*$ contains the numbers that are smaller than *n* and relatively prime to *n*. The function $\phi(n)$ calculates the number of elements in this set. The following helps to find the value of $\phi(n)$.

1. $\phi(1) = 0$.
2. $\phi(p) = p - 1$ if p is a prime.

3. $\phi(m \times n) = \phi(m) \times \phi(n)$ if m and n are relatively prime.
4. $\phi(p^e) = p^e - p^{e-1}$ if p is a prime.

We can combine the above four rules to find the value of $\phi(n)$. For example, if n can be factored as $n = {p_1}^{e1} \times {p_2}^{e2} \times \cdots \times {p_k}^{ek}$, then we combine the third and the fourth rule to find

$$\phi(n) = ({p_1}^{e_1} - {p_1}^{e_1-1}) \times ({p_2}^{e_2} - {p_2}^{e_2-1}) \times \cdots \times ({p_k}^{e_k} - {p_k}^{e_k-1})$$

It is very important to notice that the value of $\phi(n)$ for large composites can be found only if the number n can be factored into primes. In other words, the difficulty of finding $\phi(n)$ depends on the difficulty of finding the factorization of n, which is discussed in the next section.

The difficulty of finding $\phi(n)$ depends on the difficulty of finding the factorization of n.

Example 9.7

What is the value of $\phi(13)$?

Solution

Because 13 is a prime, $\phi(13) = (13 - 1) = 12$.

Example 9.8

What is the value of $\phi(10)$?

Solution

We can use the third rule: $\phi(10) = \phi(2) \times \phi(5) = 1 \times 4 = 4$, because 2 and 5 are primes.

Example 9.9

What is the value of $\phi(240)$?

Solution

We can write $240 = 2^4 \times 3^1 \times 5^1$. Then

$$\phi(240) = (2^4 - 2^3) \times (3^1 - 3^0) \times (5^1 - 5^0) = 64$$

Example 9.10

Can we say that $\phi(49) = \phi(7) \times \phi(7) = 6 \times 6 = 36$?

Solution

No. The third rule applies when m and n are relatively prime. Here $49 = 7^2$. We need to use the fourth rule: $\phi(49) = 7^2 - 7^1 = 42$.

Example 9.11

What is the number of elements in $\mathbf{Z}_{14}^*$?

Solution

The answer is $\phi(14) = \phi(7) \times \phi(2) = 6 \times 1 = 6$. The members are 1, 3, 5, 9, 11, and 13.

Interesting point: If $n > 2$, the value of $\phi(n)$ is even.

Fermat's Little Theorem

Fermat's little theorem plays a very important role in number theory and cryptography. We introduce two versions of the theorem here.

First Version

The first version says that if p is a prime and a is an integer such that p does not divide a, then $a^{p-1} \equiv 1 \bmod p$.

Second Version

The second version removes the condition on a. It says that if p is a prime and a is an integer, then $a^p \equiv a \bmod p$.

Applications

Although we will see some applications of this theorem later in this chapter, the theorem is very useful for solving some problems.

Exponentiation Fermat's little theorem sometimes is helpful for quickly finding a solution to some exponentiations. The following examples show the idea.

Example 9.12

Find the result of $6^{10} \bmod 11$.

Solution

We have $6^{10} \bmod 11 = 1$. This is the first version of Fermat's little theorem where $p = 11$.

Example 9.13

Find the result of $3^{12} \bmod 11$.

Solution

Here the exponent (12) and the modulus (11) are not the same. With substitution this can be solved using Fermat's little theorem.

$$3^{12} \bmod 11 = (3^{11} \times 3) \bmod 11 = (3^{11} \bmod 11)\,(3 \bmod 11) = (3 \times 3) \bmod 11 = 9$$

Multiplicative Inverses A very interesting application of Fermat's theorem is in finding some multiplicative inverses quickly if the modulus is a prime. If p is a prime and a is an integer such that p does not divide a ($p \nmid a$), then $a^{-1} \bmod p = a^{p-2} \bmod p$.

