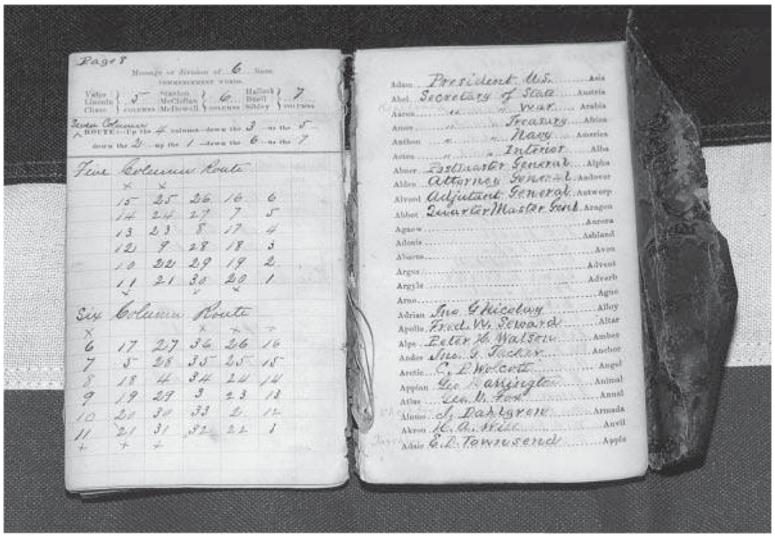
Chapter 24 Cryptography


Page 8

Message or division of 6 lines.

COMMENCEMENT WORDS.

Vales Lincoln Chase	5 columns	Stanton McClellan McDowell	6 columns	Halleck Buell Sibley	7 columns

Seven Columns ROUTE—Up the 4 column—down the 3—up the 5—down the 2—up the 1—down the 6—up the 7

Five Column Route

×	×			
15	25	26	16	6
14	24	27	7	5
13	23	8	17	4
12	9	28	18	3
10	22	29	19	2
11	21	30	20	1

Six Column Route

×			×	×	
6	17	27	36	26	16
7	5	28	35	25	15
8	18	4	34	24	14
9	19	29	3	23	13
10	20	30	33	2	12
11	21	31	32	22	1

Adam	President U.S.	Asia
Abel	Secretary of State	Austria
Aaron	" " War	Arabia
Amos	" " Treasury	Africa
Anthon	" " Navy	America
Aston	" " Interior	Alba
Abner	Postmaster General	Alpha
Alden	Attorney General	Andover
Alvord	Adjutant General	Antwerp
Abbot	Quarter Master Genl	Aragon
Agnew		Aurora
Adonis		Ashland
Alberta		Avon
Argus		Advent
Argyle		Adverb
Arno		Ague
Adrian	Jno G Nicolay	Alloy
Apollo	Fred. W. Seward	Altar
Alps	Peter H. Watson	Amber
Andes	Jno. G. Tucker	Anchor
Arctic	C. P. Wolcott	Angel
Appian	Geo [illegible]arrington	Animal
Atlas	Geo. V. Fox	Annal
Alonso	J. Dahlgren	Armada
Akron	H. A. Wise	Anvil
Adair	E. D. Townsend	Apple

Union code book used in the U.S. Civil War. U.S. government Image. NSA.

Contents

Computers today are used for a multitude of sensitive applications. Customers utilize electronic commerce to make purchases and pay their bills. Businesses use the Internet to share sensitive company documents and interact with business partners. Governments use computers to keep track of personal information about their citizens. Hospitals use databases to track patients and their billing information. And universities use networks of computers to store personal information about students and their grades. Each of us has an abundance of private and personal information that is being stored and transmitted by computers today.

Such sensitive information can be potentially damaging if it is altered, destroyed, or falls into the wrong hands. Thus, we should use powerful techniques to protect our sensitive data. In this chapter, we discuss several powerful algorithmic techniques for protecting sensitive information, so as to achieve the following goals:

- ***Data integrity***: Information should not be altered without detection. For example, it is important to prevent the modification of purchase orders or other contractually binding documents transmitted electronically.
- ***Authentication***: Individuals and organizations that are accessing or communicating sensitive information must be correctly identified, that is, authenticated. For example, corporations offering telecommuting arrangements to their employees should set up an authentication procedure for accessing corporate databases through the Internet.
- ***Authorization***: Agents that are performing computations involving sensitive information must be authorized to perform those computations.
- ***Nonrepudiation***: In transactions that imply a contract, the parties that have agreed to that contract must not have the ability of backing out of their obligations without being detected.
- ***Confidentiality***: Sensitive information should be kept secret from individuals who are not authorized to see that information. That is, we must ensure that data is viewed by the sender and by the receiver, but not by unauthorized parties who can eavesdrop on the communication. For example, many email messages are meant to be confidential.

Many of the techniques we discuss in this chapter for achieving the above goals utilize ***number theory***. Thus, we begin this chapter by discussing a number of important number theory concepts and algorithms. We describe the ancient, yet surprisingly efficient, Euclid's algorithm for computing greatest common divisors, as well as algorithms for computing modular exponents and inverses. We show how many of these number theory algorithms can be used in cryptographic algorithms that implement computer security services. We focus on encryption, including the popular public-key encryption schemes, RSA and El Gamal.

24.1 Greatest Common Divisors (GCD)

Many modern encryption schemes, like RSA and El Gamal, are based on viewing messages as numbers and using number-theoretic algorithms to process them. Thus, before we discuss some of these well-known encryption schemes, we should first discuss several fundamental algorithms for performing important computations involving numbers. Throughout this discussion, we assume that all variables are integers. Also, proofs of some mathematical facts are left as exercises.

24.1.1 Some Facts from Elementary Number Theory

To get us started, we need some facts from elementary number theory, including some notation and definitions. Given positive integers a and b, we use the notation

$$a|b$$

to indicate that a ***divides*** b, that is, b is a multiple of a. If $a|b$, then we know that there is some integer k, such that $b = ak$. The following properties of divisibility follow immediately from this definition.

Theorem 24.1: *Let a, b, and c be arbitrary integers. Then*

- *If $a|b$ and $b|c$, then $a|c$.*
- *If $a|b$ and $a|c$, then $a|(ib + jc)$, for all integers i and j.*
- *If $a|b$ and $b|a$, then $a = b$ or $a = -b$.*

Proof: See Exercise C-24.1. ■

Recall that an integer p is ***prime*** if $p \geq 2$ and its only divisors are the trivial divisors 1 and p. Thus, in the case that p is prime, $d|p$ implies $d = 1$ or $d = p$. An integer greater than 2 that is not prime is ***composite***. We also have the following:

Theorem 24.2 (*Fundamental Theorem of Arithmetic*): *Let $n > 1$ be an integer. Then there is a unique set of prime numbers $\{p_1, \ldots, p_k\}$ and positive integer exponents $\{e_1, \ldots, e_k\}$, such that*

$$n = p_1^{e_1} \cdots p_k^{e_k}.$$

The product $p_1^{e_1} \cdots p_k^{e_k}$ is known as the ***prime decomposition*** of n in this case. Theorem 24.2 and the notion of unique prime decomposition is the basis of several cryptographic schemes.

