ICS 2411 COMPUTER SECURITY AND CRYPTOGRAPHY

Name: MWANGI ALLAN VINCENT

Reg No. SCT211-0025/2021

CHAPTER 4: NUMBER THEORY AND CRYPTOGRAPHY SUMMARY

Divisibility and Modular Arithmetic

4.1.1 Introduction

This section introduces divisibility, which is fundamental in number theory and modular arithmetic. Modular arithmetic is crucial in various applications, including generating pseudorandom numbers, file allocation, check digits, and encryption.

4.1.2 Division

Dividing an integer by another integer results in a quotient and sometimes a remainder.

- **Definition:** An integer a divides b (denoted a|b) if there exists an integer [Equation] c such that b=ac. If no such c exists, we write a∤b.
- **Example:** 3∤7 since 7/3 is not an integer, but 3|12 since 12/3=4.
- **Counting Multiples:** The number of positive integers ≤ n that are divisible by d is ⌊n/d⌋.

Theorem 1: Properties of Divisibility

For integers a, b, a, b, a, b, and c (where $a \neq 0$):

1. If a|b and a|c, then a|(b+c).
2. If a|b, then a|bc for any integer c.
3. If a|b and b|c, then a|c.

Corollary 1

If $a|ba \mid mid\ b$a|b and $a|ca \mid mid\ c$a|c, then $a|(mb+nc)a \mid mid\ (mb + nc)$a|(mb+nc) for any integers mmm and n.

4.1.3 The Division Algorithm

For any integer a and positive integer d there exist unique integers q (quotient) and r (remainder) such that:

a=dq+r, 0≤r<d

This is called the **Division Algorithm**, though it is not an algorithm in the computational sense.

- **Notation:**
  → $q=a \div d$ (quotient)
  → $r=a \bmod d$ (remainder)

Examples:

- $101 \div 11=9$101 ÷11=9 remainder $22$2 → $101=11 \cdot 9+2$101 = 11 ⋅9+2, so $101 \bmod 11=2$101 mod11=2.
- $-11 \div 3=-4$−11 ÷3=−4 remainder $11$1 → $-11=3(-4)+1$-11 = 3(-4) + 1−11=3(−4)+1, so $-11 \bmod 3=1$−11 mod3=1.

Programming languages may define the ***mod*** *operator differently for negative numbers, so caution is needed when using them.*

4.1.4 Modular Arithmetic

Sometimes, we are only concerned with remainders when dividing numbers. This leads to the concept of **modular congruence**.

Definition 3: Congruence Modulo m

For integers a and b, and a positive integer mm:

$a \equiv b \pmod{m}$ *if and only if* m|(a−b)

This means that a and b have the same remainder when divided by m.

Theorem 2

a≡b(modm)⟺amodm=bmodm

Examples:

- 17≡5(mod6) since $17-5=12$17−5=12 is divisible by 6.
- 24≡14(mod6) since $24-14=10$24−14=10 is not divisible by 6.

Theorem 3

Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that a = b + km.

Proof: If a ≡ b (mod m), by the definition of congruence (Definition 3), we know that m | (a − b). This means that there is an integer k such that a − b = km, so that a = b + km.

Conversely, if there is an integer k such that a = b + km, then km = a − b. Hence, m divides a − b, so that a ≡ b (mod m).

Theorem 4

Let m be a positive integer. If a ≡ b (mod m) and c ≡ d (mod m), then a + c ≡ b + d (mod m) and ac ≡ bd (mod m).

Proof: We use a direct proof. Because a ≡ b (mod m) and c ≡ d (mod m), by Theorem 4 there are integers s and t with b = a + sm and d = c + tm. Hence, b + d = (a + sm) + (c + tm) = (a + c) + m(s + t) and bd = (a + sm)(c + tm) = ac + m(at + cs + stm). Hence, a + c ≡ b + d (mod m) and ac ≡ bd (mod m).

EXAMPLE

Because 7 ≡ 2 (mod 5) and 11 ≡ 1 (mod 5), it follows from Theorem 5 that

18 = 7 + 11 ≡ 2 + 1 = 3 (mod 5) and that 77 = 7 · 11 ≡ 2 · 1 = 2 (mod 5).

We must be careful working with congruences. Some properties we may expect to be true are not valid. For example, if ac ≡ bc (mod m), the congruence a ≡ b (mod m) may be false. Similarly, if a ≡ b (mod m) and c ≡ d (mod m), the congruence ac ≡ bd (mod m) may be false.

Corollary 2

Let m be a positive integer and let a and b be integers. Then (a + b) mod m = ((a mod m) + (b mod m)) mod m and ab mod m = ((a mod m)(b mod m)) mod m.

Proof: By the definitions of mod m and of congruence modulo m, we know that a ≡ (a mod m) (mod m) and b ≡ (b mod m) (mod m). Hence, Theorem 5 tells us that a + b ≡ (a mod m) + (b mod m) (mod m) and ab ≡ (a mod m)(b mod m) (mod m).

EXAMPLE

Find the value of (193 mod 31)4 mod 23.

Solution: To compute (193 mod 31)4 mod 23, we will first evaluate 193 mod 31. Because 193 = 6859 and 6859 = 221 · 31 + 8, we have 193 mod 31 = 6859 mod 31 = 8. So, (193 mod 31)4 mod 23 = 84 mod 23.

Next, note that 84 = 4096. Because 4096 = 178 · 23 + 2, we have 4096 mod 23 = 2. Hence, 3 (19 mod 31)4 mod 23 = 2.

4.1.5 Arithmetic Modulo m

We can define arithmetic operations on Z_m, the set of nonnegative integers less than m, that is, the set {0, 1, … , m − 1}. In particular, we define addition of these integers, denoted by $+_m$ by $a +_m b = (a + b) \bmod m$, where the addition on the right-hand side of this equation is the ordinary addition of integers, and we define multiplication of these integers, denoted by ·m by a ·m b = (a · b) mod m, where the multiplication on the right-hand side of this equation is the ordinary multiplication of integers. The operations $+_m$ and ·m are called addition and multiplication modulo m and when we use these operations, we are said to be doing ***arithmetic modulo*** m.