This can be easily proved if we multiply both sides of the equality by a and use the first version of Fermat's little theorem:

$$a \times a^{-1} \bmod p = a \times a^{p-2} \bmod p = a^{p-1} \bmod p = 1 \bmod p$$

This application eliminates the use of extended Euclidean algorithm for finding some multiplicative inverses.

Example 9.14

The answers to multiplicative inverses modulo a prime can be found without using the extended Euclidean algorithm:

a. $8^{-1} \bmod 17 = 8^{17-2} \bmod 17 = 8^{15} \bmod 17 = 15 \bmod 17$
b. $5^{-1} \bmod 23 = 5^{23-2} \bmod 23 = 5^{21} \bmod 23 = 14 \bmod 23$
c. $60^{-1} \bmod 101 = 60^{101-2} \bmod 101 = 60^{99} \bmod 101 = 32 \bmod 101$
d. $22^{-1} \bmod 211 = 22^{211-2} \bmod 211 = 22^{209} \bmod 211 = 48 \bmod 211$

Euler's Theorem

Euler's theorem can be thought of as a generalization of Fermat's little theorem. The modulus in the Fermat theorem is a prime, the modulus in Euler's theorem is an integer. We introduce two versions of this theorem.

First Version

The first version of Euler's theorem is similar to the first version of the Fermat's little theorem. If a and n are coprime, then $a^{\phi(n)} \equiv 1 \pmod{n}$.

Second Version

The second version of Euler's theorem (as we call it for the lack of anyname) is similar to the second version of Fermat's little theorem; it removes the condition that a and n should be coprime. If $n = p \times q$, $a < n$, and k an integer, then $a^{k \times \phi(n) + 1} \equiv a \pmod{n}$.

Let us give an informal proof of the second version based on the first version. Because $a < n$, three cases are possible:

1. If a is neither a multiple of p nor a multiple of q, then a and n are coprimes.

$$a^{k \times \phi(n) + 1} \bmod n = (a^{\phi(n)})^k \times a \bmod n = (1)^k \times a \bmod n = a \bmod n$$

2. If a is a multiple of p ($a = i \times p$), but not a multiple of q,

$$a^{\phi(n)} \bmod q = (a^{\phi(q)} \bmod q)^{\phi(p)} \bmod q = 1 \quad \rightarrow \quad a^{\phi(n)} \bmod q = 1$$

$$a^{k \times \phi(n)} \bmod q = (a^{\phi(n)} \bmod q)^k \bmod q = 1 \quad \rightarrow \quad a^{k \times \phi(n)} \bmod q = 1$$

$$a^{k \times \phi(n)} \bmod q = 1 \quad \rightarrow \quad a^{k \times \phi(n)} = 1 + j \times q \quad \text{(Interpretation of congruence)}$$

$$a^{k \times \phi(n) + 1} = a \times (1 + j \times q) = a + j \times q \times a = a + (i \times j) \times q \times p = a + (i \times j) \times n$$

$$a^{k \times \phi(n) + 1} = a + (i \times j) \times n \quad \rightarrow \quad a^{k \times \phi(n) + 1} = a \bmod n \quad \text{(Congruence relation)}$$

3. If a is a multiple of q ($a = i \times q$), but not a multiple of p, the proof is the same as for the second case, but the roles of p and q are changed.

The second version of Euler's theorem is used in the RSA cryptosystem in Chapter 10.

Applications

Although we will see some applications of Euler's later in this chapter, the theorem is very useful for solving some problems.

Exponentiation Euler's theorem sometimes is helpful for quickly finding a solution to some exponentiations. The following examples show the idea.

Example 9.15

Find the result of $6^{24} \bmod 35$.

Solution
We have $6^{24} \bmod 35 = 6^{\phi(35)} \bmod 35 = 1$.

Example 9.16

Find the result of $20^{62} \bmod 77$.

Solution
If we let $k = 1$ on the second version, we have $20^{62} \bmod 77 = (20 \bmod 77)\ (20^{\phi(77)+1} \bmod 77) \bmod 77 = (20)(20) \bmod 77 = 15$.