The Greatest Common Divisor (GCD)

The ***greatest common divisor*** of positive integers a and b, denoted $\gcd(a, b)$, is the largest integer that divides both a and b. Alternatively, we could say that $\gcd(a, b)$ is the number c, such that if $d|a$ and $d|b$, then $d|c$. If $\gcd(a, b) = 1$, we say that a and b are ***relatively prime***. We extend the notion of greatest common divisor to a pair of arbitrary integers by the following two rules:

- $\gcd(a, 0) = \gcd(0, a) = a$.
- $\gcd(a, b) = \gcd(|a|, |b|)$, which takes care of negative values.

Thus, $\gcd(12, 0) = 12$, $\gcd(10\,403, 303) = 101$, and $\gcd(-12, 78) = 6$.

Relating the Modulo Operator and the GCD

The following theorem gives an alternative characterization of the greatest common divisor. Its proof makes use of the modulo operator.

Theorem 24.3: *For any positive integers a and b, $\gcd(a, b)$ is the smallest positive integer d such that $d = ia + jb$ for some integers i and j. In other words, if d is the smallest positive integer linear combination of a and b, then $d = \gcd(a, b)$.*

Proof: Suppose d is the smallest integer such that $d = ia + jb$ for integers i and j. Note that, immediately from the definition of d, any common divisor of both a and b is also a divisor of d. Thus, $d \geq \gcd(a, b)$. To complete the proof, we need to show that $d \leq \gcd(a, b)$.

Let $h = \lfloor a/d \rfloor$. That is, h is the integer such that $a \bmod d = a - hd$. Then

$$\begin{aligned} a \bmod d &= a - hd \\ &= a - h(ia + jb) \\ &= (1 - hi)a + (-hj)b. \end{aligned}$$

In other words, $a \bmod d$ is also an integer linear combination of a and b. Moreover, by the definition of the modulo operator, $a \bmod d < d$. But d is the smallest positive integer linear combination of a and b. Thus, we must conclude that $a \bmod d = 0$, which implies that $d|a$. In addition, by a similar argument, we get that $d|b$. Thus, d is a divisor of both a and b, which implies $d \leq \gcd(a, b)$. ■

As we will show in Section 24.2, this theorem shows that the gcd function is useful for computing multiplicative modular inverses. In the next subsection, we show how to quickly compute the gcd function.

24.1.2 Euclid's GCD Algorithm

To compute the greatest common divisor of two numbers, we can use one of the oldest algorithms known, Euclid's algorithm. This algorithm is based on the following property of $\gcd(a, b)$:

Lemma 24.4: *Let a and b be two positive integers. For any integer r, we have*

$$\gcd(a, b) = \gcd(b, a - rb).$$

Proof: Let $d = \gcd(a, b)$ and $c = \gcd(b, a - rb)$. That is, d is the largest integer such that $d|a$ and $d|b$, and c is the largest integer such that $c|b$ and $c|(a - rb)$. We want to prove that $d = c$. By the definition of d, the number

$$(a - rb)/d = a/d - r(b/d)$$

is an integer. Thus, d divides both a and $a - rb$; hence, $d \leq c$.

By the definition of c, $k = b/c$ must be an integer, since $c|b$. Moreover,

$$(a - rb)/c = a/c - rk$$

must also be an integer, since $c|(a - rb)$. Thus, a/c must also be an integer, that is, $c|a$. Therefore, c divides both a and b; hence, $c \leq d$. We conclude then that $d = c$. ■

Lemma 24.4 leads us easily to an ancient algorithm, known as Euclid's algorithm, for computing the greatest common divisor (GCD) of two numbers, shown next in Algorithm 24.1.

```
Algorithm EuclidGCD($a, b$):
   Input: Nonnegative integers $a$ and $b$
   Output: $\gcd(a, b)$
  if $b = 0$ then
      return $a$
  return EuclidGCD($b, a \bmod b$)
```

Algorithm 24.1: Euclid's GCD algorithm.

An example of the execution of Euclid's algorithm is shown in Table 24.2.

	1	2	3	4	5	6	7
a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0

Table 24.2: Example of an execution of Euclid's algorithm to compute $\gcd(412, 260) = 4$. The arguments a and b of each recursive invocation of method EuclidGCD$(412, 260)$ are shown left-to-right, with the column headings showing the level of recursion in the EuclidGCD method.

Analyzing Euclid's Algorithm

Let us analyze the running time of Euclid's GCD algorithm. The number of arithmetic operations performed by method EuclidGCD(a, b) is proportional to the number of recursive calls. So to bound the number of arithmetic operations performed by Euclid's algorithm, we need only bound the number of recursive calls. First, we observe that after the first call, the first argument is always larger than the second one. For $i > 0$, let a_i be the first argument of the ith recursive call of method EuclidGCD. Clearly, the second argument of a recursive call is equal to a_{i+1}, the first argument of the next call. Also, we have

$$a_{i+2} = a_i \bmod a_{i+1},$$

which implies that the sequence of the a_i's is strictly decreasing. We will now show that the sequence decreases quickly. Specifically, we claim that

$$a_{i+2} < \frac{1}{2}a_i.$$

To prove the claim, we distinguish two cases:

Case 1: $a_{i+1} \leq \frac{1}{2}a_i$. Since the sequence of the a_i's is strictly decreasing, we have

$$a_{i+2} < a_{i+1} \leq \frac{1}{2}a_i.$$

Case 2: $a_{i+1} > \frac{1}{2}a_i$. In this case, since $a_{i+2} = a_i \bmod a_{i+1}$, we have

$$a_{i+2} = a_i \bmod a_{i+1} = a_i - a_{i+1} < \frac{1}{2}a_i.$$

Thus, the size of the first argument to the EuclidGCD method decreases by half with every other recursive call. We may therefore summarize the above analysis as follows.

Theorem 24.5: *Let a and b be two positive integers. Euclid's algorithm computes $\gcd(a, b)$ by executing $O(\log \max(a, b))$ arithmetic operations.*

We note that the complexity bound here is based on counting arithmetic operations, which themselves can be implemented to have reasonably-fast running times as a function of their input sizes, which are defined by the number of bits of the numbers involved. Moreover, note that the numbers, a and b, are represented using $O(\log \max(a, b))$ bits. So if we let n denote the input size for Euclid's algorithm, in bits, then we can characterize its performance as using $O(n)$ arithmetic operations.

