Contents

Preface for the Second Edition

The second edition of my introduction to cryptography contains updates and new material. I have updated the discussion of the security of encryption and signature schemes and the state of the art in factoring and computing discrete logarithms. I have added descriptions of time-memory trade of attacks and algebraic attacks on block ciphers, the Advanced Encryption Standard (AES), the Secure Hash Algorithm (SHA-1), secret sharing schemes, and undeniable and blind signatures. I have also corrected the errors that have been reported to me. I thank the readers of the first edition for all comments and suggestions.

October 2003 *Johannes Buchmann*

Preface

Cryptography is a key technology in electronic security systems. Modern cryptograpic techniques have many uses, such as to digitally sign documents, for access control, to implement electronic money, and for copyright protection. Because of these important uses it is necessary that users be able to estimate the efficiency and security of cryptographic techniques. It is not sufficient for them to know only how the techniques work.

This book is written for readers who want to learn about modern cryptographic algorithms and their mathematical foundation but who do not have the necessary mathematical background. It is my goal to explain the basic techniques of modern cryptography, including the necessary mathematical results from linear algebra, algebra, number theory, and probability theory. I only assume basic mathematical knowledge.

The book is based on courses in cryptography that I have been teaching at the Technical University Darmstadt, since 1996. I thank all students who attended the courses and who read the manuscript carefully for their interest and support. In particular, I would like to thank Harald Baier, Gabi Barking, Manuel Breuning, Safuat Hamdy, Birgit Henhapl, Michael Jacobson (who also corrected my English), Markus Maurer, Andreas Meyer, Stefan Neis, Sachar Paulus, Thomas

Pfahler, Marita Skrobic, Edlyn Teske, Patrick Theobald, and Ralf-Philipp Weinmann. I also thank the staff at Springer-Verlag, in particular Martin Peters, Agnes Herrmann, Claudia Kehl, Ina Lindemann, and Terry Kornak, for their support in the preparation of this book.

Darmstadt
June 1999 *Johannes Buchmann*

1 CHAPTER Integers

Integers play a fundamental role in cryptography. In this chapter we present important properties of integers and describe fundamental algorithms. Efficient Implementations of the algorithms described in this chapter can, for example, be found in the C++ library LiDIA [46].

1.1 Basics

As usual, $\mathbb{N} = \{1, 2, 3, 4, 5, \ldots\}$ is the set of positive integers and $\mathbb{Z} = \{0, \pm 1, \pm 2, \pm 3, \ldots\}$ is the set of *integers*. The rational numbers are denoted by $\mathbb{Q}$ and the real numbers by $\mathbb{R}$.

Clearly, we have $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R}$. Real numbers (including integers and rational numbers) can be added and multiplied. We assume that this is known.

We use the following rules.

If the product of two real numbers is zero, then at least one factor is zero so it is impossible that both factors are non-zero but the product is zero.

Real numbers can be compared. For example, $\sqrt{2}$ is less than 2 but greater than 1. If a real number α is less than another real number β, then we write $\alpha < \beta$. If α is less than or equal to β, we write $\alpha \leq \beta$. If α is greater than β, then we write $\alpha > \beta$. If α is greater than or equal to β we write $\alpha \geq \beta$. If γ is another real number, then $\alpha < \beta$ implies $\alpha + \gamma < \beta + \gamma$. Analogous statements hold for $\leq$, $>$, and $\geq$. If $0 < \alpha$ and $0 < \beta$, then $0 < \alpha\beta$.

A set M of real numbers is called *bounded from below* if there is a real number γ such that all elements of M are greater than γ. We also say that M is bounded from below by γ. For example, the set of positive integers is bounded from below by 0, but the set of even integers is not bounded from below. An important property of the integers is the fact that every set of integers that is bounded from below contains a smallest element. For example, the smallest positive integer is 1. In an analogous way one defines sets of real numbers that are bounded from above. Every set of integers that is bounded from above contains a greatest element.

For any real number α, we write

$$\lfloor \alpha \rfloor = \max\{b \in \mathbb{Z} : b \leq \alpha\}.$$

Hence, $\lfloor \alpha \rfloor$ is the greatest integer, which is less than or equal to α. This number exists because the set $\{b \in \mathbb{Z} : b \leq \alpha\}$ is bounded from above.

Example 1.1.1
We have $\lfloor 3.43 \rfloor = 3$ and $\lfloor -3.43 \rfloor = -4$.

Finally, we need *induction*: If a statement, which depends on a positive integer n, is true for $n = 1$ and if the truth for any integer m with $1 \leq m \leq n$ (or just for n) implies the truth for $n+1$, then the statement is true for any positive integer n.

In this chapter, lower case italic letters denote integers.

1.2 Divisibility

Definition 1.2.1
We say that a *divides* n if there is an integer b with $n = ab$.

If a divides n, then a is called a *divisor* of n, n is called a *multiple* of a, and we write $a \mid n$. We also say that n is *divisible* by a. If a is not a divisor of n, then we write $a \nmid n$.

Example 1.2.2
We have $13 \mid 182$ because $182 = 14 * 13$. Likewise, we have $-5 \mid 30$ because $30 = (-6) * (-5)$. The divisors of 30 are ± 1, ± 2, ± 3, ± 5, ± 6, ± 10, ± 15, ± 30.

Any integer a divides 0 because $0 = a * 0$. The only integer that is divisible by 0 is 0 because $a = 0 * b$ implies $a = 0$.

We prove a few simple rules.

Theorem 1.2.3
1. *If $a \mid b$ and $b \mid c$, then $a \mid c$.*
2. *If $a \mid b$, then $ac \mid bc$ for all c.*
3. *If $c \mid a$ and $c \mid b$, then $c \mid da + eb$ for all d and e.*
4. *If $a \mid b$ and $b \neq 0$, then $|a| \leq |b|$.*
5. *If $a \mid b$ and $b \mid a$, then $|a| = |b|$.*

Proof. 1. If $a \mid b$ and $b \mid c$, then there are f, g with $b = af$ and $c = bg$. This implies $c = bg = (af)g = a(fg)$.