Multiplicative Inverses Euler's theorem can be used to find multiplicative inverses modulo a prime; Euler's theorem can be used to find multiplicative inverses modulo a composite. If n and a are coprime, then $a^{-1} \bmod n = a^{\phi(n)-1} \bmod n$.

This can be easily proved if we multiply both sides of the equality by a:

$$a \times a^{-1} \bmod n = a \times a^{\phi(n)-1} \bmod n = a^{\phi(n)} \bmod n = 1 \bmod n$$

Example 9.17

The answers to multiplicative inverses modulo a composite can be found without using the extended Euclidean algorithm if we know the factorization of the composite:

a. $8^{-1} \bmod 77 = 8^{\phi(77)-1} \bmod 77 = 8^{59} \bmod 77 = 29 \bmod 77$
b. $7^{-1} \bmod 15 = 7^{\phi(15)-1} \bmod 15 = 7^{7} \bmod 15 = 13 \bmod 15$
c. $60^{-1} \bmod 187 = 60^{\phi(187)-1} \bmod 187 = 60^{159} \bmod 187 = 53 \bmod 187$
d. $71^{-1} \bmod 100 = 71^{\phi(100)-1} \bmod 100 = 71^{39} \bmod 100 = 31 \bmod 100$

Generating Primes

Two mathematicians, Mersenne and Fermat, attempted to develop a formula that could generate primes.

Mersenne Primes

Mersenne defined the following formula, called the **Mersenne numbers,** that was supposed to enumerate all primes.

$$\mathbf{M_p = 2^p - 1}$$

If p in the above formula is a prime, then M_p was thought to be a prime. Years later, it was proven that not all numbers created by the Mersenne formula are primes. The following lists some Mersenne numbers.

$M_2 = 2^2 - 1 = 3$
$M_3 = 2^3 - 1 = 7$
$M_5 = 2^5 - 1 = 31$
$M_7 = 2^7 - 1 = 127$
$M_{11} = 2^{11} - 1 = 2047$ **Not a prime ($2047 = 23 \times 89$)**
$M_{13} = 2^{13} - 1 = 8191$
$M_{17} = 2^{17} - 1 = 131071$

It turned out that M_{11} is not a prime. However, 41 Mersenne primes have been found; the latest one is $M_{124036583}$, a very large number with 7,253,733 digits. The search continues.

A number in the form $M_p = 2^p - 1$ is called a Mersenne number and may or may not be a prime.

Fermat Primes

Fermat tried to find a formula to generate primes. The following is the formula for a **Fermat number:**

$$\mathbf{F_n = 2^{2^n} + 1}$$

Fermat tested numbers up to F_4, but it turned out that F_5 is not a prime. No number

$F_0 = 3$
$F_1 = 5$
$F_2 = 17$
$F_3 = 257$
$F_4 = 65537$
$F_5 = 4294967297 = 641 \times 6700417$ **Not a prime**

greater than F_4 has been proven to be a prime. As a matter of fact many numbers up to F_{24} have been proven to be composite numbers.

9.2 PRIMALITY TESTING

If schemes for generating primes, like Fermat's or Mersenne's, have failed to produce large primes, how can we create large primes for cryptography? We could just choose a large random number and test it to be sure that it is a prime.

Finding an algorithm to correctly and efficiently test a very large integer and output *a prime* or *a composite* has always been a challenge in number theory, and consequently in cryptography. However, recent developments (one of which we discuss in this section) look very promising.

Algorithms that deal with this issue can be divided into two broad categories: **deterministic algorithms** and **probabilistic algorithms.** Some members of both categories are discussed here. A deterministic algorithm always gives a correct answer; a probabilistic algorithm gives an answer that is correct most of the time, but not all of the time. Although a deterministic algorithm is ideal, it is normally less efficient than the corresponding probabilistic one.

Deterministic Algorithms

A deterministic primality testing algorithm accepts an integer and always outputs *a prime* or *a composite.* Until recently, all deterministic algorithms were so inefficient at finding larger primes that they were considered infeasible. As we will show shortly, a newer algorithm looks more promising.