24.2 Modular Arithmetic

Let Z_n denote the set of nonnegative integers less than n:

$$Z_n = \{0, 1, \cdots, (n-1)\}.$$

The set Z_n is also called the set of ***residues*** modulo n, because if $b = a \bmod n$, b is sometimes called the ***residue*** of a modulo n. Modular arithmetic in Z_n, where operations on the elements of Z_n are performed $\mathrm{mod} n$, exhibits properties similar to those of traditional arithmetic, such as the associativity, commutativity, distributivity of addition and multiplication, and the existence of identity elements 0 and 1 for addition and multiplication, respectively. Moreover, in any arithmetic expression, reducing each of its subexpressions modulo n produces the same result as computing the entire expression and then reducing that value modulo n. Also, every element x in Z_n has an ***additive inverse***, that is, for each $x \in Z_n$, there is a $y \in Z_n$ such that $x + y \bmod n = 0$. For example, the additive inverse of 5 modulo 11 is 6.

When it comes to multiplicative inverses, however, an important difference arises. Let x be an element of Z_n. A ***multiplicative inverse*** of x is an element $z^{-1} \in Z_n$ such that $xx^{-1} \equiv 1 \bmod n$. For example, the multiplicative inverse of 5 modulo 9 is 2, that is, $5^{-1} = 2$ in Z_9. As in standard arithmetic, 0 does not have a multiplicative inverse in Z_n. Interestingly, some nonzero elements also may not have a multiplicative inverse in Z_n. For example, 3 does not have a multiplicative inverse in Z_9. However, if n is prime, then every element $x \neq 0$ of Z_n has a multiplicative inverse in Z_n (1 is its own multiplicative inverse).

Theorem 24.6: *An element $x > 0$ of Z_n has a multiplicative inverse in Z_n if and only if $\gcd(x, n) = 1$ (that is, x and n have no common factors other than 1).*

Proof: Suppose that $\gcd(x, n) = 1$. By Theorem 24.3, there are integers i and j such that $ix + jn = 1$. This implies $ix \bmod n = 1$, that is, $i \bmod n$ is the multiplicative inverse of x in Z_n, which proves the "if" part of the theorem.

To prove the "only if" part, suppose, for a contradiction, that $x > 1$ divides n, and there is an element y such that $xy \equiv 1 \bmod n$. We have $xy = kn+1$, for some integer k. Thus, we have found integers $i = y$ and $j = -k$ such that $ix + jn = 1$. By Theorem 24.3, this implies that $\gcd(x, n) = 1$, a contradiction. ■

Recall that if $\gcd(x, n) = 1$, we say x and n are ***relatively prime*** (1 is relatively prime to all other integers). Thus, Theorem 24.6 implies that x has a multiplicative inverse in Z_n if and only if x is relatively prime to n. In addition, Theorem 24.6 implies that the sequence $0, x, 2x, 3x, \ldots, (n-1)x$ is simply a reordering of the elements of Z_n, that is, it is a permutation of the elements Z_n.

Example Multiplicative Inverses

In Table 24.3, we show the multiplicative inverses of the elements of Z_{11} as an example. When the multiplicative inverse x^{-1} of x exists in Z_n, the notation y/x in an expression taken modulo n means "$y\,x^{-1} \bmod n$."

x	0	1	2	3	4	5	6	7	8	9	10
$x^{-1} \bmod 11$		1	6	4	3	9	2	8	7	5	10

Table 24.3: Multiplicative inverses of the elements of Z_{11}.

In Table 24.4, we show the powers of the nonnull elements of Z_{11}. We observe the following interesting patterns:

- The last column of the table, with the values $x^{10} \bmod 11$ for $x = 1, \cdots, 10$, contains all ones, as given by Fermat's Little Theorem (19.5).
- In row 1, a subsequence of one element (1), is repeated ten times.
- In row 10, a subsequence of two elements, ending with 1, is repeated five times, since $10^2 \bmod 11 = 1$.
- In rows 3, 4, 5, and 9, a subsequence of five elements, ending with 1, is repeated twice.
- In each of the rows 2, 6, 7, and 8, the ten elements are all distinct.
- The lengths of the subsequences forming the rows of the table, and their number of repetitions, are the divisors of 10, that is, 1, 2, 5, and 10.

x	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^{10}
1	1	1	1	1	1	1	1	1	1
2	4	8	5	10	9	7	3	6	1
3	9	5	4	1	3	9	5	4	1
4	5	9	3	1	4	5	9	3	1
5	3	4	9	1	5	3	4	9	1
6	3	7	9	10	5	8	4	2	1
7	5	2	3	10	4	6	9	8	1
8	9	6	4	10	3	2	5	7	1
9	4	3	5	1	9	4	3	5	1
10	1	10	1	10	1	10	1	10	1

Table 24.4: Successive powers of the elements of Z_{11} modulo 11.

Euler's Totient Function

Euler's ***totient function*** of a positive integer n, denoted $\phi(n)$, is defined as the number of positive integers less than or equal to n that are relatively prime to n. That is, $\phi(n)$ is equal to the number of elements in Z_n that have multiplicative inverses in Z_n. If p is a prime, then

$$\phi(p) = p - 1.$$

Indeed, since p is prime, each of the numbers $1, 2, \ldots, p-1$ are relatively prime to it, and $\phi(p) = p - 1$.

What if n isn't a prime number? Suppose $n = pq$, where p and q are distinct primes. How many numbers are relatively prime to n? Well, initially, we observe that there are pq positive integers between 1 and n. However, q of them (including n) are multiples of p, and so they have a $\gcd$ of p with n. Similarly, there are p multiples of q (again, including n). Those multiples can't be counted in $\phi(n)$. Thus, we see that

$$\phi(n) = pq - q - (p-1) = (p-1)(q-1).$$

Euler's totient function is closely related to an important subset of Z_n known as the ***multiplicative group*** for Z_n, which is denoted as Z_n^*. The set Z_n^* is defined to be the set of integers between 1 and n that are relatively prime to n. If n is prime, then Z_n^* consists of the $n-1$ nonzero elements in Z_n, that is,

$$Z_n^* = \{1, 2, \ldots, n-1\},$$

if n is prime. In general, Z_n^* contains $\phi(n)$ elements.

The set Z_n^* possesses several interesting properties, with one of the most important being that this set is closed under multiplication modulo n. That is, for any pair of elements a and b of Z_n^*, we have that $c = ab \bmod n$ is also in Z_n^*. Indeed, by Theorem 24.6, a and b have multiplicative inverses in Z_n. To see that Z_n^* has this closure property, let $d = a^{-1}b^{-1} \bmod n$. Clearly, $cd \bmod n = 1$, which implies that d is the multiplicative inverse of c in Z_n. Thus, again applying Theorem 24.6, we have that c is relatively prime to n, that is, $c \in Z_n^*$. In algebraic terminology, we say that Z_n^* is a ***group***, which is a shorthand way of saying that each element in Z_n^* has a multiplicative inverse and multiplication in Z_n^* is associative, has an identity, and is closed in Z_n^*.

The fact that Z_n^* has $\phi(n)$ elements and is a multiplicative group naturally leads to an extension of Fermat's Little Theorem (19.5). Recall that in Fermat's Little Theorem, the exponent is $p - 1 = \phi(p)$, since p is prime. As it turns out, a generalized form of Fermat's Little Theorem is true, too. This generalized form is presented in the following, which is known as ***Euler's Theorem***.