Use the definition of addition and multiplication in Zm to find 7 +11 9 and 7 ·11 9.

Solution: Using the definition of addition modulo 11, we find that

7 +11 9 = (7 + 9) mod 11 = 16 mod 11 = 5, and 7 ·11 9 = (7 · 9) mod 11 = 63 mod 11 = 8.

Hence, 7 +11 9 = 5 and 7 ·11 9 = 8.

The operations +m and ·m satisfy many of the same properties of ordinary addition and multiplication of integers. In particular, they satisfy these properties:

Closure If a and b belong to Zm , then a +m b and a ·m b belong to Zm .

Associativity If a, b, and c belong to Zm , then (a +m b) +m c = a +m (b +m c) and (a ·m b) ·m c = a ·m (b ·m c).

Commutativity If a and b belong to Zm , then a +m b = b +m a and a ·m b = b ·m a.

Identity elements The elements 0 and 1 are identity elements for addition and multiplication modulo m, respectively. That is, if a belongs to Zm , then a +m 0 = 0 +m a = a and a ·m 1 = 1 ·m a = a.

Additive inverses If a ≠ 0 belongs to Zm , then m − a is an additive inverse of a modulo m and 0 is its own additive inverse. That is, a +m (m − a) = 0 and 0 +m 0 = 0.

Distributivity If a, b, and c belong to Zm , then a ·m (b +m c) = (a ·m b) +m (a ·m c) and (a +m b) ·m c = (a ·m c) +m (b ·m c).

These properties follow from the properties we have developed for congruences and remainders modulo m, together with the properties of integers. Note that we have listed the property that every element of Z_m has an additive inverse, but no analogous property for multiplicative inverses has been included. This is because multiplicative inverses do not always exist modulo m. For instance, there is no multiplicative inverse of 2 modulo 6, as the reader can verify. We will return to the question of when an integer has a multiplicative inverse modulo m later in this chapter.

- ***Remark: Because Z_m with the operations of addition and multiplication modulo m satisfies the properties listed, Z_m with modular addition is said to be a commutative group and Z_m with both of these operations is said to be a commutative rin***g.

Integer Representations and Algorithms

1. Introduction

Integers can be represented using any base greater than one. While decimal (base 10) is commonly used, binary (base 2), octal (base 8), and hexadecimal (base 16) are essential in computer science. This section explains how to convert integers between different bases and introduces algorithms for arithmetic operations, division, and modular exponentiation, which are crucial for cryptography.

2. Integer Representations

An integer nnn in base bbb is represented as:

$n=akbk+ak−1bk−1+⋯+a1b+a0n = a_k b^k + a_{k-1} b^{k-1} + \dots + a_1 b + a_0$n=ak bk+ak−1 bk−1+⋯+a1 b+a0

where aja_jaj is a digit between 0 and $b−1b-1$b−1, and $ak≠0a_k \neq 0$ak ≠0.

Example 1: Decimal Expansion of a Binary Number

Convert $(101011111)2(101011111)_2$(101011111)2 to decimal:

$(101011111)2=1\cdot28+0\cdot27+1\cdot26+0\cdot25+1\cdot24+1\cdot23+1\cdot22+1\cdot21+1\cdot20(101011111)_2 = 1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$(101011111)2
=1·28+0·27+1·26+0·25+1·24+1·23+1·22+1·21+1·20 $=351= 351$=351

Example 2: Decimal Expansion of an Octal Number

Convert $(7016)8(7016)_8$(7016)8 to decimal:

$(7016)8=7\cdot83+0\cdot82+1\cdot81+6\cdot80(7016)_8 = 7 \cdot 8^3 + 0 \cdot 8^2 + 1 \cdot 8^1 + 6 \cdot 8^0$(7016)8 =7·83+0·82+1·81+6·80 $=3598= 3598$=3598

Example 3: Decimal Expansion of a Hexadecimal Number

Convert $(2AE0B)16(2AE0B)_{16}$(2AE0B)16 to decimal:

$(2AE0B)16=2\cdot164+10\cdot163+14\cdot162+0\cdot161+11\cdot160(2AE0B)_{16} = 2 \cdot 16^4 + 10 \cdot 16^3 + 14 \cdot 16^2 + 0 \cdot 16^1 + 11 \cdot 16^0$(2AE0B)16
=2·164+10·163+14·162+0·161+11·160 $=175627= 175627$=175627

3. Base Conversions

To convert a decimal number to base bbb:

1. Divide the number by bbb, recording the quotient and remainder.
2. Use the remainder as a digit (starting from the right).
3. Repeat until the quotient becomes zero.

Example 4: Decimal to Octal Conversion

Convert $(12345)10(12345)_{10}$(12345)10 to octal:

$12345\div8=1543 \text{ remainder } 112345 \div 8 = 1543 \text{ remainder } 1$12345÷8=1543 remainder 1 $1543\div8=192 \text{ remainder } 71543 \div 8 = 192 \text{ remainder } 7$1543÷8=192 remainder 7 $192\div8=24 \text{ remainder } 0192 \div 8 = 24 \text{ remainder } 0$192÷8=24 remainder 0 $24\div8=3 \text{ remainder } 024 \div 8 = 3 \text{ remainder } 0$24÷8=3 remainder 0 $3\div8=0 \text{ remainder } 33 \div 8 = 0 \text{ remainder } 3$3÷8=0 remainder 3

Thus, $(12345)10=(30071)8(12345)_{10} = (30071)_{8}$(12345)10 =(30071)8 .

Example 5: Decimal to Hexadecimal Conversion

Convert $(177130)10(177130)_{10}$(177130)10 to hexadecimal:

$177130 \div 16 = 11070 \text{ remainder } A177130 \mid div\ 16 = 11070 \mid text\{ remainder \}$ A177130÷16=11070 remainder A $11070 \div 16 = 691 \text{ remainder } E11070 \mid div\ 16 = 691 \mid text\{ remainder \}$ E11070÷16=691 remainder E $691 \div 16 = 43 \text{ remainder } 3691 \mid div\ 16 = 43 \mid text\{ remainder \}$ $3$691÷16=43 remainder 3 $43 \div 16 = 2 \text{ remainder } B43 \mid div\ 16 = 2 \mid text\{ remainder \}$ B43÷16=2 remainder B $2 \div 16 = 0 \text{ remainder } 22 \mid div\ 16 = 0 \mid text\{ remainder \}$ $2$2÷16=0 remainder 2

Thus, $(177130)10 = (2B3EA)16(177130)_{10} = (2B3EA)_{16}$(177130)10 =(2B3EA)16 .