Divisibility Algorithm

The most elementary deterministic test for primality is the **divisibility test.** We use as divisors all numbers smaller that $\sqrt{n}$. If any of these numbers divides n, then n is composite. Algorithm 9.1 shows the divisibility test in its primitive, very inefficient form.

The algorithm can be improved by testing only odd numbers. It can be further improved by using a table of primes between 2 and $\sqrt{n}$. The number of arithmatic operations in Algorithm 9.1 is $\sqrt{n}$. If we assume that each arithmatic operation uses only one bit operation (unrealistic) then the bit-operation complexity of Algorithm 9.1 is $f(n_b) = \sqrt{2^{n_b}} = 2^{n_b/2}$, where n_b is the number of bits in n. In Big-O notation, the complexity can be shown as $O(2^{n_b})$: *exponential* (see Appendix L). In other words, the divisibility algorithm is infeasible (intractable) if n_b is large.

The bit-operation complexity of the divisibility test is exponential.

Example 9.18

Assume n has 200 bits. What is the number of bit operations needed to run the divisibility-test algorithm?

Solution

The bit-operation complexity of this algorithm is $2^{n_b/2}$. This means that the algorithm needs 2^{100} bit operations. On a computer capable of doing 2^{30} bit operations per second, the algorithm needs 2^{70} seconds to do the testing (forever).

Algorithm 9.1 *Pseudocode for the divisibility test*

```
Divisibility_Test (n)                    // n is the number to test for primality
{
  r ← 2
  while (r < √n )
  {
   if (r | n) return "a composite"
   r ← r + 1
  }
  return "a prime"
}
```

AKS Algorithm

In 2002, Agrawal, Kayal, and Saxena announced that they had found an algorithm for primality testing with polynomial bit-operation time complexity of $\boldsymbol{O}((\log_2 n_b)^{12})$. The algorithm uses the fact that $(x - a)^p \equiv (x^p - a) \bmod p$. It is not surprising to see some future refinements make this algorithm the standard primality test in mathematics and computer science.

Example 9.19

Assume n has 200 bits. What is the number of bit operations needed to run the AKS algorithm?

Solution

The bit-operation complexity of this algorithm is $\boldsymbol{O}((\log_2 n_b)^{12})$. This means that the algorithm needs only $(\log_2 200)^{12} = 39{,}547{,}615{,}483$ bit operations. On a computer capable of doing 1 billion bit operations per second, the algorithm needs only 40 seconds.

Probabilistic Algorithms

Before the AKS algorithm, all efficient methods for primality testing have been probabilistic. These methods may be used for a while until the AKS is formally accepted as the standard. A probabilistic algorithm does not guarantee the correctness of the result. However, we can make the probability of error so small that it is almost certain that the algorithm has returned a correct answer. The bit-operation complexity of the algorithm can become polynomial if we allow a small chance for mistakes. A probabilistic algorithm in this category returns either *a prime* or *a composite* based on the following rules:

a. If the integer to be tested is actually a prime, the algorithm definitely returns *a prime*.
b. If the integer to be tested is actually a composite, it returns *a composite* with probability $1 - \varepsilon$, but it may return *a prime* with the probability ε.

The probability of mistake can be improved if we run the algorithm more than once with different parameters or using different methods. If we run the algorithm m times, the probability of error may reduce to ε^m.

Fermat Test

The first probabilistic method we discuss is the **Fermat primality test.** Recall the *Fermat little theorem*

If n is a prime, then $a^{n-1} \equiv 1 \bmod n$.

Note that this means that if n is a prime, the congruence holds. It does not mean that if the congruence holds, n is a prime. The integer can be a prime or composite. We can define the following as the Fermat test

If n is a prime, $a^{n-1} \equiv 1 \bmod n$
If n is a composite, it is possible that $a^{n-1} \equiv 1 \bmod n$

A prime passes the Fermat test; a composite may pass the Fermat test with probability ε. The bit-operation complexity of Fermat test is the same as the complexity of an algorithm that calculates exponentiation. Later in this chapter, we introduce an algorithm for fast exponentiation with bit-operation complexity of $\boldsymbol{O}(n_b)$, where n_b is the number of bits in n. The probability can be improved by testing with several bases (a_1, a_2, a_3, and so on). Each test increases the probability that the number is a prime.