2. If $a \mid b$, then there is f with $b = af$. Hence, $bc = (af)c = f(ac)$.

3. If $c \mid a$ and $c \mid b$, then there is f, g with $a = fc$ and $b = gc$. This implies $da + eb = dfc + egc = (df + eg)c$.

4. If $a \mid b$ and $b \neq 0$, then there is $f \neq 0$ with $b = af$. This implies $|b| = |af| \geq |a|$.

5. Suppose that $a \mid b$ and $b \mid a$. If $a = 0$, then $b = 0$ and vice versa. If $a \neq 0$ and $b \neq 0$, then 4. implies $|a| \leq |b|$ and $|b| \leq |a|$, and hence $|a| = |b|$. □

The following result is very important. It shows that division with remainder of integers is possible.

Theorem 1.2.4
If a and b are integers, $b > 0$, then there are uniquely determined integers q and r such that $a = qb + r$ and $0 \le r < b$, namely $q = \lfloor a/b \rfloor$ and $r = a - bq$.

Proof. If $a = qb+r$ and $0 \le r < b$, then $0 \le r/b = a/b-q < 1$. This implies $a/b - 1 < q \le a/b$; hence $q = \lfloor a/b \rfloor$. Conversely, $q = \lfloor a/b \rfloor$ and $r = a - bq$ satisfy the assertion. □

In the situation of Theorem 1.2.4, the integer q is called the (integral) *quotient* and r the *remainder* of the division of a by b. We write $r = a \bmod b$. If a is replaced by $a \bmod b$, then we say that a is *reduced* modulo b.

Example 1.2.5
If $a = 133$ and $b = 21$, then $q = 6$ and $r = 7$, so $133 \bmod 21 = 7$. Likewise, we have $-50 \bmod 8 = 6$.

1.3 Representation of Integers

In books, integers are written in decimal expansion. On computers, binary expansion is used. More generally, integers can be represented using the so-called g-adic expansion, which is explained in this section. For an integer $g > 1$ and a positive real number α, denote by $\log_g \alpha$ the logarithm for base g of α. For a set M, let M^k be the set of all sequences of length k with entries from M.

Example 1.3.1
We have $\log_2 8 = 3$ because $2^3 = 8$. Also, $\log_8 8 = 1$ because $8^1 = 8$.

Example 1.3.2
The sequence $(0, 1, 1, 1, 0)$ is an element of $\{0, 1\}^5$. Also $\{1, 2\}^2 = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$.

Theorem 1.3.3
Let g be an integer, $g > 1$. For each positive integer a, there is a uniquely determined positive integer k and a uniquely determined sequence

$$(a_1, \ldots, a_k) \in \{0, \ldots, g-1\}^k$$

with $a_1 \neq 0$ and

$$a = \sum_{i=1}^{k} a_i g^{k-i}. \tag{1.1}$$

In addition, $k = \lfloor \log_g a \rfloor + 1$, and a_i is the integral quotient of $a - \sum_{j=1}^{i-1} a_j g^{k-j}$ by g^{k-i} for $1 \le i \le k$.

Proof. Let a be a positive integer. If a can be represented as in (1.1), then $g^{k-1} \le a = \sum_{i=1}^{k} a_i g^{k-i} \le (g-1)\sum_{i=1}^{k} g^{k-i} = g^k - 1 < g^k$. Hence, $k = \lfloor \log_g a \rfloor + 1$. This proves the uniqueness of k. We prove the existence and the uniqueness of the sequence $(a_1, \ldots, a_k)$ by induction on k.

For $k = 1$, set $a_1 = a$. Then (1.1) is satisfied and there is no other choice for a_1.

Let $k > 1$. We first prove the uniqueness. If there is a representation as in (1.1), then $0 \le a - a_1 g^{k-1} < g^{k-1}$ and therefore $0 \le a/g^{k-1} - a_1 < 1$. Therefore, a_1 is the integral quotient of a divided by g^{k-1} and is hence uniquely determined. Set $a' = a - a_1 g^{k-1} = \sum_{i=2}^{k} a_i g^{k-i}$. Either we have $a' = 0$, in which case $a_i = 0$, $2 \le i \le n$ or $a' = \sum_{i=2}^{k} a_i g^{k-i}$ is the uniquely determined representation of a' by the induction hypothesis. It is also clear that a representation (1.1) exists. We only need to set $a_1 = \lfloor a/g^{k-1} \rfloor$ and to take the other coefficients from the representation $a' = a - a_1 g^{k-1}$. □

Definition 1.3.4
The sequence $(a_1, \ldots, a_k)$ from Theorem 1.3.3 is called the *g-adic expansion* of a. Its elements are called *digits*. Its *length* is $k = \lfloor \log_g a \rfloor + 1$. If $g = 2$, the sequence is called the *binary expansion* of a. If $g = 16$, then the sequence is called the *hexadecimal expansion* of a.

Instead of $(a_1, \ldots, a_k)$, we also write $a_1 a_2 \ldots a_k$.

Example 1.3.5
The sequence 10101 is the binary expansion of $2^4 + 2^2 + 2^0 = 21$. When writing the hexadecimal expansion, we use instead of the digits $10, 11, \ldots, 15$ the letters A, B, C, D, E, F, so A1C is the hexadecimal expansion of $10 * 16^2 + 16 + 12 = 2588$.

Theorem 1.3.3 contains a procedure for computing the g-adic expansion of a positive integer. This is applied in the next example.