Example 6: Decimal to Binary Conversion

Convert $(241)_{10}$ to binary:

241÷2=120 remainder 1

120÷2=60 remainder 0

$60 \div 2 = 30 \text{ remainder } 0$

30÷2=15 remainder 0

$15 \div 2 = 7 \text{ remainder } 1$

7÷2=3 remainder 1

$3 \div 2 = 1 \text{ remainder } 1$

$1 \div 2 = 0 \text{ remainder } 1$

Thus, $(241)_{10} = (11110001)_{2}$.

4. Quick Conversions between Binary, Octal, and Hexadecimal

- Each **octal** digit corresponds to **three binary** digits.
    - Example: $(111001)2 = (71)8(111001)_{2} = (71)_{8}$(111001)2 =(71)8 .
- Each **hexadecimal** digit corresponds to **four binary** digits.

- Example: $(11100101)2=(E5)16(11100101)_{2} = (E5)_{16}$(11100101)2 =(E5)16 .

Table: Decimal, Hexadecimal, and Octal Equivalents (0–15)

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

4.2.3 Algorithms for Integer Operations

ADDITION ALGORITHM

A procedure to perform addition can be based on the usual method for adding numbers with pencil and paper. This method proceeds by adding pairs of binary digits together with carries, when they occur, to compute the sum of two integers. To add a and b, first add their rightmost bits. This gives $a0 + b0 = c0 \cdot 2 + s0$, where s0 is the rightmost bit in the binary expansion of a + b and c0 is the carry, which is either 0 or 1. Then add the next pair of bits and the carry, $a1 + b1 + c0 = c1 \cdot 2 + s1$, where s1 is the next bit (from the right) in the binary expansion of a + b, and c1 is the carry.

Continue this process, adding the corresponding bits in the two binary expansions and the carry, to determine the next bit from the right in the binary expansion of a + b. At the last

stage, add an−1 , bn−1 , and cn−2 to obtain cn−1 · 2 + sn−1 . The leading bit of the sum is sn = cn−1 . This procedure produces the binary expansion of the sum, namely, a + b = (sn sn−1 sn−2 … s1 s0)$_2$.

EXAMPLE *How many additions of bits are required to use Algorithm 2 to add two integers with n bits (or less) in their binary representations?*

Solution: Two integers are added by successively adding pairs of bits and, when it occurs, a carry. Adding each pair of bits and the carry requires two additions of bits. Thus, the total number of additions of bits used is less than twice the number of bits in the expansion. Hence, the number of additions of bits used by Algorithm 2 to add two n-bit integers is O(n).

MULTIPLICATION ALGORITHM *Next, consider the multiplication of two n-bit integers a and b. The conventional algorithm (used when multiplying with pencil and paper) works as follows. Using the distributive law, we see that*

$$ab = a(b_0 2^0 + b_1 2^1 + \cdots + b_{n-1} 2^{n-1}) = a(b_0 2^0) + a(b_1 2^1) + \cdots + a(b_{n-1} 2^{n-1}).$$

We can compute ab using this equation. We first note that $ab_j = a$ *if* $b_j = 1$ *and* $ab_j = 0$ *if* $b_j = 0$*. Each time we multiply a term by 2, we shift its binary expansion one place to the left and add a zero at the tail end of the expansion. Consequently, we can obtain* $(ab_j)2^j$ *by shifting the binary expansion of* ab_j j *places to the left, adding j zero bits at the tail end of this binary expansion. Finally, we obtain ab by adding the n integers* $ab_j 2^j$**,** $j = 0, 1, 2, \ldots, n-1$.

ALGORITHM FOR div AND mod *Given integers a and d, d > 0, we can find q = a div d and r = a mod d using Algorithm 4. In this brute-force algorithm, when a is positive we subtract d from a as many times as necessary until what is left is less than d. The number of times we perform this subtraction is the quotient and what is left over after all these subtractions is the remainder. Algorithm 4 also covers the case where a is negative. This algorithm finds the quotient q and remainder r when |a| is divided by d. Then, when a < 0 and r > 0, it uses these to find the quotient −(q + 1) and remainder d − r when a is divided by d.*

4.2.3 Modular Exponentiation

In cryptography, efficiently computing $b^n \bmod m$ is essential, especially when b, n, and m are large. A direct computation of b^n followed by modulo reduction is impractical due to excessive memory and time requirements.

To optimize this, we use the **binary expansion** of n and apply **successive squaring** to compute $b^n \bmod m$ **m** efficiently. The key idea is:

1. Express **n** in binary:

$n=(ak−1ak−2…a1a0)2n = (a_{k-1} a_{k-2} … a_1 a_0)_2$n=(ak−1 ak−2 …a1 a0)2

which expands as:

bn=bak−1 2k−1·bak−2 2k−2⋯ba1 21·ba0 20

2. Compute powers efficiently using successive squaring:
   a. Compute $b,b2,b4,b8,…b, b^2, b^4, b^8, …$**b,b2,b4,b8,…** modulo mm**m**
   b. Multiply only the necessary terms $b^{2^j} \bmod m$ where the corresponding $a_j=1$.
   c. Reduce the result modulo **m** after each multiplication to keep numbers manageable.

Example

To compute $3^{11} \bmod 3$:

- Convert **11 →** $(1011)_2$
- Compute successive squares:
  - $3^2=9$
  - $3^4=92=81$
  - $3^8 = 81^2=6561$
- Multiply selected terms:
  - 311= 38 × 32 × 31= 6561 × 9 × 3 =177,147

This **reduces computation to** $O(log⁡2n)$ **multiplications**, making it efficient for large exponents.