Example 9.20

Does the number 561 pass the Fermat test?

Solution
Use base 2

$$2^{561-1} = 1 \bmod 561$$

The number passes the Fermat test, but it is not a prime, because $561 = 33 \times 17$.

Square Root Test

In modular arithmetic, if n is a prime, the square root of 1 is either +1 or −1. If n is composite, the square root is +1 or −1, but there may be other roots. This is known as the **square root primality test.** Note that in modular arithmetic, −1 means $n - 1$.

If n is a prime, $\sqrt{1} \bmod n = \pm 1$.
If n is a composite, $\sqrt{1} \bmod n = \pm 1$ and possibly other values.

Example 9.21

What are the square roots of 1 mod n if n is 7 (a prime)?

Solution
The only square roots are 1 and −1. We can see that

$1^2 = 1 \bmod 7$ $\quad$ $(-1)^2 = 1 \bmod 7$
$2^2 = 4 \bmod 7$ $\quad$ $(-2)^2 = 4 \bmod 7$
$3^2 = 2 \bmod 7$ $\quad$ $(-3)^2 = 2 \bmod 7$

Note that we don't have to test 4, 5 and 6 because 4 = –3 mod 7, 5 = –2 mod 7 and 6 = –1 mod 7.

Example 9.22

What are the square roots of 1 mod n if n is 8 (a composite)?

Solution
There are four solutions: 1, 3, 5, and 7 (which is −1). We can see that

$1^2 = 1 \bmod 8$ $\quad$ $(-1)^2 = 1 \bmod 8$
$3^2 = 1 \bmod 8$ $\quad$ $5^2 = 1 \bmod 8$

Example 9.23

What are the square roots of 1 mod n if n is 17 (a prime)?

Solution
There are only two solutions: 1 and −1

$1^2 = 1 \bmod 17$ $\quad$ $(-1)^2 = 1 \bmod 17$
$2^2 = 4 \bmod 17$ $\quad$ $(-2)^2 = 4 \bmod 17$
$3^2 = 9 \bmod 17$ $\quad$ $(-3)^2 = 9 \bmod 17$
$4^2 = 16 \bmod 17$ $\quad$ $(-4)^2 = 16 \bmod 17$
$5^2 = 8 \bmod 17$ $\quad$ $(-5)^2 = 8 \bmod 17$
$6^2 = 2 \bmod 17$ $\quad$ $(-6)^2 = 2 \bmod 17$
$(7)^2 = 15 \bmod 17$ $\quad$ $(-7)^2 = 15 \bmod 17$
$(8)^2 = 13 \bmod 17$ $\quad$ $(-8)^2 = 13 \bmod 17$

Note that there is no need to check integers larger than 8 because 9 = –8 mod 17, and so on.

Example 9.24

What are the square roots of 1 mod n if n is 22 (a composite)?

Solution
Surprisingly, there are only two solutions, +1 and −1, although 22 is a composite.

$1^2 = 1 \bmod 22$
$(-1)^2 = 1 \bmod 22$

Although this test can tell us if a number is composite, it is difficult to do the testing. Given a number n, all numbers less than n (except 1 and n −1) must be squared to be sure that none of them is 1. This test can be used for a number (not +1 or −1) that when squared in modulus n has the value 1. This fact helps in the Miller-Rabin test in the next section.

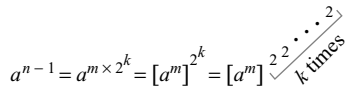
Miller-Rabin Test

The **Miller-Rabin primality test** combines the *Fermat test* and the *square root test* in a very elegant way to find a **strong pseudoprime** (a prime with a very high probability). In this test, we write $n - 1$ as the product of an odd number m and a power of 2:

$$n - 1 = m \times 2^k$$

The Fermat test in base a can be written as shown in Figure 9.2.