Example 1.3.6
We determine the binary expansion of 105. Since $64 = 2^6 < 105 < 128 = 2^7$, it is of length 7. We find the following: $a_1 = \lfloor 105/64 \rfloor = 1$; $105 - 64 = 41$; $a_2 = \lfloor 41/32 \rfloor = 1$; $41 - 32 = 9$; $a_3 = \lfloor 9/16 \rfloor = 0$; $a_4 = \lfloor 9/8 \rfloor = 1$; $9 - 8 = 1$; $a_5 = a_6 = 0$; $a_7 = 1$. Hence, the binary expansion of 105 is the sequence 1101001.

The transformation of hexadecimal expansions to binary expansions is particularly simple. Let $(h_1, h_2, \ldots, h_k)$ be the hexadecimal expansion of a positive integer n. For $1 \leq i \leq k$, let $(b_{1,i}, b_{2,i}, b_{3,i}, b_{4,i})$ be the bit-string of length 4, which represents h_i (i.e., $h_i = b_{1,i}2^3 + b_{2,i}2^2 + b_{3,i}2 + b_{4,i}$). Then $(b_{1,1}, b_{2,1}, b_{3,1}, b_{4,1}, b_{1,2}, \ldots, b_{4,k})$ is the binary expansion of n.

Example 1.3.7
Consider the hexadecimal number $n = 6EF$. The length 4 normalized binary expansions of the digits are $6 = 0110$, $E = 1110$, $F = 1111$. Therefore, 011011101111 is the binary expansion of n.

The length of the binary expansion of a positive integer is also referred to as its *binary length*. The binary length of 0 is defined to be 1. The binary length of an integer is defined to be the binary length of its absolute value. It is denoted by size (a) or size a.

1.4 O- and Ω-Notation

When designing a cryptographic algorithm, it is necessary to estimate how much computing time and how much storage it requires. To simplify such estimates, we introduce the O- and the Ω-notation.

Let k be a positive integer, $X, Y \subset \mathbb{N}^k$ and $f : X \to \mathbb{R}_{\geq 0}$, $g : Y \to \mathbb{R}_{\geq 0}$ functions. We write $f = O(g)$ if there are positive integers B and C such that for all $(n_1, \ldots, n_k) \in \mathbb{N}^k$ with $n_i > B$, $1 \leq i \leq k$ the following is true:

1. $(n_1, \ldots, n_k) \in X \cap Y$; that is, $f(n_1, \ldots, n_k)$ and $g(n_1, \ldots, n_k)$ are defined,
2. $f(n_1, \ldots, n_k) \leq Cg(n_1, \ldots, n_k)$.

This means that almost always $f(n_1, \ldots, n_k) \leq Cg(n_1, \ldots, n_k)$. We also write $g = \Omega(f)$. If g is constant, then we write $f = O(1)$.

Example 1.4.1
We have $2n^2 + n + 1 = O(n^2)$ because $2n^2 + n + 1 \leq 4n^2$ for all $n \geq 1$. Also, $2n^2 + n + 1 = \Omega(n^2)$ because $2n^2 + n + 1 \geq 2n^2$ for all $n \geq 1$.

Example 1.4.2
If g is an integer, $g > 2$, and if $f(n)$ denotes the length of the g-adic expansion of a positive integer n, then $f(n) = O(\log n)$, where $\log n$ is the natural logarithm of n. In fact, this length is $\lfloor \log_g n \rfloor + 1 \leq \log_g n + 1 = \log n / \log g + 1$. If $n > 3$, then $\log n > 1$ and therefore $\log n / \log g + 1 < (1/\log g + 1) \log n$.

1.5 Cost of Addition, Multiplication, and Division with Remainder

In many cryptographic applications, multi-precision integers must be added, multiplied, and divided with remainder. To estimate the running time of such applications, we must study how long such operations take. To do so, one has to choose a model of computation that is as similar as possible to real computers. This is described in detail in [3] and [4]. Here, we only use a naive model, which, however, yields reasonable estimates.

Let a and b be positive integers, which are given by their binary representations. Let m be the binary length of a and let n be the binary length of m. To compute $a + b$, we use the school method, which adds bit by bit with carry.

Example 1.5.1
Let $a = 10101$, $b = 111$. We compute $a + b$.

	1	0	1	0	1
+			1	1	1
carry		1	1	1	
	1	1	1	0	0

We assume that the addition of two bits takes time $O(1)$. Then the whole addition requires time $O(\max\{m, n\})$. Analogously, one can show that the difference $b-a$ can be computed in time $O(\max\{m, n\})$. This implies that the addition of two integers a and b with binary lengths m and n takes time $O(\max\{m, n\})$.

We use the school method also for multiplication.

Example 1.5.2
Let $a = 10101$, $b = 101$. We compute $a * b$.

	1	0	1	0	1	*	1	0	1
					1	0	1	0	1
+			1	0	1	0	1		
carry				1		1			
			1	1	0	1	0	0	1

We scan b from right to left. For each 1, we write a such that the rightmost bit of a is below the current 1. Then this a is added to the previous result. Any such addition takes time $O(m)$, and $O(n)$ additions are necessary. The computation takes time $O(mn)$. In [3], the algorithm of Schönhage and Strassen is explained, which can multiply two n-bit numbers in time $O(n \log n \log\log n)$. In practice, this algorithm is less efficient than the school method for operands that have fewer than 10,000 bits.

We also use the school method to divide a by b with remainder.

Example 1.5.3
Let $a = 10101$, $b = 101$. We divide a with remainder by b.

```
1 0 1 0 1  =  1 0 1  *  1 0 0  +  1
1 0 1
  0 0 0
  0 0 0
    0 0 1
    0 0 0
      1
```

When analyzing the algorithm, we see the following. Let k be the binary length of the quotient. Then one has to subtract at most k times two numbers of binary length $\leq n + 1$. This takes time

$O(kn)$. We therefore obtain the following bounds, which will be used henceforth:

Let a and b be integers.

1. Adding a and b requires time $O(\max\{\text{size}\, a, \text{size}\, b\})$.
2. Multiplying a and b requires time $O((\text{size}\, a)(\text{size}\, b))$.
3. Dividing a with remainder by b requires time $O((\text{size}\, b)(\text{size}\, q))$, where q is the quotient.