Primes and Greatest Common Divisors

4.3.1 Primes

Every integer greater than 1 is divisible by at least two integers, because a positive integer is divisible by 1 and by itself. Positive integers that have exactly two different positive integer factors are called primes.

Definition 1 - An integer p greater than 1 is called prime if the only positive factors of p are 1 and p. A positive integer that is greater than 1 and is not prime is called composite.

Remark: The integer 1 is not prime, because it has only one positive factor. Note also that an integer n is composite if and only if there exists an integer a such that a | n and $1 < a < n$

THEOREM 1: **THE FUNDAMENTAL THEOREM OF ARITHMETIC** Every integer greater than 1 can be written uniquely as a prime or as the product of two or more primes, where the prime factors are written in order of nondecreasing size.

4.3.1.1 Trial Division

In cryptology, large primes are used in some methods for making messages secret. One procedure for showing that an integer is prime is based on the following observation.

THEOREM 2: If n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$.

Proof: If n is composite, by the definition of a composite integer, we know that it has a factor a with $1 < a < n$. Hence, by the definition of a factor of a positive integer, we have ab, where b is a positive integer greater than 1. We will show that $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. If $a > \sqrt{n}$ and $b > \sqrt{n}$, then $ab > \sqrt{n} \cdot \sqrt{n} = n$, which is a contradiction. Consequently, $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. Because both a and b are divisors of n, we see that n has a positive divisor not exceeding $\sqrt{n}$. This divisor is either prime or, by the fundamental theorem of arithmetic, Has a prime divisor less than itself. In either case, n has a prime divisor less than or equal to $\sqrt{n}$.

From Theorem 2, it follows that an integer is prime if it is not divisible by any prime less than or equal to its square root. This leads to the brute-force algorithm known as ***trial division***. To use trial division we divide n by all primes not exceeding $\sqrt{n}$ and conclude that n is prime if it is not divisible by any of these primes.

4.3.2 Greatest Common Divisors

The largest integer that divides both of two integers is called the ***greatest common divisor*** of these integers.

Definition 2: Let a and b be integers, not both zero. The largest integer d such that d | a and d | b is called the greatest common divisor of a and b. The greatest common divisor of a and b is denoted by gcd(a, b).

The greatest common divisor of two integers, not both zero, exists because the set of common divisors of these integers is nonempty and finite. One way to find the greatest common divisor of two integers is to find all the positive common divisors of both integers and then take the largest divisor. This is done in Examples 10 and 11. Later, a more efficient method of finding greatest common divisors will be given.

Definition 3: The integers a and b are relatively prime if their greatest common divisor is 1.

EXAMPLE: It follows that the integers 17 and 22 are relatively prime, because
gcd(17, 22) = 1.

4.3.2 Least common Multiple

Prime factorizations can also be used to find the least common multiple of two integers.

Definition 4: The least common multiple of the positive integers a and b is the smallest positive integer that is divisible by both a and b. The least common multiple of a and b is denoted by lcm(a, b).

The least common multiple exists because the set of integers divisible by both a and b is nonempty (because ab belongs to this set, for instance), and every nonempty set of positive integers has a least element (by the well-ordering property). Suppose that the prime factorizations of a and b are as before. Then the least common multiple of a and b is given by max(a1 , b1) max(a2 , b2) p2 ⋯ pnmax(an , bn) , lcm(a, b) = p1 where max(x, y) denotes the maximum of the two numbers x and y. This formula is valid because a common multiple of a and b has at least max(ai , bi) factors of pi in its prime factorization, and the least common multiple has no other prime factors besides those in a and b.

4.3.2 The Euclidean Algorithm

Computing the greatest common divisor of two integers directly from the prime factorizations of these integers is inefficient. The reason is that it is time-consuming to find prime factorizations. There is a more efficient method of finding the greatest common divisor, called the Euclidean algorithm.

EXAMPLE: *Here is how it is used to find gcd(91, 287).*

First, divide 287, the larger of the two integers, by 91, the smaller, to obtain 287 = 91 · 3 + 14. Any divisor of 91 and 287 must also be a divisor of 287 − 91 · 3 = 14. Also, any divisor of 91 and 14 must also be a divisor of 287 = 91 · 3 + 14. Hence, the greatest common divisor of 91 and 287 is the same as the greatest common divisor of 91 and 14. This means that the problem of finding gcd(91, 287) has been reduced to the problem of finding gcd(91, 14). Next, divide 91 by 14 to obtain 91 = 14 · 6 + 7. Because any common divisor of 91 and 14 also divides 91 − 14 · 6 = 7 and any common divisor of 14 and 7 divides 91, it follows that gcd(91, 14) = gcd(14, 7). Continue by dividing 14 by 7, to obtain 14 = 7 · 2. Because 7 divides 14, it follows that gcd(14, 7) = 7. Furthermore, because gcd(287, 91) = gcd(91, 14) = gcd(14, 7) = 7, the original problem has been solved.

Remark: Here is how the Euclidean Algorithm works in general:
The algorithm is based on the principle that:

$$\gcd(a,b)=\gcd(b,a \bmod b)$$

This property is derived from the fact that the GCD of two numbers does not change if the larger number is replaced by its remainder when divided by the smaller number.

Steps of the Euclidean Algorithm

1. **Start with two numbers a and (a≥b).**
2. **Divide *a* by *b*** and compute the remainder: r=amodb
3. **Replace *a* with *b* and b with r:** $(a,b) \leftarrow (b,r)$
4. **Repeat the process until $b=0$**
5. **The GCD is the last nonzero value of a.**

Complexity and Efficiency

- The algorithm runs in $O(\log(\max(a,b)))$ time complexity.
- Each step significantly reduces the size of the numbers involved.
- It is **much faster** than checking all common divisors.

Solving Congruences

4.4.1 Linear Congruences

A congruence of the form $ax \equiv b \pmod{m}$, where m is a positive integer, a and b are integers, and x is a variable, is called a linear congruence.

THEOREM 1: If a and m are relatively prime integers and $m > 1$, then an inverse of a modulo m exists. Furthermore, this inverse is unique modulo m. (That is, there is a unique positive integer a less than m that is an inverse of a modulo m and every other inverse of a modulo m is congruent to a modulo m.)