Euler's Theorem

Theorem 24.7 (*Euler's Theorem*): *Let n be a positive integer, and let x be an integer such that* $\gcd(x, n) = 1$. *Then*

$$x^{\phi(n)} \equiv 1 \pmod{n}.$$

Proof: The proof technique is similar to that of Fermat's Little Theorem. Denote the elements of set Z_n^*, the multiplicative group for Z_n, as $u_1, u_2, \ldots, u_{\phi(n)}$. By the closure property of Z_n^*,

$$Z_n^* = \{xu_i : \ i = 1, \cdots, \phi(n)\},$$

that is, multiplying elements in Z_n^* by x modulo n merely permutes the sequence $u_1, u_2, \ldots, u_{\phi(n)}$. Thus, multiplying together the elements of Z_n^*, we obtain

$$(xu_1) \cdot (xu_2) \cdots (xu_{\phi(n)}) \equiv u_1\, u_2 \cdots u_{\phi(n)} \pmod{n}.$$

Again, we collect a term $x^{\phi(n)}$ on one side, giving us the congruence

$$x^{\phi(n)}(u_1\, u_2 \cdots u_{\phi(n)}) \equiv u_1\, u_2 \cdots u_{\phi(n)} \pmod{n}.$$

Dividing by the product of the u_i's, gives us $x^{\phi(n)} \equiv 1 \bmod n$. ■

Theorem 24.7 gives a closed-form expression for the multiplicative inverses. Namely, if x and n are relatively prime, we can write

$$x^{-1} \equiv x^{\phi(n)-1} \pmod{n}.$$

Generators

Given a prime p and an integer a between 1 and $p - 1$, the ***order*** of a is the smallest exponent $e > 1$ such that

$$a^e \equiv 1 \bmod p.$$

A ***generator*** (also called ***primitive root***) of Z_p is an element g of Z_p with order $p-1$. We use the term "generator" for such an element a, because the repeated exponentiation of a can generate all of Z_p^*. For example, as shown in Table 24.4, the generators of Z_{11} are 2, 6, 7, and 8. Generators play an important role in many computations, including several in image analysis and cryptography. The existence of generators is established by the following theorem, stated without proof.

Theorem 24.8: *If p is a prime, then set Z_p has $\phi(p-1)$ generators.*

24.2.1 Modular Exponentiation

Suppose we want to compute $30{,}192^{43{,}791} \bmod 65{,}301$. Multiplying $30{,}192$ by itself $43{,}791$ times and ***then*** taking the result modulo $65{,}301$ will yield unpredictable results in most programming languages due to arithmetic overflows. Thus, we should take the modulo after each multiplication, keeping the numbers in Z_n, where $n = 65{,}301$ is the modulus.

The Main Drawback with This Naive Approach

Unfortunately, although this "naive" exponentiation algorithm is correct, it is inefficient, for it requires $\Theta(p)$ multiplications and divisions, where p is the exponent. With large exponents, this running time would be quite slow, since it is exponential in the size of the input, which is the number of bits needed to represent the various input numbers. Fortunately, there is a better algorithm.

The Repeated Squaring Algorithm

A simple but important observation for an improved exponentiation algorithm is that squaring a number a^p is equivalent to multiplying its exponent p by two. In addition, multiplying two numbers a^p and a^q is equivalent to computing $a^{(p+q)}$. Based on these observations, we can evaluate $a^p \bmod n$ with the recursive computation, called the ***repeated squaring*** method, given in Algorithm 24.5.

```
Algorithm FastExponentiation($a, p, n$):
   Input: Integers $a$, $p$, and $n$
   Output: $r = a^p \bmod n$
  if $p = 0$ then
     return $1$
  if $p$ is even then
     $t \leftarrow$ FastExponentiation($a, p/2, n$) // $p$ is even, so $t = a^{p/2} \bmod n$
     return $t^2 \bmod n$
  $t \leftarrow$ FastExponentiation($a, (p-1)/2, n$) // $p$ is odd, so $t = a^{(p-1)/2} \bmod n$
  return $a(t^2 \bmod n) \bmod n$
```

Algorithm 24.5: Algorithm FastExponentiation for modular exponentiation using the repeated squaring method. Note that, since the modulo operator is applied after each arithmetic operation in method FastExponentiation, the size of the operands of each multiplication and modulo operation is never more than $2\lceil \log_2 n \rceil$ bits.

Analyzing the Repeated Squaring Algorithm

The main idea of this repeated squaring algorithm is to consider each bit of the exponent, p, in turn by dividing p by two until p goes to zero, squaring the current product Q_i for each such bit. In addition, if the current bit is a 1 (that is, p is odd), then we multiply in the base, a, as well.

To see why this algorithm works, define, for $i = 1, \ldots, b$,

$$Q_i = a^{q_i} \bmod n.$$

From the recursive definition of q_i, we derive the following definition of Q_i:

$$\begin{array}{lcll} Q_i & = & (Q_{i-1}^2 \bmod n) a^{p_{b-i}} \bmod n & \text{for} \quad 1 < i \le b \\ Q_1 & = & a^{p_{b-1}} \bmod n. & \end{array} \qquad (24.1)$$

It is easy to verify that $Q_b = a^p \bmod n$.

We show a sample execution of the repeated squaring algorithm for modular exponentiation in Table 24.6.

p	12	6	3	1	0
r	1	12	8	2	1

Table 24.6: Example of an execution of the repeated squaring algorithm for modular exponentiation. For each recursive invocation of FastExponentiation$(2, 12, 13)$, we show the second argument, p, and the output value $r = 2^p \bmod 13$.

The running time of the repeated squaring algorithm is easy to analyze. Referring to Algorithm 24.5, a constant number of arithmetic operations are performed, excluding those in the recursive call. Also, in each recursive call, the exponent p gets halved. Thus, the number of recursive calls and arithmetic operations is $O(\log p)$. We may therefore summarize as follows.

Theorem 24.9: *Let a p, and n be positive integers, with $a < n$. The repeated squaring algorithm computes $a^p \bmod n$ using $O(\log p)$ arithmetic operations.*

Since the number, p, is represented in binary using $O(\log p)$ bits, this theorem implies that the number of arithmetic operations used in the repeated squaring algorithm is linear in the input size of the exponent, p, in bits.

24.2.2 Modular Multiplicative Inverses

We turn now to the problem of computing multiplicative inverses in Z_n. First, we recall Theorem 24.6, which states that a nonnegative element x of Z_n admits an inverse if and only if $\gcd(x, n) = 1$. The proof of Theorem 24.6 actually suggests a way to compute $x^{-1} \bmod n$. Namely, we should find the integers i and j referred to by Theorem 24.3, such that

$$ix + jn = \gcd(x, n) = 1.$$

If we can find such integers i and j, we immediately obtain

$$i \equiv x^{-1} \bmod n.$$

The computation of the integers i and j referred to by Theorem 24.3 can be done with a variation of Euclid's algorithm, called ***extended Euclid's algorithm***.