All algorithms use space $O(\text{size}\, a + \text{size}\, b)$.

1.6 Polynomial Time

When analyzing a cryptographic algorithm, we must show that it works efficiently but is difficult to break. We make the notion of "efficiency" more precise.

Suppose an algorithm receives as input integers $z_1, \ldots, z_n$. We say that the algorithm has *polynomial running time* if there are nonnegative integers $e_1, \ldots, e_n$ such that the running time of the algorithm is

$$O((\text{size}\, z_1)^{e_1}(\text{size}\, z_2)^{e_2} \cdots (\text{size}\, z_n)^{e_n}).$$

An algorithm is considered to be efficient if it has polynomial running time. Observe, however, that in order for the algorithm to be efficient in practice, the exponents e_i and the O-constant must be small.

1.7 Greatest Common Divisor

We define the greatest common divisor of two integers.

Definition 1.7.1
A *common divisor* of a and b is an integer that divides both a and b.

Theorem 1.7.2
Among all common divisors of two integers a and b, which are not both zero, there is exactly one greatest (with respect to $\leq$). It is called the greatest common divisor (gcd) of a and b.

Proof. Let $a \neq 0$. By Theorem 1.2.3, all divisors of a are bounded by $|a|$. Therefore, among the common divisors of a and b there is a unique greatest. □

For completeness, we set the greatest common divisor of 0 and 0 to 0 (i.e., $\gcd(0, 0) = 0$). Hence, the greatest common divisor of two numbers is never negative.

Example 1.7.3
The greatest common divisor of 18 and 30 is 6. The greatest common divisor of -10 and 20 is 10. The greatest common divisor of -20 and -14 is 2. The greatest common divisor of 12 and 0 is 12.

The greatest common divisor of integers $a_1, \ldots, a_k$, $k \geq 1$ is defined as follows. If at least one of the a_i is nonzero, then $\gcd(a_1, \ldots, a_k)$ is the greatest positive integer that divides all the a_i. If all the a_i are zero, then we set $\gcd(a_1, \ldots, a_k) = 0$.

Next, we present an important way of representing a greatest common divisor. We need the following notation.

If $\alpha_1, \ldots, \alpha_k$ are real numbers, then we write

$$\alpha_1\mathbb{Z} + \ldots + \alpha_k\mathbb{Z} = \{\alpha_1 z_1 + \ldots + \alpha_k z_k : z_i \in \mathbb{Z}, 1 \leq i \leq k\}.$$

This is the set of all *integer linear combinations* of the a_i.

Example 1.7.4
The set of all integer linear combinations of 3 and 4 is $3\mathbb{Z} + 4\mathbb{Z}$. It contains $1 = 3 * (-1) + 4$. It therefore also contains all integer multiples of 1. Hence, this set is $\mathbb{Z}$.

The next theorem shows that the result in the previous example was not an accident.

Theorem 1.7.5
The set of all integer linear combinations of a and b is the set of all integer multiples of $\gcd(a, b)$*; i.e.,*

$$a\mathbb{Z} + b\mathbb{Z} = \gcd(a, b)\mathbb{Z}.$$

Proof. For $a = b = 0$, the assertion is obviously correct, so let a or b be nonzero.

Set

$$I = a\mathbb{Z} + b\mathbb{Z}.$$

Let g be the smallest positive integer in I. We claim that $I = g\mathbb{Z}$. To see this, choose a nonzero element c in I. We must show that $c = qg$ for some q. By Theorem 1.2.4, there are q, r with $c = qg + r$ and $0 \leq r < g$. Therefore, $r = c - qg$ belongs to I. But since g is the smallest positive integer in I, we must have $r = 0$ and $c = qg$.

It remains to be shown that $g = \gcd(a, b)$. Since $a, b \in I$, it follows from $I = g\mathbb{Z}$ that g is a common divisor of a and b. Moreover, since $g \in I$ there are x, y with $g = xa + yb$. Therefore, if d is a common divisor of a and b, then d is also a divisor of g. Theorem 1.2.3 implies $|d| \leq g$. This shows that $g = \gcd(a, b)$. □

We could have obtained the result of Example 1.7.4 directly from Theorem 1.7.5. Since $\gcd(3, 4) = 1$, it follows that $3\mathbb{Z}+4\mathbb{Z} = 1\mathbb{Z} = \mathbb{Z}$.

Theorem 1.7.5 has important implications.

Corollary 1.7.6
For all a, b, n the equation $ax + by = n$ is solvable in integers x and y if and only if $\gcd(a, b)$ *divides n.*

Proof. If there are x and y with $n = ax + by$, then $n \in a\mathbb{Z} + b\mathbb{Z}$ and by Theorem 1.7.5 we have $n \in \gcd(a, b)\mathbb{Z}$, which implies that n is a multiple of $\gcd(a, b)$.

Conversely, if n is a multiple of $\gcd(a, b)$, then n is an element of $\gcd(a, b)\mathbb{Z}$. It follows from Theorem 1.7.5 that $n \in a\mathbb{Z} + b\mathbb{Z}$. Therefore, there are integers x and y with $n = ax + by$. □

Corollary 1.7.6 tells us that the equation

$$3x + 4y = 123$$

has a solution, because $\gcd(3, 4) = 1$.

Corollary 1.7.7
There are integers x and y with $ax + by = \gcd(a, b)$.

Proof. Since $\gcd(a, b)$ divides itself, the assertion follows immediately from Corollary 1.7.6. □

We present another useful characterization of the greatest common divisor.

Corollary 1.7.8
There is exactly one nonnegative common divisor of a and b, which is divisible by all other common divisors of a and b, namely the greatest common divisor of a and b.