Proof: ***B***ecause gcd(a, m) = 1, there are integers s and t such that $sa + tm = 1$. This implies that $sa + tm \equiv 1 \pmod{m}$. Because $tm \equiv 0 \pmod{m}$, it follows that $sa \equiv 1 \pmod{m}$.

EXAMPLE: Find an inverse of 3 modulo 7 by first finding Bézout coefficients of 3 and 7. (Note that we have already shown that 5 is an inverse of 3 modulo 7 by inspection.)

Solution: Because gcd(3, 7) = 1, Theorem 1 tells us that an inverse of 3 modulo 7 exists. The Euclidean algorithm ends quickly when used to find the greatest common divisor of 3 and 7: $7 = 2 \cdot 3 + 1$.

From this equation we see that $-2 \cdot 3 + 1 \cdot 7 = 1$.

This shows that −2 and 1 are Bézout coefficients of 3 and 7. We see that −2 is an inverse of 3 modulo 7. Note that every integer congruent to −2 modulo 7 is also an inverse of 3, such as 5, −9, 12, and so on.

4.4.2 The Chinese Remainder Theorem

This a mathematical principle that solves systems of modular equations by finding a unique solution from the remainder of the division.

THE CHINESE REMAINDER THEOREM *Let $m_1, m_2, \ldots, m_n$ be pairwise relatively prime positive integers greater than one and $a_1, a_2, \ldots,$ an arbitrary integers. Then the system $x \equiv a_1 \pmod{m_1}$, $x \equiv a_2 \pmod{m_2}$, $x \equiv a_n \pmod{m_n}$ has a unique solution modulo $m = m_1 m_2 \cdots m_n$. (That is, there is a solution x with $0 \leq x < m$, and all other solutions are congruent modulo m to this solution.)*

According to the Theorem, if in a given set of equations, each equation has a different number (say, n1, n2, ..., nk), and these numbers are all relatively prime to each other, and also have some numbers (say, a1, a2,..., ak), then there is a unique solution (let's call it x) that satisfies all the equations at once.

This solution is found modulo the product of all the numbers ($N = n1 \cdot n2 \cdot \ldots \cdot nk$). Therefore, x satisfies each equation by leaving the same remainder when divided by its corresponding number (ni) as the given number (ai).

Proof: To establish this theorem, we need to show that a solution exists and that it is unique modulo m. We will show that a solution exists by describing a way to construct this solution;

To construct a simultaneous solution, first let $Mk = m/mk$ for $k = 1, 2, \ldots, n$. That is, Mk is the product of the moduli except for mk . Because mi and mk have no common factors greater than 1 when $i \neq k$, it follows that $\gcd(mk, Mk) = 1$. Consequently, by Theorem 1, we know that there is an integer yk , an inverse of Mk modulo mk , such that $Mk\, yk \equiv 1 \pmod{mk}$. To construct a simultaneous solution, form the sum $x = a1\, M1\, y1 + a2\, M2\, y2 + \cdots + an\, Mn\, yn$. We will now show that x is a simultaneous solution. First, note that because $Mj \equiv 0 \pmod{mk}$ whenever $j \neq k$, all terms except the kth term in this sum are congruent to 0 modulo mk . Because $Mk\, yk \equiv 1 \pmod{mk}$ we see that $x \equiv ak\, Mk\, yk \equiv ak \pmod{mk}$, for $k = 1, 2, \ldots, n$. We have shown that x is a simultaneous solution to the n congruences.

NB: *"If a number is divided by several other numbers that have no common factors, one can find the remainder when dividing by the product of those numbers."*

4.4.1 Application of Congruences

4.4.1.1 Hashing Functions

The central computer at an insurance company maintains records for each of its customers. How can memory locations be assigned so that customer records can be retrieved quickly? The solution to this problem is to use a suitably chosen hashing function. Records are identified using a key, which uniquely identifies each customer's records. For instance, customer records are often identified using the Social Security number of the customer as the key. A hashing function h assigns memory location h(k) to the record that has k as its key. In practice, many different hashing functions are used. One of the most common is the function $h(k) = k \bmod m$ where m is the number of available

memory locations. Hashing functions should be easily evaluated so that files can be quickly located. The hashing function h(k) = k mod m meets this requirement; to find h(k), we need only compute the remainder when k is divided by m. Furthermore, the hashing function should be onto, so that all memory locations are possible. The function h(k) = k mod m also satisfies this property.

EXAMPLE: Find the memory locations assigned by the hashing function h(k) = k mod 111 to the records of customers with Social Security numbers 064212848 and 037149212.

Solution: The record of the customer with Social Security number 064212848 is assigned to memory location 14, because h(064212848) = 064212848 mod 111 = 14. Similarly, because h(037149212) = 037149212 mod 111 = 65, the record of the customer with Social Security number 037149212 is assigned to memory location 65.

Because a hashing function is not one-to-one (because there are more possible keys than memory locations), more than one file may be assigned to a memory location. When this happens, we say that a collision occurs. One way to resolve a collision is to assign the first free location following the occupied memory location assigned by the hashing function.

4.4.1.2 Pseudorandom Numbers

Randomly chosen numbers are often needed for computer simulations. Because numbers generated by systematic methods are not truly random, they are called pseudorandom numbers. The most commonly used procedure for generating pseudorandom numbers is the linear congruential method.

We choose four integers: the modulus m, multiplier a, increment c, and seed x0 , with $2 \leq a < m$, $0 \leq c < m$, and $0 \leq x0 < m$. We generate a sequence of pseudorandom numbers {xn }, with $0 \leq xn < m$ for all n, by successively using the recursively defined function $xn+1 = (axn + c) \bmod m$.

Many computer experiments require the generation of pseudorandom numbers between 0 and 1. To generate such numbers, we divide numbers generated with a linear congruential generator by the modulus: that is, we use the numbers xn /m.

EXAMPLE: Find the sequence of pseudorandom numbers generated by the linear congruential method with modulus m = 9, multiplier a = 7, increment c = 4, and seed x0=3.