Revisiting Euclid's GCD Algorithm

Let a and b be positive integers, and denote with d their greatest common divisor,

$$d = \gcd(a, b).$$

Let $q = a \bmod b$ and r be the integer such that $a = rb + q$, that is,

$$q = a - rb.$$

Euclid's algorithm is based on the repeated application of the formula

$$d = \gcd(a, b) = \gcd(b, q),$$

which immediately follows from Lemma 24.4.

Suppose that the recursive call of the algorithm, with arguments b and q, also returns integers k and l, such that

$$d = kb + lq.$$

Recalling the definition of r, we have

$$d = kb + lq = kb + l(a - rb) = la + (k - lr)b.$$

Thus, we have

$$d = ia + jb, \qquad \text{for } i = l \text{ and } j = k - lr.$$

This last equation suggests a method to compute the integers i and j, in addition to the GCD of a and b.

Extended Euclid's Algorithm

The resulting method, known as the extended Euclid's algorithm, is shown in Algorithm 24.7.

```
Algorithm ExtendedEuclidGCD($a, b$):
   Input: Nonnegative integers $a$ and $b$
   Output: Triplet of integers $(d, i, j)$ such that $d = \gcd(a, b) = ia + jb$
 if $b = 0$ then
     return $(a, 1, 0)$
 $q \leftarrow a \bmod b$
 Let $r$ be the integer such that $a = rb + q$
 $(d, k, l) \leftarrow$ ExtendedEuclidGCD($b, q$)
 return $(d, l, k - lr)$
```

Algorithm 24.7: Extended Euclid's algorithm.

We present, in Table 24.8, a sample execution of this algorithm. Its analysis is analogous to that of Euclid's algorithm.

a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0
r	1	1	1	2	2	5	
i	12	-7	5	-2	1	0	1
j	-19	12	-7	5	-2	1	0

Table 24.8: Execution of ExtendedEuclidGCD(a, b), for $a = 412$ and $b = 260$, to compute (d, i, j) such that $d = \gcd(a, b) = ia + jb$. For each recursive invocation, we show the arguments a and b, variable r, and output values i and j. The output value d is always $\gcd(412, 260) = 4$.

Theorem 24.10: *Let a and b be two positive integers. The extended Euclid's algorithm for computing a triplet of integers (d, i, j) such that*

$$d = \gcd(a, b) = ia + jb,$$

executes $O(\log \max(a, b))$ arithmetic operations.

Corollary 24.11: *Let x be an element of Z_n such that $\gcd(x, n) = 1$. The multiplicative inverse of x in Z_n can be computed with $O(\log n)$ arithmetic operations.*

24.3 Cryptographic Operations

The Internet enables a growing number of useful activities, such as email, shopping, and financial transactions, to be performed electronically. However, the Internet itself is an insecure transmission network: data transmitted over Wi-Fi and other communication media travels can be observed and potentially modified enroute to its destination. A variety of cryptographic techniques have therefore been developed to support secure communication over an insecure network such as the Internet. In particular, cryptography research has developed the following useful cryptographic computations:

- ***Encryption/decryption***: A message M to be transmitted, called the ***plaintext***, is transformed into an unrecognizable string of characters C, called the ***ciphertext***, before being sent over the network. This transformation is known as ***encryption***. After the ciphertext C is received, it is converted back to the plaintext M using an inverse transformation (that depends on additional secret information). This reverse transformation is called ***decryption***. An essential ingredient in encryption is that it should be computationally infeasible for an outsider to transform C back to M (without knowing the secret information possessed by the receiver).
- ***Digital signatures***: The author of a message M computes a message S that is derived from M and secret information known by the author. The message S is a ***digital signature*** if another party can easily verify that only the author of M could have computed S in a reasonable amount of time.

The two techniques lead to methods for supporting the information security services discussed in the introduction:

- ***Data integrity***: Computing a digital signature S of a message M not only helps us determine the author of M, it also verifies the integrity of M, for a modification to M would produce a different signature. So to perform a data integrity check we can perform a verification test that S is, in fact, a digital signature for the message M.
- ***Authentication***: The above cryptographic tools can be used for authentication in two possible ways. In ***password*** authentication schemes, a user will type a user ID and password in a client application, with this combination being immediately encrypted and sent to an authenticator. If the encrypted user ID and password combination matches that in a user database, then the individual is authenticated (and the database never stores passwords in plaintext). Alternatively, an authenticator can issue a challenge to a user in the form of a random message M that the user must immediately digitally sign for authentication.

- ***Authorization***: Given a scheme for authentication, we can issue authorizations by keeping lists, called ***access control lists***, that are associated with sensitive data or computations that should be accessed only by authorized individuals. Alternatively, the holder of a right to sensitive data or computations can digitally sign a message C that authorizes a user to perform certain tasks. For example, the message could be of the form, "I, U.S. Corporation vice president, give person x permission to access our fourth quarter earnings data."
- ***Confidentiality***: Sensitive information can be kept secret from nonauthorized agents by encrypting it.
- ***Nonrepudiation***: If we make the parties negotiating a contract, M, digitally sign that message, then we can have a way of proving that they have seen and agreed to the content of the message M.

Symmetric Encryption Schemes

As mentioned above, a fundamental problem in cryptography is confidentiality, that is, sending a message from Alice to Bob so that a third party, Eve, cannot gain any information from an intercepted copy of the message. Moreover, we have observed that confidentiality can be achieved by ***encryption schemes***, or ***ciphers***, where the message M to be transmitted, called the ***plaintext***, is ***encrypted*** into an unrecognizable string of characters C, called the ***ciphertext***, before being sent over the network. After the ciphertext C is received, it is decrypted back to the plaintext M using an inverse transformation called ***decryption***.

Secret Keys

In describing the details of an encryption scheme, we must explain all the steps needed in order to encrypt a plaintext M into a ciphertext C, and how to then decrypt that ciphertext back to M. Moreover, in order for Eve to be unable to extract M from C, there must be some secret information that is kept private from her.

In traditional cryptography, a common ***secret key*** k is shared by Alice and Bob, and is used to both encrypt and decrypt the message. Such schemes are also called ***symmetric encryption*** schemes, since k is used for both encryption and decryption and the same secret is shared by both Alice and Bob.

Substitution Ciphers

A classic example of a symmetric cipher is a ***substitution cipher***, where the secret key is a permutation π of the characters of the alphabet. Encrypting plaintext M into ciphertext C consists of replacing each character x of M with character $y = \pi(x)$. Decryption can be easily performed by knowing the permutation function π.

Indeed, M is derived from C by replacing each character y of C with character $x = \pi^{-1}(y)$. The ***Caesar cipher*** is an early example of a substitution cipher, where each character x is replaced by character

$$y = x + k \bmod n,$$

where n is the size of the alphabet and $1 < k < n$ is the secret key. This substitution scheme is known as the "Caesar cipher," for Julius Caesar is known to have used it with $k = 3$.