Proof. The greatest common divisor of a and b is a nonnegative common divisor of a and b. Moreover, by Corollary 1.7.7 there are integers x and y with $ax + by = \gcd(a, b)$. Therefore, every common divisor of a and b is a divisor of $\gcd(a, b)$. This shows that there exists a common divisor of a and b that is divisible by any common divisor of a and b.

Conversely, let g be a nonnegative divisor of a and b that is divisible by every common divisor of a and b. If $a = b = 0$, then $g = 0$ since 0 is only divisible by 0. If a or b is nonzero, then by Theorem 1.2.3 every common divisor of a and b is $\leq g$. Therefore, $g = \gcd(a, b)$. □

The question remains how to compute $\gcd(a, b)$ and integers x and y with $ax + by = \gcd(a, b)$. The fact that both problems admit efficient solutions is crucial for many cryptographic systems. In the next sections we present and analyze the euclidean algorithm, which solves both problems.

1.8 Euclidean Algorithm

The euclidean algorithm determines the greatest common divisor of two integers very efficiently. It is based on the following theorem.

Theorem 1.8.1
1. *If $b = 0$, then* $\gcd(a, b) = |a|$.
2. *If $b \neq 0$, then* $\gcd(a, b) = \gcd(|b|, a \bmod |b|)$.


```
euclid(int a, int b, int gcd)
begin
  int r
  a = |a|
  b = |b|
  while (b != 0)
    r = a%b
    a = b
    b = r
  end while
  gcd = a
end
```

FIGURE 1.1 The euclidean algorithm

Proof. The first assertion is obviously correct. We prove the second assertion. By Theorem 1.2.4, there is an integer q with $a = q|b| + (a \bmod |b|)$. Therefore, the greatest common divisor of a and b divides the greatest common divisor of $|b|$ and $a \bmod |b|$ and vice versa. Since both greatest common divisors are nonnegative, the assertion follows from Theorem 1.2.3. □

We explain the euclidean algorithm in an example.

Example 1.8.2
We want to compute $\gcd(100, 35)$. From Theorem 1.8.1, we obtain $\gcd(100, 35) = \gcd(35, 100 \bmod 35) = \gcd(35, 30) = \gcd(30, 5) = \gcd(5, 0) = 5$.

First, the euclidean algorithm replaces a by $|a|$ and b by $|b|$. This has no effect in our example. As long as b is nonzero, the algorithm replaces a by b and b by $a \bmod b$. As soon as $b = 0$, the algorithm returns a. Figure 1.1 shows the pseudocode for the euclidean algorithm.

We prove the correctness of the euclidean algorithm.

Theorem 1.8.3
The euclidean algorithm computes the greatest common divisor of a and b.

Proof. To prove that the euclidean algorithm terminates and yields $\gcd(a, b)$, we introduce some notation that will also be used later. We set

$$r_0 = |a|, r_1 = |b| \tag{1.2}$$

and for $k \geq 1$ and $r_k \neq 0$

$$r_{k+1} = r_{k-1} \bmod r_k. \tag{1.3}$$

Then $r_2, r_3, \ldots$ is the sequence of remainders that are computed in the `while`-loop of the euclidean algorithm. Also, after the kth iteration of the `while`-loop, we have

$$a = r_k, \quad b = r_{k+1}.$$

It follows from Theorem 1.8.1 that the greatest common divisor of a and b is not changed in the algorithm, so we only need to prove that there is k such that $r_k = 0$. But this follows from the fact that by (1.3) the sequence $(r_k)_{k\geq 1}$ is strictly decreasing. This concludes the correctness proof for the euclidean algorithm. □

The euclidean algorithm computes $\gcd(a, b)$ very efficiently. This is important for cryptographic applications. To prove the efficiency, we estimate the number of iterations required by the euclidean algorithm. For simplicity, we assume

$$a > b > 0.$$

This is no restriction, since the euclidean algorithm requires one step to determine $\gcd(a, b)$ (if $b = 0$) or to produce this situation.

Let r_n be the last nonzero remainder in the sequence (r_k). Then n is the number of iterations, which the euclidean algorithm requires to compute $\gcd(a, b)$. Furthermore, let

$$q_k = \lfloor r_{k-1}/r_k \rfloor, \quad 1 \leq k \leq n. \tag{1.4}$$

Then q_k is the quotient of r_{k-1} divided by r_k, and we have

$$r_{k-1} = q_k r_k + r_{k+1}. \tag{1.5}$$

Example 1.8.4
If $a = 100$ and $b = 35$, then we obtain the remainder sequence

k	0	1	2	3	4
r_k	100	35	30	5	0
q_k		2	1	6	

To estimate the number n of iterations, we prove the following auxiliary result. Recall that we have assumed $a > b > 0$.

Lemma 1.8.5
We have $q_k \geq 1$ for $1 \leq k \leq n-1$ and $q_n \geq 2$.

Proof. Since $r_{k-1} > r_k > r_{k+1}$, it follows from (1.5) that $q_k \geq 1$ for $1 \leq k \leq n$. Suppose $q_n = 1$. Then $r_{n-1} = r_n$, and this is impossible because the sequence of remainders is strictly decreasing. Therefore, $q_n \geq 2$. □

Theorem 1.8.6
In the euclidean algorithm, let $a > b > 0$. Also, let $\Theta = (1 + \sqrt{5})/2$. Then the number of iterations in the euclidean algorithm is at most $(\log b)/(\log \Theta) + 1 < 1.441 * \log_2(b) + 1$.

Proof. By Exercise 1.12.19, we may assume that $\gcd(a, b) = r_n = 1$. We prove

$$r_k \geq \Theta^{n-k}, \quad 0 \leq k \leq n. \tag{1.6}$$

Then

$$b = r_1 \geq \Theta^{n-1}.$$

Taking logarithms, we obtain

$$n \leq (\log b)/(\log \Theta) + 1,$$

as asserted.