Solution: We compute the terms of this sequence by successively using the recursively defined function $xn+1 = (7xn + 4) \bmod 9$, beginning by inserting the seed x0 = 3 to find x1 . We find that

$x1 = 7x0 + 4 \bmod 9 = 7 \cdot 3 + 4 \bmod 9 = 25 \bmod 9 = 7,$

$x_2 = 7x_1 + 4 \bmod 9 = 7 \cdot 7 + 4 \bmod 9 = 53 \bmod 9 = 8,$

$x_3 = 7x_2 + 4 \bmod 9 = 7 \cdot 8 + 4 \bmod 9 = 60 \bmod 9 = 6,$

$x_4 = 7x_3 + 4 \bmod 9 = 7 \cdot 6 + 4 \bmod 9 = 46 \bmod 9 = 1,$

$x_5 = 7x_4 + 4 \bmod 9 = 7 \cdot 1 + 4 \bmod 9 = 11 \bmod 9 = 2,$

$x_6 = 7x_5 + 4 \bmod 9 = 7 \cdot 2 + 4 \bmod 9 = 18 \bmod 9 = 0,$

$x_7 = 7x_6 + 4 \bmod 9 = 7 \cdot 0 + 4 \bmod 9 = 4 \bmod 9 = 4,$

$x_8 = 7x_7 + 4 \bmod 9 = 7 \cdot 4 + 4 \bmod 9 = 32 \bmod 9 = 5,$

$x_9 = 7x_8 + 4 \bmod 9 = 7 \cdot 5 + 4 \bmod 9 = 39 \bmod 9 = 3.$

Because $x_9 = x_0$ and because each term depends only on the previous term, we see that the sequence

$$3, 7, 8, 6, 1, 2, 0, 4, 5, 3, 7, 8, 6, 1, 2, 0, 4, 5, 3, \ldots$$

is generated. This sequence contains nine different numbers before repeating.

4.4.1.3 Check Digits

Congruences are used to check for errors in digit strings. A common technique for detecting errors in such strings is to add an extra digit at the end of the string. This final digit, or check digit, is calculated using a particular function. Then, to determine whether a digit string is correct, a check is made to see whether this final digit has the correct value. We begin with an application of this idea for checking the correctness of bit strings.

EXAMPLE 4: Parity Check Bits Digital information is represented by bit string, split into blocks of a specified size. Before each block is stored or transmitted, an extra bit, called a parity check bit, can be appended to each block. The parity check bit x_{n+1} for the bit string $x_1 x_2 \ldots x_n$ is defined by $x_{n+1} = x_1 + x_2 + \cdots + x_n \bmod 2$. It follows that x_{n+1} is 0 if there are an even number of 1 bits in the block of n bits and it is 1 if there are an odd number of 1 bits in the block of n bits. When we examine a string that includes a parity check bit, we know that there is an error in it if the parity check bit is wrong. However, when the parity check bit is correct, there still may be an error. A parity check can detect an odd number of errors in the previous bits, but not an even number of errors. (See Exercise 14.) Suppose we receive in a transmission the bit strings 01100101 and 11010110, each ending with a parity check bit. Should we accept these bit strings as correct?

Solution: Before accepting these strings as correct, we examine their parity check bits. The parity check bit of the first string is 1. Because $0 + 1 + 1 + 0 + 0 + 1 + 0 \equiv 1 \pmod 2$, the parity check bit is correct. The parity check bit of the second string is 0. We find that $1 + 1 + 0 + 1 + 0 + 1 + 1 \equiv 1 \pmod 2$, so the parity check is incorrect. We conclude that the first string may have been transmitted correctly and we know for certain that the second string

was transmitted incorrectly. We accept the first string as correct (even though it still may contain an even number of errors), but we reject the second string.

Check bits computed using congruences are used extensively to verify the correctness of various kinds of identification numbers. Examples above show how check bits are computed for codes that identify products (Universal Product Codes) and books (International Standard Book Numbers). The preambles to Exercises 18, 28, and 32 introduce the use of congruences to find and use check digits in money order numbers, airline ticket numbers, and identification numbers for periodicals, respectively. Note that congruences are also used to compute check digits for bank account numbers, drivers license numbers, credit card numbers, and many other types of identification numbers.

Cryptography

4.5.1 Classical Cryptography

One of the earliest known uses of cryptography was by Julius Caesar. He made messages secret by shifting each letter three letters forward in the alphabet (sending the last three letters of the alphabet to the first three). For instance, using this scheme the letter B is sent to E and the letter X is sent to A.

This is an example of encryption, that is, the process of making a message secret. To express Caesar's encryption process mathematically, first replace each letter by an element of Z_{26} , that is, an integer from 0 to 25 equal to one less than its position in the alphabet. For example, replace A by 0, K by 10, and Z by 25. Caesar's encryption method can be represented by the function f that assigns to the nonnegative integer p, $p \leq 25$, the integer f (p) in the set $\{0, 1, 2, \ldots, 25\}$ with $f(p) = (p + 3) \bmod 26$. In the encrypted version of the message, the letter represented by p is replaced with the letter represented by $(p + 3) \bmod 26$.

EXAMPLE: What is the secret message produced from the message "MEET YOU IN THE PARK" using the Caesar cipher?

Solution: First replace the letters in the message with numbers. This produces

12 4 4 19 24 14 20 8 13 19 7 4 15 0 17 10.

Now replace each of these numbers p by f (p) = (p + 3) mod 26. This gives

15 7 7 22 1 17 23 11 16 22 10 7 18 3 20 13.

Translating this back to letters produces the encrypted message “PHHW BRX LQ WKH SDUN.”

To recover the original message from a secret message encrypted by the Caesar cipher, the function f −1 , the inverse of f , is used. Note that the function f −1 sends an integer p from Z26, to f −1 (p) = (p − 3) mod 26. In other words, to find the original message, each letter is shifted back three letters in the alphabet, with the first three letters sent to the last three letters of the alphabet. The process of determining the original message from the encrypted message is called decryption. There are various ways to generalize the Caesar cipher. For example, instead of shifting the numerical equivalent of each letter by 3, we can shift the numerical equivalent of each letter by k, so that f (p) = (p + k) mod 26. Such a cipher is called a shift cipher. Note that decryption can be carried out using f −1 (p) = (p − k) mod 26. Here the integer k is called a key.