Substitution ciphers are quite easy to use, but they are not secure. Indeed, the secret key can be quickly inferred using ***frequency analysis***, based on the knowledge of the frequency of the various letters, or groups of consecutive letters in the text language.

Symmetric Cryptosystems

Secure and efficient symmetric ciphers do exist, and are often referred to by their acronyms, such as "3DES," "IDEA," and "AES." They perform a sequence of complex substitution and permutation transformations on the bits of the plaintext. While these systems are important in many applications, they are only mildly interesting from an algorithmic viewpoint; hence, they are out of the scope of this book. They run in time proportional to the length of the message being encrypted or decrypted. Thus, we mention that these algorithms exist and are fast, but in this book we do not discuss any of these efficient symmetric ciphers in any detail.

Public-Key Cryptosystems

A major problem with symmetric ciphers, however, is ***key transfer***, or how to distribute the secret key for encryption and decryption. In 1976, Diffie and Hellman described an abstract system that would avoid these problems, the ***public-key cryptosystem***. While they didn't actually publish a particular public-key system, they discussed the features of such a system. Specifically, given a message M, encryption function E, and decryption function D, the following four properties must hold:

1. $D(E(M)) = M$.
2. Both E and D are easy to compute.
3. It is computationally infeasible to derive D from E.
4. $E(D(M)) = M$.

In retrospect, these properties might seem like common sense, but they actually represent a significant innovation. The first property merely states that once a message has been encrypted, applying the decryption procedure will restore it. Property two is perhaps more obvious. In order for a cryptosystem to be practical, encryption and decryption must be computationally fast.

The third property is the start of the innovation. It means that E only goes one way; it is computationally infeasible to invert E, unless you already know D. Thus, the encryption procedure E can be made public. Any party can send a message, while only one knows how to decrypt it.

If the fourth property holds, then the mapping is one-to-one. Thus, the cryptosystem is a solution to the ***digital signature*** problem. Given an electronic message from Bob to Alice, how can we prove that Bob actually sent it? Bob can apply his decryption procedure to some signature message M. Any other party can then verify that Bob actually sent the message by applying the public encryption procedure E. Since only Bob knows the decryption function, only Bob can generate a signature message which can be correctly decoded by the function E.

One-Way Hash Functions

Public-key cryptosystems are often used in conjunction with a ***one-way hash function***, also called a ***message digest*** or ***fingerprint***. We provide an informal description of such a function next. A formal discussion is beyond the scope of this book.

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

1. Given a string M, the digest of M can be computed quickly.
2. Given the digest d of M, but not M, it is computationally infeasible to find M.

A one-way hash function is said to be ***collision-resistant*** if, given a string M, it is computationally infeasible to find another string M' with the same digest, and is said to be ***strongly collision-resistant*** if it is computationally infeasible to find two strings, M_1 and M_2, with the same digest.

Several functions believed to be strongly collision-resistant one-way hash functions have been devised. The ones currently advocated the most in practice are SHA-1, which produces a 160-bit digest, and SHA-256, which produces a 256-bit digest.

Using One-Way Hash Functions with Digital Signatures

One of the main applications of one-way hash functions is to speed up the construction of digital signatures. If we have a collision-resistant, one-way hash function, we can sign the digest of a message instead of the message itself, that is, the signature S is given by:

$$S = D(H(M)).$$

Except for small messages, hashing the message and signing the digest is faster, in practice, than signing the message directly.

24.4 The RSA Cryptosystem

Probably the most well-known public-key cryptosystem is tied to the difficulty of factoring large numbers. It is named ***RSA*** after its inventors, Rivest, Shamir, and Adleman. In this cryptosystem, we begin by selecting two large primes, p and q. Let $n = pq$ be their product and recall that $\phi(n) = (p-1)(q-1)$. Encryption and decryption keys e and d are selected so that

- e and $\phi(n)$ are relatively prime
- $ed \equiv 1 \pmod{\phi(n)}$.

The second condition means that d is the multiplicative inverse of e mod$\phi(n)$. The pair of values n and e form the public key, while d is the private key. In practice, e is chosen either randomly or as one of the following numbers: 3, 17, or 65 537.

The rules for encrypting and decrypting with RSA are simple. Let us assume, for simplicity, that the plaintext is an integer M, with $0 < M < n$. If M is a string, we can view it as an integer by concatenating the bits of its characters. The plaintext M is encrypted into ciphertext C with one modular exponentiation using the encryption key e as the exponent:

$$C \leftarrow M^e \bmod n \quad \text{(RSA encryption)}.$$

The decryption of ciphertext C is also performed with an exponentiation, using now the decryption key d as the exponent:

$$M \leftarrow C^d \bmod n \quad \text{(RSA decryption)}.$$

The correctness of the above encryption and decryption rules is justified by the following theorem.

Theorem 24.12: *Let p and q be two odd primes, and define $n = pq$. Let e be relatively prime with $\phi(n)$ and let d be the multiplicative inverse of e modulo $\phi(n)$. For each integer x such that $0 < x < n$,*

$$x^{ed} \equiv x \pmod{n}.$$

Proof: Let $y = x^{ed} \bmod n$. We want to prove that $y = x$. Because of the way we have selected e and d, we can write $ed = k\phi(n) + 1$, for some integer k. Thus, we have

$$y = x^{k\phi(n)+1} \bmod n.$$

We distinguish two cases.

Case 1: x does not divide n. We rewrite y as follows:

$$\begin{aligned} y &= x^{k\phi(n)+1} \bmod n \\ &= xx^{k\phi(n)} \bmod n \\ &= x(x^{\phi(n)} \bmod n)^k \bmod n. \end{aligned}$$

By Theorem 24.7 (Euler's theorem), we have $x^{\phi(n)} \bmod n = 1$, which implies $y = x \cdot 1^k \bmod n = x$.