We now prove (1.6) by induction. First, we have

$$r_n = 1 = \Theta^0$$

and by Lemma 1.8.5

$$r_{n-1} = q_n r_n = q_n \geq 2 > \Theta.$$

Let $n - 2 \geq k \geq 0$, and assume that the assertion is true for $k' > k$. Then Lemma 1.8.5 implies

$$r_k = q_{k+1} r_{k+1} + r_{k+2} \geq r_{k+1} + r_{k+2}$$

$$\geq \Theta^{n-k-1} + \Theta^{n-k-2} = \Theta^{n-k-1}\left(1 + \frac{1}{\Theta}\right) = \Theta^{n-k},$$

so (1.6) and the theorem are proved. □

1.9 Extended Euclidean Algorithm

In the previous section, we have seen how the greatest common divisor of two integers can be computed. Corollary 1.7.7 tells us that there are integers x, y with $\gcd(a, b) = ax + by$. In this section, we extend the euclidean algorithm in such a way that it also determines such coefficients x and y. As in Section 1.8, we denote by $r_0, \ldots, r_{n+1}$ the sequence of remainders and by $q_1, \ldots, q_n$ the sequence of quotients that are computed in the course of the euclidean algorithm.

We now explain the construction of two sequences (x_k) and (y_k), such that $x = (-1)^n x_n$ and $y = (-1)^{n+1} y_n$ are the required coefficients.

We set

$$x_0 = 1, \quad x_1 = 0, \quad y_0 = 0, \quad y_1 = 1.$$

Furthermore, we let

$$x_{k+1} = q_k x_k + x_{k-1}, \quad y_{k+1} = q_k y_k + y_{k-1}, \quad 1 \leq k \leq n. \tag{1.7}$$

We assume that a and b are nonnegative.

Theorem 1.9.1
We have $r_k = (-1)^k x_k a + (-1)^{k+1} y_k b$ *for* $0 \leq k \leq n+1$.

Proof. We note first that

$$r_0 = a = 1 * a - 0 * b = x_0 * a - y_0 * b.$$

Moreover,

$$r_1 = b = -0 * a + 1 * b = -x_1 * a + y_1 * b.$$

Now let $k \geq 2$ and suppose that the assertion is true for all $k' < k$. Then

$r_k = r_{k-2} - q_{k-1} r_{k-1}$

$$\begin{aligned}
&= (-1)^{k-2} x_{k-2} a + (-1)^{k-1} y_{k-2} b - q_{k-1}((-1)^{k-1} x_{k-1} a + (-1)^k y_{k-1} b) \\
&= (-1)^k a(x_{k-2} + q_{k-1} x_{k-1}) + (-1)^{k+1} b(y_{k-2} + q_{k-1} y_{k-1}) \\
&= (-1)^k x_k a + (-1)^{k+1} y_k b,
\end{aligned}$$

so our theorem is proved. □

We see that in particular

$$r_n = (-1)^n x_n a + (-1)^{n+1} y_n b,$$

so we have represented the greatest common divisor of a and b as a linear combination of a and b. The required coefficients are

$$x = (-1)^n x_n \quad y = (-1)^{n+1} y_n.$$

Example 1.9.2
Choose $a = 100$ and $b = 35$. Then the values r_k, q_k, x_k, and y_k are listed in the following table.

k	0	1	2	3	4
r_k	100	35	30	5	0
q_k		2	1	6	
x_k	1	0	1	1	7
y_k	0	1	2	3	20

We find therefore that $n = 3$ and $\gcd(100, 35) = 5 = -1 * 100 + 3 * 35$.

The pseudocode of the extended euclidean algorithm can be found in Figure 1.2.

```
xeuclid(int a, int b,int gcd, int x, int y) {
begin
  int q, r, xx, yy, sign
  int xs[2], ys[2]

  // The coefficients are initialized

  xs[0] = 1 xs[1] = 0
  ys[0] = 0 ys[1] = 1
  sign = 1

  // As long as b != 0, we replace  a by b and b by a%b.
```

```
// We also update the coefficients x and y.

while (b != 0)
  r = a%b
  q = a/b
  a = b
  b = r
  xx = xs[1]
  yy = ys[1]
  xs[1] = q*xs[1] + xs[0]
  ys[1] = q*ys[1] + ys[0]
  xs[0] = xx
  ys[0] = yy
  sign = -sign
end while

// Final computation of the coefficients.

x = sign*xs[0]
y = -sign*ys[0]

  // Determination of gcd(a,b)
  gcd = a
end
```

FIGURE 1.2 The extended euclidean algorithm

1.10 Analysis of the Extended Euclidean Algorithm

First, we estimate the size of the coefficients x and y. We use the matrices

$$E_k = \begin{pmatrix} q_k & 1 \\ 1 & 0 \end{pmatrix}, \quad 1 \le k \le n,$$

$$T_k = \begin{pmatrix} y_k & y_{k-1} \\ x_k & x_{k-1} \end{pmatrix}, \quad 1 \le k \le n+1.$$

We have

$$T_{k+1} = T_k E_k, \quad 1 \le k \le n,$$

and since T_1 is the identity matrix, we have

$$T_{n+1} = E_1 E_2 \cdots E_n.$$

If we set

$$S_k = E_{k+1} E_{k+2} \cdots E_n, \quad 0 \le k \le n,$$

where S_n is the identity matrix, then

$$S_0 = T_{n+1}.$$

We use the matrices S_k to estimate x_n and y_n. If we write

$$S_k = \begin{pmatrix} u_k & v_k \\ u_{k+1} & v_{k+1} \end{pmatrix}, \quad 0 \le k \le n,$$

then because of

$$S_{k-1} = E_k S_k, \quad 1 \le k \le n,$$

we obtain the recursions

$$u_{k-1} = q_k u_k + u_{k+1}, \quad v_{k-1} = q_k v_k + v_{k+1}, \quad 1 \le k \le n. \tag{1.8}$$

The remainders r_k satisfy the same recursion.

Now we estimate the entries v_k of the matrices S_k.