EXAMPLE: Encrypt the plaintext message “STOP GLOBAL WARMING” using the shift cipher with shift k = 11.312 4 / Number Theory and Cryptography

Solution: To encrypt the message “STOP GLOBAL WARMING” we first translate each letter to the corresponding element of Z26 . This produces the string

18 19 14 15 6 11 14 1 0 11 22 0 17 12 8 13 6.

We now apply the shift f (p) = (p + 11) mod 26 to each number in this string. We obtain

3 4 25 0 17 22 25 12 11 22 7 11 2 23 19 24 17.

Translating this last string back to letters, we obtain the ciphertext “DEZA RWZMLW HLCX-TYR.”

EXAMPLE: Decrypt the ciphertext message “LEWLYPLUJL PZ H NYLHA ALHJOLY” that was encrypted with the shift cipher with shift k = 7.

Solution: To decrypt the ciphertext “LEWLYPLUJL PZ H NYLHA ALHJOLY” we first translate the letters back to elements of Z26 . We obtain

11 4 22 11 24 15 11 20 9 11 15 25 7 13 24 11 7 0 0 11 7 9 14 11 24.

Next, we shift each of these numbers by −k = −7 modulo 26 to obtain

4 23 15 4 17 8 4 13 2 4 8 18 0 6 17 4 0 19 19 4 0 2 7 4 17.

Finally, we translate these numbers back to letters to obtain the plaintext. We obtain "EXPERIENCE IS A GREAT TEACHER." We can generalize shift ciphers further to slightly enhance security by using a function of the form f (p) = (ap + b) mod 26, where a and b are integers, chosen so that f is a bijection. (The function f (p) = (ap + b) mod 26 is a bijection if and only if gcd(a, 26) = 1.) Such a mapping is called an affine transformation, and the resulting cipher is called an affine cipher.

CRYPTANALYSIS The process of recovering plaintext from ciphertext without knowledge of both the encryption method and the key is known as cryptanalysis or breaking codes. In general, cryptanalysis is a difficult process, especially when the encryption method is unknown.

BLOCK CIPHERS

Shift ciphers and affine ciphers proceed by replacing each letter of the alphabet by another letter in the alphabet. Because of this, these ciphers are called character or monoalphabetic ciphers. Encryption methods of this kind are vulnerable to attacks based on the analysis of letter frequency in the ciphertext, as we just illustrated. We can make it harder to successfully attack ciphertext by replacing blocks of letters with other blocks of letters in- stead of replacing individual characters with individual characters; such ciphers are called ***block ciphers***.

We will now introduce a simple type of block cipher, called the transposition cipher. As a key we use a permutation σ of the set {1, 2, ... , m} for some positive integer m, that is, a one-to-one function from {1, 2, ... , m} to itself. To encrypt a message we first split its letters into blocks of size m. (If the number of letters in the message is not divisible by m we add some random letters at the end to fill out the final block.) We encrypt the block p1 p2 ... pm as c1 c2 ... cm = pσ(1) pσ(2) ... , pσ(m) . To decrypt a ciphertext block c1 c2 ... cm , we transpose its letters using the permutation σ^{-1} , the inverse of σ. Example 6 illustrates encryption and decryption for a transposition cipher. Using the transposition cipher based on the permutation σ of the set {1, 2, 3, 4} with $\sigma(1) = 3$, $\sigma(2) = 1$, $\sigma(3) = 4$, and $\sigma(4) = 2$,

(a) Encrypt the plaintext message PIRATE ATTACK.

(b) Decrypt the ciphertext message SWUE TRAE OEHS, which was encrypted using this cipher.314 4 / Number Theory and Cryptography

Solution: (a) We first split the letters of the plaintext into blocks of four letters. We obtain PIRA TEAT TACK. To encrypt each block, we send the first letter to the third position, the second letter to the first position, the third letter to the fourth position, and the fourth letter to the second position. We obtain IAPR ETTA AKTC.

(b) We note that $\sigma -1$, the inverse of σ, sends 1 to 2, sends 2 to 4, sends 3 to 1, and sends 4 to 3. Applying $\sigma -1$ (m) to each block gives us the plaintext: USEW ATER HOSE. (Grouping together these letters to form common words, we surmise that the plaintext is USE WATER HOSE.)

CRYPTOSYSTEMS *We have defined two families of ciphers: shift ciphers and affine ciphers. We now introduce the notion of a cryptosystem, which provides a general structure for defining new families of ciphers.*

Definition 1 A cryptosystem is a five-tuple (P,C,K,E,D), where P is the set of plaintext strings, C is the set of ciphertext strings, K is the keyspace (the set of all possible keys), E is the set of encryption functions, and D is the set of decryption functions. We denote by Ek the encryption function in □ corresponding to the key k and Dk the decryption function in □ that decrypts ciphertext that was encrypted using Ek , that is, Dk (Ek (p)) = p, for all plaintext strings p. We now illustrate the use of the definition of a cryptosystem.

EXAMPLE: Describe the family of shift ciphers as a cryptosytem.

Solution: To encrypt a string of English letters with a shift cipher, we first translate each letter to an integer between 0 and 25, that is, to an element of Z26 . We then shift each of these integers by a fixed integer modulo 26, and finally, we translate the integers back to letters. To apply the definition of a cryptosystem to shift ciphers, we assume that our messages are already integers, that is, elements of Z26 . That is, we assume that the translation between letters and integers is outside of the cryptosystem. Consequently, both the set of plaintext strings □ and the set of ciphertext strings □ are the set of strings of elements of Z26 . The set of keys □ is the set of possible shifts, so □ = Z26 . The set □ consists of functions of the form Ek (p) = (p + k) mod 26, and the set D of decryption functions is the same as the set of encrypting functions where Dk (p) = (p − k) mod 26.

The concept of a cryptosystem is useful in the discussion of additional families of ciphers and is used extensively in cryptography.