Case 2: x divides n. Since $n = pq$, with p and q primes, x is a multiple of either p or q. Suppose x is a multiple of p, that is, $x = hp$ for some positive integer h. Clearly, x cannot be a multiple of q as well, since otherwise x would be greater than $n = pq$, a contradiction. Thus, $\gcd(x, q) = 1$ and by Theorem 24.7 (Euler's theorem), we have

$$x^{\phi(q)} \equiv 1 \pmod q.$$

Since $\phi(n) = \phi(p)\phi(q)$, raising both sides of the above congruence to the power of $k\phi(q)$, we obtain

$$x^{k\phi(n)} \equiv 1 \pmod q,$$

which we rewrite as

$$x^{k\phi(n)} = 1 + iq,$$

for some integer i. Multiplying both sides of the above equality by x, and recalling that $x = hp$ and $n = pq$, we obtain

$$\begin{aligned} x^{k\phi(n)+1} &= x + xiq \\ &= x + hpiq \\ &= x + (hi)n. \end{aligned}$$

Thus, we have

$$y = x^{k\phi(n)+1} \bmod n = x.$$

In either case, we have shown that $y = x$, which concludes the proof of the theorem. ■

The symmetry of the encryption and decryption functions implies that the RSA cryptosystem directly supports digital signatures. Indeed, a digital signature S for message M is obtained by applying the decryption function to M, that is,

$$S \leftarrow M^d \bmod n \quad \text{(RSA signature).}$$

The verification of the digital signature S is now performed with the encryption function, that is, by checking that

$$M \equiv S^e \pmod n \quad \text{(RSA verification).}$$

The Difficulty of Breaking RSA

Note that even if we know the value e, we cannot figure out d unless we know $\phi(n)$. Most cryptography researchers generally believe that breaking RSA requires that we compute $\phi(n)$ and that this requires factoring n. While there is no ***proof*** that factorization is computationally difficult, a whole series of famous mathematicians have worked on the problem over the past few hundred years. Especially if n is large ($\approx$ 200 digits), it will take a very long time to factor it. To give you an idea of the state of the art, mathematicians were quite excited when a nationwide network of computers was able to factor the ninth Fermat number, $2^{512} - 1$. This number has "only" 155 decimal digits. Barring a major breakthrough, the RSA system will remain secure. For if technology somehow advances to a point where it is feasible to factor 200-digit numbers, we need only choose an n with 300 or 400 digits.

Analysis and Setup for RSA Encryption

The running time of RSA encryption, decryption, signature, and verification is simple to analyze. Indeed, each such operation requires a constant number of modular exponentiations, which can be performed with method FastExponentiation (Algorithm 24.5).

Theorem 24.13: *Let n be the modulus used in the RSA cryptosystem. RSA encryption, decryption, signature, and verification each take $O(\log n)$ arithmetic operations.*

To set up the RSA cryptosystem, we need to generate the public and private key pair. Namely, we need to compute the private key (d, p, q) and the public key (e, n) that goes with it. This involves the following computations:

- Selection of two random primes p and q with a given number of bits. This can be accomplished by testing random integers for primality, as discussed at the end of Section 19.4.1.
- Selection of an integer e relatively prime to $\phi(n)$. This can be done by picking random primes less than $\phi(n)$ until we find one that does not divide $\phi(n)$. In practice, it is sufficient to check small primes from a list of known primes (often $e = 3$ or $e = 17$ will work).
- Computing the multiplicative inverse d of e in $Z_{\phi(n)}$. This can be done using the extended Euclid's algorithm (Corollary 24.11).

24.5 The El Gamal Cryptosystem

We have seen that the security of the RSA cryptosystem is related to the difficulty of factoring large numbers. It is possible to construct cryptosystems based on other difficult number-theoretic problems. We now consider the El Gamal cryptosystem, named after its inventor, Taher El Gamal, which is based on the difficulty of a problem called the "discrete logarithm."

The Discrete Logarithm

When we're working with the real numbers, $\log_b y$ is the value x, such that $b^x = y$. We can define an analogous discrete logarithm. Given integers b and n, with $b < n$, the ***discrete logarithm*** of an integer y to the base b is an integer x, such that

$$b^x \equiv y \bmod n.$$

The discrete logarithm is also called ***index***, and we write

$$x = \text{ind}_{b,n} y.$$

While it is quite efficient to raise numbers to large powers modulo p (recall the repeated squaring algorithm, Algorithm 24.5), the inverse computation of the discrete logarithm is much harder. The El Gamal system relies on the difficulty of this computation.

El Gamal Encryption

Let p be a prime, and g be a generator of Z_p^*. The private key x is an integer between 1 and $p-2$. Let $y = g^x \bmod p$. The public key for El Gamal encryption is the triplet (p, g, y). If taking discrete logarithms is as difficult as it is widely believed, releasing $y = g^x \bmod p$ does not reveal x.

To encrypt a plaintext M, a random integer k relatively prime to $p-1$ is selected, and the following pair of values is computed:

$$\begin{array}{c} a \leftarrow g^k \bmod p \\ b \leftarrow My^k \bmod p \end{array} \quad \text{(El Gamal encryption).}$$

The ciphertext C consists of the pair (a, b) computed above.

El Gamal Decryption

The decryption of the ciphertext $C = (a, b)$ in the El Gamal scheme, to retrieve the plaintext M, is simple:

$$M \leftarrow b/a^x \bmod p \quad \text{(El Gamal decryption).}$$

In the above expression, the "division" by a^x should be interpreted in the context of modular arithmetic, that is, b is multiplied by the inverse of a^x in Z_p. The correctness of the El Gamal encryption scheme is easy to verify. Indeed, we have

$$\begin{aligned} b/a^x \bmod p &= My^k(a^x)^{-1} \bmod p \\ &= Mg^{xk}(g^{kx})^{-1} \bmod p \\ &= M. \end{aligned}$$

Using El Gamal for Digital Signatures

A variation of the above scheme provides a digital signature. Namely, a signature for message M is a pair $S = (a, b)$ obtained by selecting a random integer k relatively prime to $p - 1$ (which, of course, equals $\phi(p)$) and computing

$$\left.\begin{aligned} a &\leftarrow g^k \bmod p \\ b &\leftarrow k^{-1}(M - xa) \bmod (p-1) \end{aligned}\right. \quad \text{(El Gamal signature).}$$

To verify a digital signature $S = (a, b)$, we check that

$$y^a a^b \equiv g^M \pmod p \quad \text{(El Gamal verification).}$$

The correctness of this digital signature scheme is based on the following:

$$\begin{aligned} y^a a^b \bmod p &= ((g^x \bmod p)^a \bmod p)((g^k \bmod p)^{k^{-1}(M-xa) \bmod (p-1)} \bmod p) \\ &= g^{xa} g^{kk^{-1}(M-xa) \bmod (p-1)} \bmod p \\ &= g^{xa+M-xa} \bmod p \\ &= g^M \bmod p. \end{aligned}$$

Analysis of El Gamal Encryption

The analysis of the performance of the El Gamal cryptosystem is similar to that of RSA. Namely, we have the following.

Theorem 24.14: *Let n be the modulus used in the El Gamal cryptosystem. El Gamal encryption, decryption, signature, and verification each take $O(\log n)$ arithmetic operations.*

24.6 Exercises

Reinforcement

R-24.1 Show the execution of method EuclidGCD$(14300, 5915)$ by constructing a table similar to Table 24.2.

R-24.2 Write a nonrecursive version of Algorithm EuclidGCD.

R-24.3 What is $9^{60} \bmod 77$?

R-24.4 Construct the multiplication table of the elements of Z_{11}, where the element in row i and column j ($0 \leq i, j \leq 10$) is given by $i \cdot j \bmod 11$.

R-24.5 Show the execution of method FastExponentiation$(5, 12, 13)$ by constructing a table similar to Table 24.6.