Lemma 1.10.1
We have $0 \le v_k \le r_k/(2\gcd(a,b))$ *for* $0 \le k \le n$.

Proof. Note that $0 = v_n < r_n/(2\gcd(a,b))$, $q_n \ge 2$ by Lemma 1.8.5, and $v_{n-1} = 1$. Therefore, $r_{n-1} = q_n r_n \ge 2\gcd(a,b) \ge 2\gcd(a,b)v_{n-1}$. Suppose that the assertion is true for $k' \ge k$. Then $v_{k-1} = q_k v_k + v_{k+1} \le (q_k r_k + r_{k+1})/(2\gcd(a,b)) = r_{k-1}/(2\gcd(a,b))$, so the asserted estimate is proved. □

From Lemma 1.10.1, we can deduce the estimates for the coefficients x_k and y_k.

Corollary 1.10.2
We have $x_k \le b/(2\gcd(a,b))$ *and* $y_k \le a/(2\gcd(a,b))$ *for* $1 \le k \le n$.

Proof. It follows from $S_0 = T_{n+1}$ that $x_n = v_1$ and $y_n = v_0$. Therefore, we obtain the asserted estimate for $k = n$ from Lemma 1.10.1. But since $(x_k)_{k\ge1}$ and $(y_k)_{k\ge0}$ are increasing sequences, the assertion is proved for $1 \le k \le n$. □

For the coefficients x and y, which are computed by the extended euclidean algorithm, we obtain the following estimate.

Corollary 1.10.3
We have $|x| \le b/(2\gcd(a,b))$ *and* $|y| \le a/(2\gcd(a,b))$.

We are also able to determine the coefficients x_{n+1} and y_{n+1}.

Lemma 1.10.4
We have $x_{n+1} = b/\gcd(a,b)$ *and* $y_{n+1} = a/\gcd(a,b)$.

We leave the proof to the reader.

We will now estimate the running time of the euclidean algorithm. It turns out that this running time is of the same order of magnitude as the running time for multiplying two integers. This is quite surprising because the euclidean algorithm looks much more difficult than the multiplication algorithm.

Theorem 1.10.5
The extended euclidean algorithm uses time $O((\text{size}\,a)(\text{size}\,b))$ *to compute* $\gcd(a,b)$ *including a representation* $\gcd(a,b) = xa + yb$.

Proof. We assume that $a > b > 0$. We have already seen that the euclidean algorithm requires one iteration to compute $\gcd(a,b)$ or to generate this situation. The running time for this one iteration is $O(\text{size}\,(a)\text{size}\,(b))$.

The euclidean algorithm computes the remainder sequence $(r_k)_{2\le k\le n+1}$ and the quotient sequence $(q_k)_{1\le k\le n}$. The number r_{k+1} is the remainder of the division of r_{k-1} by r_k for $1 \le k \le n$. As explained in Section 1.5, the computation of r_{k+1} requires time $O(\text{size}\,(r_k)\text{size}\,(q_k))$, where q_k is the quotient of this division.

We know that $r_k \le b$, hence $\text{size}\,(r_k) \le \text{size}\,(b)$ for $1 \le k \le n+1$. Moreover, we know that $\text{size}\,(q_k) = \log_2(q_k) + 1$ for $1 \le k \le n$.

Therefore, the euclidean algorithm takes time

$$T_1(a,b) = O\left(\text{size}\,(b)\left(n + \sum_{k=1}^{n} \log q_k\right)\right). \tag{1.9}$$

By Theorem 1.8.6, we have

$$n = O(\text{size}\,b). \tag{1.10}$$

Also,

$$\begin{aligned} a = r_0 &= q_1 r_1 + r_2 \ge q_1 r_1 = q_1(q_2 r_2 + r_3) \\ &\ge q_1 q_2 r_2 > \ldots \ge q_1 q_2 \cdots q_n. \end{aligned}$$

This implies

$$\sum_{k=1}^{n} \log q_k = O(\text{size}\,a). \tag{1.11}$$

If we use (1.10) and (1.11) in (1.9), then the running time of the simple euclidean algorithm is proven.

We also estimate the time that the extended euclidean algorithm needs to compute the coefficients x and y. In the first iteration, we have

$$x_2 = q_1 x_1 + x_0 = 1, \quad y_2 = q_1 y_1 + y_0 = q_1.$$

This takes time $O(\text{size}\,(q_1)) = O(\text{size}\,(a))$. Then,

$$x_{k+1} = q_k x_k + x_{k-1}, \quad y_{k+1} = q_k y_k + y_{k-1}$$

is computed for $2 \le k \le n$. By Lemma 1.10.2, we have $x_k, y_k = O(a)$ for $0 \le k \le n$. The time to compute x and y is therefore

$$\begin{aligned} T_2(a,b) &= O\left(\text{size}\,(a)\left(1 + \sum_{k=2}^{n} \text{size}\,(q_k)\right)\right) \\ &= O\left(\text{size}\,(a)\left(n + \sum_{k=2}^{n} \log q_k\right)\right). \end{aligned}$$

As above, it is easy to see that

$$\prod_{k=2}^{n} q_k \le b. \tag{1.12}$$

If this is used in (1.12), then the assertion is proved. □

1.11 Factoring into Primes

A central notion of elementary number theory is that of a prime number. Prime numbers are used in many cryptographic algorithms. In this section, we introduce prime numbers and prove that every positive integer is a product of primes in which the factors are unique up to permutation.

Definition 1.11.1
An integer $p > 1$ is called a *prime number* if it has exactly two positive divisors, namely 1 and p.

Instead of "prime number" we also simply say "prime". The first nine prime numbers are $2, 3, 5, 7, 11, 13, 17, 19, 23$. We denote the set of all primes by $\mathbb{P}$. An integer $a > 1$ that is not a prime is called *composite* . If the prime p divides the integer a, then p is called *prime divisor* of a.