Public Key Cryptography

All classical ciphers, including shift ciphers and affine ciphers, are examples of private key cryptosystems. In a private key cryptosystem, once you know an encryption key, you can quickly find the decryption key. So, knowing how to encrypt messages using a particular key allows you to decrypt messages that were encrypted using this key.

For example, when a shift cipher is used with encryption key k, the plaintext integer p is sent to c = (p + k) mod 26. Decryption is carried out by shifting by −k; that is, p = (c − k) mod 26. So knowing how to encrypt with a shift cipher also tells you how to decrypt.

When a private key cryptosystem is used, two parties who wish to communicate in secret must share a secret key. Because anyone who knows this key can both encrypt and decrypt messages, two people who want to communicate securely need to securely exchange this key. The shift cipher and affine cipher cryptosystems are private key cryptosystems. They are quite simple and are extremely vulnerable to cryptanalysis. However, the same is not true of many modern private key cryptosystems. In particular, the current US government standard for private key cryptography, the Advanced Encryption Standard (AES), is extremely complex and is considered to be highly resistant to cryptanalysis. However, it still shares the property that for secure communications keys be shared. Furthermore, for extra security, a new key is used for each communication session between two parties, which requires a method for generating keys and securely sharing them.

To avoid the need for keys to be shared by every pair of parties that wish to communicate securely, in the *1970s cryptologists introduced the concept of public key cryptosystems*. When such cryptosystems are used, knowing how to send an encrypted message does not help decrypt messages. In such a system, everyone can have a publicly known encryption key. Only the decryption keys are kept secret, and only the intended recipient of a message can decrypt it, because, as far as it is currently known, knowledge of the encryption key does not let someone recover the plaintext message without an extraordinary amount of work (such as billions of years of computer time).

With regards to public key cryptosystems, the most commonly used public key cryptosystem, known as the RSA system. Although public key cryptography has the advantage that two parties who wish to communicate securely do not need to exchange keys, it has the disadvantage that encryption and decryption can be extremely time consuming.

4.6.1 Cryptographic Protocols

These are exchanges of messages carried out by two or more parties to achieve a particular security goal.

KEY EXCHANGE - Below is a protocol that two parties can use to exchange a secret key over an insecure communications channel without having shared any information in the past. Generating a key that two parties can share is important for many applications of cryptography.

For example, for two people to send secure messages to each other using a private key cryptosystem they need to share a common key. The protocol is known as the ***Diffie-Hellman key agreement protocol, after Whitfield Diffie and Martin Hellman.***

Suppose that Alice and Bob want to share a common key. The protocol follows these steps, where the computations are done in Zp .

(1) Alice and Bob agree to use a prime p and a primitive root a of p.

(2) Alice chooses a secret integer k1 and sends ak1 mod p to Bob.

(3) Bob chooses a secret integer k2 and sends ak2 mod p to Alice.

(4) Alice computes (ak2)k1 mod p.

(5) Bob computes (ak1)k2 mod p.

At the end of this protocol, Alice and Bob have computed their shared key, namely,

$(a^{k2})k1 \bmod p = (a^{k1})^{k2} \bmod p$.

To analyze the security of this protocol, note that the messages sent in steps (1), (2), and (3) are not assumed to be sent securely. We can even assume that these communications were in the clear and that their contents are public information. So, p, a, ak1 mod p, and ak2 mod p are assumed to be public information. The protocol ensures that k1 , k2 , and the common key (ak2)k1 mod p = (ak1)k2 mod p are kept secret. To find the secret information from this public information requires that an adversary solves instances of the discrete logarithm problem, because the adversary would need to find k1 and k2 from ak1 mod p and ak2 mod p, respectively. Furthermore, no other method is known for finding the shared key using just the public information.

DIGITAL SIGNATURES - Not only can cryptography be used to secure the confidentiality of a message, but it also can be used so that the recipient of the message knows that it came from the person they think it came from. Below shows how a message can be sent so that a recipient of the message will be sure that the message came from the purported sender of the message. In particular, shows how this can be accomplished using the RSA cryptosystem to apply a digital signature to a message.

Suppose that Alice's RSA public key is (n, e) and her private key is d. Alice encrypts a plaintext message x using the encryption function E(n,e) (x) = xe mod n. She decrypts a ciphertext message y using the decryption function D(n,e) = xd mod n. Alice wants to send the message M so that everyone who receives the message knows that it came from her. Just as in RSA encryption, she translates the letters into their numerical equivalents and splits the resulting string into blocks m1 , m2 , … , mk such that each block is the same size, which is as large as possible so that 0 ≤ mi ≤ n for i = 1, 2, …, k. She then applies her decryption function D(n,e) to each block, obtaining Dn,e (mi), i = 1, 2, … , k. She sends the result to all intended recipients of the message. When a recipient receives her message, they apply Alice's encryption function E(n,e) to each block, which everyone has available because Alice's key (n, e) is public information. The result is the original plaintext block because E(n,e) (D(n,e) (x)) = x. So, Alice can send her message to as many people as she wants and by signing it in this way, every recipient can be sure it came from Alice.

EXAMPLE: Suppose Alice's public RSA cryptosystem key is the same as in Example above. That is, $n = 43 \cdot 59 = 2537$ and $e = 13$. Her decryption key is $d = 937$, as described in Example 9. She wants to send the message "MEET AT NOON" to her friends so that they are sure it came from her. What should she send?

Solution: Alice first translates the message into blocks of digits, obtaining 1204 0419 0019 1314 1413 (as the reader should verify). She then applies her decryption transformation $D(2537,13)(x) = x^{937} \bmod 2537$ to each block. Using fast modular exponentiation (with the help of a computational aid), she finds that $1204^{937} \bmod 2537 = 817$, $419^{937} \bmod 2537 = 555$, $19^{937} \bmod 2537 = 1310$, $1314^{937} \bmod 2537 = 2173$, and $1413^{937} \bmod 2537 = 1026$. So, the message she sends, split into blocks, is 0817 0555 1310 2173 1026. When one of her friends gets this message, they apply her encryption transformation $E(2537,13)$ to each block. When they do this, they obtain the blocks of digits of the original message, which they translate back to English letters.