R-24.6 Write a nonrecursive version of Algorithm ExtendedEuclidGCD.

R-24.7 Extend Table 24.8 with two rows giving the values of ia and jb at each step of the algorithm and verify that $ia + jb = 1$.

R-24.8 Show the execution of method ExtendedEuclidGCD$(412, 113)$ by constructing a table similar to Table 24.8.

R-24.9 What are the multiplicative inverses of 113, 114, and 127 in Z_{299}.

R-24.10 Construct a table showing an example of the RSA cryptosystem with parameters $p = 17$, $q = 19$, and $e = 5$. The table should have two rows, one for the plaintext M and the other for the ciphertext C. The columns should correspond to integer values in the range $[10, 20]$ for M.

R-24.11 Show the result of an El Gamal encryption of the message $M = 8$ using $k = 4$ for the public key $(p, g, y) = (59, 2, 25)$.

Creativity

C-24.1 Prove Theorem 24.1.

C-24.2 Show the existence of additive inverses in Z_p, that is, prove that for each $x \in Z_p$, there is a $y \in Z_p$, such that $x + y \bmod p = 0$.

C-24.3 Let p be a prime. Give an efficient alternative algorithm for computing the multiplicative inverse of an element of Z_p that is not based on the extended Euclid's algorithm. What is the running time of your algorithm?

C-24.4 Give an alternative proof of Theorem 24.6 that does not use Theorem 24.3.

C-24.5 Show how to modify Algorithm ExtendedEuclidGCD to compute the multiplicative inverse of an element in Z_n using arithmetic operations on operands with at most $2\lceil \log_2 n \rceil$ bits.

C-24.6 Suppose Alice wants to send Bob a message, M, that is the price she is willing to pay for his old bike. Here, M is just an integer in binary. She uses the RSA algorithm to encrypt M, to produce the ciphertext, C, using Bob's public key, and sends it to Bob. Unfortunately, Eve has intercepted C before it gets to Bob Explain how Eve can use Bob's public key to alter the ciphertext C to change it into C', so that if she sends C' to Bob (with Eve pretending to be Alice), then, after Bob has decrypted C', he will get a plaintext that is twice the value of M.

C-24.7 Solve the previous exercise, but use the El Gamal cryptosystem instead of RSA.

C-24.8 Suppose the primes p and q used in the RSA cryptosystem, to define $n = pq$, are in the range $[\sqrt{n} - \log n,\ \sqrt{n} + \log n]$. Explain how you can efficiently factor n using this information.

C-24.9 Why can't you use the pair $(1, n)$ as an RSA public key, even if $n = pq$, for two large primes, p and q?

Applications

A-24.1 There is a perfectly secure cipher, known as the ***one-time pad***, which is said to have been used for encrypting messages on the "hot line" between Moscow and Washington, DC, during the Cold War. In this cryptosystem, Alice and Bob each share a random bit string, K, as large as any message they might wish to communicate. The string K is the secret key. To compute a ciphertext, C, from a message, M, Alice computes

$$C = M \oplus K,$$

where "$\oplus$" denotes the bitwise exclusive-or operator. Show that Bob can decrypt the ciphertext, C, by computing $C \oplus K$. Also, show that this scheme achieves perfect confidentiality, based on the facts that each bit of the output is independent, random, and every plaintext of length $|M|$ is a possible plaintext for the ciphertext, C.

A-24.2 There are instances when it is useful to prove that a document, D, exists on a certain date. In order to facilitate such proofs, Bob collects a group of documents, $D_1, D_2, \ldots, D_n$, every day from people wanting time stamps for their documents on that day. Bob constructs a complete binary tree, T, with n leaves, of height $\lceil \log n \rceil$, with each leaf, i, associated with a document, D_i. He stores at i the result, $h_i = H(D_i)$, of a one-way hash function computed on D_i. For each internal node, v, with children x and y, he stores $h_v = H(h_x || h_y)$, where $||$ denotes concatenation. Finally, he publishes the value, h_r, for the root, r, of T in a classified advertisement in a local newspaper. How can Bob give each document owner a set of $O(\log n)$ numbers so that together with the classified advertisement, each document owner can prove the existence of his or her document on the date the ad appeared, with the confidence of the security properties of the one-way hash function, H?

A-24.3 Consider the time stamping problem from the previous exercise, but now suppose that each day that there is one document added to the set, and one document that is removed from the set, but all of the remaining documents still need to be proven to exist as a part of the set on that day. Explain how Bob can update the tree, T, in $O(\log n)$ time, to reflect these changes?

A-24.4 Suppose Alice is a U.S. spy on a 7-day trip to a faraway land and wants to prove for each day she is gone that she not been captured. She has chosen a secret random number, x, which she is keeping secret. But she did tell her CIA handler the value $y = H(H(H(H(H(H(H(x)))))))$, where H is a one-way cryptographic hash function. Unfortunately, her enemy, Eve, was able to listen in on this message; hence, Eve also knows the value of y. Explain how Alice can send a single message every day that proves she has not been captured.

A-24.5 Digital certificates are signed documents, where a respected authority verifies the binding between a person's identity information (like their name, email address, etc.) and their public key. But if that person loses the private key that goes with his or her public key, then this certification needs to be revoked. To support this service, the respected authority can keep a dictionary, D, of revoked certificates. Explain how the authority can answer any request for the revocation status for any digital certificate in $O(\log n)$ time, where n is the size of D. Also, how can the person asking this query be able to prove to a third party that the result is valid?

A-24.6 One of the main uses for public-key cryptography is that it can be used to establish a secret key for a communication session between Alice and Bob even if they have never met to share that secret key in advance. Explain how public key cryptography can be used for this purpose.

A-24.7 Suppose Bob has an RSA public key, (e, n), and that he has promised Eve that she can send him any single message, $M < n$, and he will sign M using a simple RSA signature method to compute $S = M^d \bmod n$, where d is his private RSA exponent, sending S back to Eve. What Bob doesn't know is that Eve has previously captured a ciphertext C that Alice encrypted for Bob using her plaintext, P, and Bob's RSA public key, (e, n). Eve wants to trick Bob into decrypting C for her without revealing P. So Eve asks Bob to sign the message $M = r^e C \bmod n$, where r is a random number that Eve chose to be relatively prime to n. Explain how, even though she does not know the value of d, Eve can use Bob's signature, S, on M, to discover the plaintext, P, for C.

Chapter Notes

An introduction to number theory is provided in books by Koblitz [133] and Kranakis [136]. The classic textbook on numerical algorithms is the second volume of Knuth's series on ***The Art of Computer Programming*** [130]. Algorithms for number theoretic problems are also presented in the books by Bressoud and Wagon [38] and by Bach and Shallit [19].

The RSA cryptosystem is named after the initials of its inventors, Rivest, Shamir, and Adleman [178]. The El Gamal cryptosystem is named after its inventor [79]. Sections 24.1 and 24.3 are based in part on an unpublished manuscript of Achter and Tamassia [1].