Figure 9.2 *Idea behind Fermat primality test*



In other words, instead of calculating $a^{n-1} \pmod{n}$ in one step, we can do it in $k + 1$ steps. What is the benefit of using $k + 1$ steps instead of just one? The benefit is that, in each step, the square root test can be performed. If the square root test fails, we stop and declare n a composite number. In each step, we assure ourself that the Fermat test is passed and the square root test is satisfied between all pairs of adjacent steps, if applicable (if the result is 1).

Initialization:

Choose a base a and calculate $T = a^m$, in which $m = (n - 1) / 2^k$

a. If T is +1 or −1, declare that n is a strong pseudoprime and stop. We say that n has passed two tests, the Fermat test and the square root test. Why? Because if T is ±1, T will become 1 in the next step and remains 1 until it passes the *Fermat test*. In addition T has passed the *square root test,* because T would be 1 in the next step and the square root of 1 (in the next step) is ±1 (in this step).
b. If T is anything else, we are not sure if n is a prime or a composite, so we continue to the next step.

Step 1:

We square T.

a. If the result is +1, we definitely know that the Fermat test will be passed, because T remains 1 for the succeeding tests. The square root test, however, has not been passed. Because T is 1 in this step and was something other than ±1 in the previous step (the reason why we did not stop in the previous step), we declare n a composite and stop.
b. If the result is −1, we know that n will eventually pass the Fermat test. We also know that it will pass the square root test because T is −1 in this step and becomes 1 in the next step. We declare n a strong peseudoprime and stop.
c. If T is anything else, we are not sure whether we do or do not have a prime. We continue to the next step.

Steps 2 to k − 1:

This step and all steps until step $k - 1$ are the same as step 1.

Step k:

This step is not needed. If we have reached this step and we have not made a decision, this step will not help us. If the result of this step is 1, the Fermat test is passed, but because the result of the previous step is not ±1, the square root test is not passed. After step $k - 1$, if we have not already stopped, we declare that n is composite.

The Miller-Rabin test needs from step 0 to step $k - 1$.

Algorithm 9.2 shows the pseudocode for the Miller-Rabin test.

Algorithm 9.2 *Pseudocode for Miller-Rabin test*

```
Miller_Rabin_Test (n, a)                    // n is the number; a is the base.
{
    Find m and k such that n − 1 = m × 2^k
    T ← a^m mod n
    if (T = ± 1)  return "a prime"
    for (i ← 1 to k − 1)                    // k − 1 is the maximum number of steps.
    {
        T ← T^2 mod n
        if (T = +1) return "a composite"
        if (T = −1) return "a prime"
    }
    return "a composite"
}
```

There exists a proof that each time a number passes a Miller-Rabin test, the probability that it is not a prime is 1/4. If the number passes m tests (with m different bases), the probability that it is not a prime is $(1/4)^m$.

Example 9.25

Does the number 561 pass the Miller-Rabin test?

Solution

Using base 2, let $561 - 1 = 35 \times 2^4$, which means $m = 35$, $k = 4$, and $a = 2$

Initialization:	$T = 2^{35} \bmod 561 = 263 \bmod 561$	
$k = 1$:	$T = 263^2 \bmod 561 = 166 \bmod 561$	
$k = 2$:	$T = 166^2 \bmod 561 = 67 \bmod 561$	
$k = 3$:	$T = 67^2 \bmod 561 = +1 \bmod 561$	$\rightarrow$ **a composite**

Example 9.26

We already know that 27 is not a prime. Let us apply the Miller-Rabin test.

Solution
With base 2, let $27 - 1 = 13 \times 2^1$, which means that $m = 13$, $k = 1$, and $a = 2$. In this case, because $k - 1 = 0$, we should do only the initialization step: $T = 2^{13} \bmod 27 = 11 \bmod 27$. However, because the algorithm never enters the loop, it returns *a composite.*

Example 9.27

We know that 61 is a prime, let us see if it passes the Miller-Rabin test.

Solution
We use base 2.

$61 - 1 = 15 \times 2^2 \rightarrow m = 15 \quad k = 2 \quad a = 2$
Initialization: $T = 2^{15} \bmod 61 = 11 \bmod 61$
$k = 1 \quad T = 11^2 \bmod 61 = -1 \bmod 61 \rightarrow$ **a prime**

Note that the last result is 60 mod 61, but we know that $60 = -1$ in mod 61.

Recommended Primality Test

Today, one of the most popular primality test is a combination of the divisibility test and the Miller-Rabin test. Following are the recommended steps:

1. Choose an odd integer, because all even integers (except 2) are definitely composites.
2. Do some trivial divisibility tests on some known primes such as 3, 5, 7, 11, 13, and so on to be sure that you are not dealing with an obvious composite. If the number passes all of these tests, move to the next step. If the number fails any of these tests, go back to step 1 and choose another odd number.
3. Choose a set of bases for testing. A large set of bases is preferable.
4. Do Miller-Rabin tests on each of the bases. If any of them fails, go back to step 1 and choose another odd number. If the test passes for all bases, declare the number a strong pseudoprime.

Example 9.28

The number 4033 is a composite (37×109). Does it pass the recommended primality test?

Solution

1. Perform the divisibility tests first. The numbers 2, 3, 5, 7, 11, 17, and 23 are not divisors of 4033.
2. Perform the Miller-Rabin test with a base of 2, $4033 - 1 = 63 \times 2^6$, which means m is 63 and k is 6.

Initialization: $T \equiv 2^{63} \pmod{4033} \equiv 3521 \pmod{4033}$
$k = 1 \quad T \equiv T^2 \equiv 3521^2 \pmod{4033} \equiv -1 \pmod{4033} \rightarrow$ **Passes**

3. But we are not satisfied. We continue with another base, 3.

Initialization: $T \equiv 3^{63} \pmod{4033} \equiv 3551 \pmod{4033}$

$k = 1$ $T \equiv T^2 \equiv 3551^2 \pmod{4033} \equiv 2443 \pmod{4033}$
$k = 2$ $T \equiv T^2 \equiv 2443^2 \pmod{4033} \equiv 3442 \pmod{4033}$
$k = 3$ $T \equiv T^2 \equiv 3442^2 \pmod{4033} \equiv 2443 \pmod{4033}$
$k = 4$ $T \equiv T^2 \equiv 2443^2 \pmod{4033} \equiv 3442 \pmod{4033}$
$k = 5$ $T \equiv T^2 \equiv 3442^2 \pmod{4033} \equiv 2443 \pmod{4033}$ **→Failed (composite)**

9.3 FACTORIZATION

Factorization has been the subject of continuous research in the past; such research is likely to continue in the future. Factorization plays a very important role in the security of several public-key cryptosystems (see Chapter 10).

Fundamental Theorem of Arithmetic

According to the *Fundamental Theorem of Arithmetic,* any positive integer greater than one can be written uniquely in the following prime **factorization** form where p_1, p_2,…, p_k are primes and e_1, e_2, …, e_k are positive integers.

$$n = p_1^{e1} \times p_2^{e2} \times \cdots \times p_k^{ek}$$

There are immediate applications of factorization, such as the calculation of the greatest common divisor and the least common multiplier.

Greatest Common Divisor

Chapter 2 discussed the greatest common divisor of two numbers, gcd (a, b). Recall that the Euclidean algorithm gives this value, but this value can also be found if we know the factorization of a and b.

$$a = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_k^{a_k} \qquad b = p_1^{b_1} \times p_2^{b_2} \times \cdots \times p_k^{b_k}$$

$$\gcd(a, b) = p_1^{\min(a_1, b_1)} \times p_2^{\min(a_2, b_2)} \times \cdots \times p_k^{\min(a_k, b_k)}$$

Least Common Multiplier

The *least common multiplier, lcm* (a, b), is the smallest integer that is a multiple of both a and b. Using factorization, we also find lcm (a, b).

$$a = p_1^{a1} \times p_2^{a2} \times \cdots \times p_k^{ak} \qquad b = p_1^{b1} \times p_2^{b2} \times \cdots \times p_k^{bk}$$

$$\text{lcm}(a, b) = p_1^{\max(a_1, b_1)} \times p_2^{\max(a_2, b_2)} \times \cdots \times p_k^{\max(a_k, b_k)}$$