Theorem 1.11.2
Every integer $a > 1$ has a prime divisor.

Proof. The integer a has a divisor that is greater than 1, namely a. Among all divisors of a that are greater than 1, let p be the smallest. Then p must be prime. Otherwise, p would have a divisor b with

$$1 < b < p \leq a.$$

This contradicts the assumption that p is the smallest divisor of a that is greater than 1. □

The following result is crucial for the proof of the decomposition theorem.

Lemma 1.11.3
If a prime number divides the product of two integers, then it divides at least one factor.

Proof. Suppose the prime number p divides ab but not a. Since p is a prime number, we must have $\gcd(a, p) = 1$. By Corollary 1.7.7, there are x, y with $1 = ax + py$. This implies

$$b = abx + pby.$$

Since p divides abx and pby, Theorem 1.2.3 implies that p is a divisor of b. □

Corollary 1.11.4
If a prime number p divides a product $\prod_{i=1}^{k} q_i$ of prime numbers, then p is equal to one of the factors $q_1, q_2, \ldots, q_k$.

Proof. The proof uses induction on k. If $k = 1$, then p is a divisor of q_1 which is greater than 1, hence $p = q_1$. If $k > 1$, then p divides $q_1(q_2 \cdots q_k)$. By Lemma 1.11.3, the prime p divides q_1 or $q_2 \cdots q_k$. Because both products have fewer than k factors, the assertion follows from the induction hypothesis. □

Now we prove the main theorem of elementary number theory.

Theorem 1.11.5
Every integer $a > 1$ can be written as the product of prime numbers. Up to permutation, the factors in this product are uniquely determined.

Proof. The theorem is proved by induction on a. For $a = 2$, the theorem is true. Let $a > 2$. By Theorem 1.11.2, there is a prime divisor p of a. If $a/p = 1$, then $a = p$ and the assertion holds. Let $a/p > 1$. By the induction hypothesis, a/p is a product of primes. Therefore, a is also a product of primes. This proves the existence of the prime factor decomposition of a. We must still show the uniqueness, so let $a = p_1 \cdots p_k$ and $a = q_1 \cdots q_l$ be factorizations of a into prime numbers. By Corollary 1.11.4, the prime p_1 is equal to one of the primes $q_1, \ldots, q_k$. By permuting the q_i, we can make sure that $p_1 = q_1$. But by the induction hypothesis, the factorization of $a/p_1 = a/q_1$ into prime numbers is unique. Hence, $k = l$ and $q_i = p_i$ for $1 \leq i \leq k$ after an appropriate permutation of the q_i. □

The *prime factorization* of an integer a is the representation of $|a|$ as the product of prime numbers. The problem of finding the prime factorization of an integer a is referred to as the integer factorization problem. Efficient algorithms for solving the integer factorization problem are not known. This fact is the basis of the security of the RSA cryptosystem and other important cryptographic schemes. But we have no proof that the integer factorization problem is difficult. For example, if quantum computers can be built, then factoring is possible in polynomial time.

Example 1.11.6
The French mathematician Pierre de Fermat (1601 to 1665) thought that all of the so-called *Fermat numbers*

$$F_i = 2^{2^i} + 1$$

are primes. In fact, $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, and $F_4 = 65537$ are prime numbers. However, in 1732 Euler discovered that $F_5 = 641 * 6700417$ is composite. Both factors in this decomposition are primes. F_6, F_7, F_8, and F_9 are also composite. The factorization of F_6 was found in 1880 by Landry and Le Lasseur. The factorization of F_7 was found in 1970 by Brillhart and Morrison. The factorization of F_8 was computed in 1980 by Brent and Pollard and F_9 was factored in 1990 by Lenstra, Lenstra, Manasse, and Pollard. This shows the the difficulty of the factoring problem. But on the other hand, we also see that there is considerable progress. It took until 1970 to factor the 39-digit number F_7, but only 20 years later the 155-digit number F_9 was factored. The current factorization status for Fermat numbers can be found in `www.prothsearch.net/fermat.html`.

1.12 Exercises

Exercise 1.12.1
Let α be a real number. Show that $\lfloor \alpha \rfloor$ is the uniquely determined integer z with $0 \leq \alpha - z < 1$.

Exercise 1.12.2
Determine the number of divisors of 2^n, $n \in \mathbb{Z}_{\geq 0}$.

Exercise 1.12.3
Determine all divisors of 195.

Exercise 1.12.4
Prove the following modification of division with remainder: If a, b are integers, $b > 0$, then there are uniquely determined integers q and r such that $a = qb + r$ and $-b/2 < r \leq b/2$. Write a program that determines the remainder r.

Exercise 1.12.5
Compute 1243 mod 45 and -1243 mod 45.

Exercise 1.12.6
Find an integer a with $a \bmod 2 = 1$, $a \bmod 3 = 1$, and $a \bmod 5 = 1$.

Exercise 1.12.7
Let m be a positive integer and let a, b be integers. Prove that $a \bmod m = b \bmod m$ if and only if m divides the difference $b - a$.

Exercise 1.12.8
Determine the binary length of the nth Fermat number $2^{2^n} + 1$, $n \in \mathbb{Z}_{\geq 0}$.

Exercise 1.12.9
Determine the binary expansion and the hexadecimal expansion of 225.

Exercise 1.12.10
Write a program that computes the g-adic expansion of a positive integer n for any integer $g > 1$.

Exercise 1.12.11
Let $f(n) = a_d n^d + a_{d-1} n^{d-1} + \ldots + a_0$ be a polynomial with real coefficients and let $a_d > 0$. Prove that $f(n) = O(n^d)$.

Exercise 1.12.12
Let $k \in \mathbb{N}$ and $X \subset \mathbb{N}^k$. Assume that $f, g, F, G : X \to \mathbb{R}_{\geq 0}$ with $f = O(F)$ and $g = O(G)$. Prove that $f \pm g = O(F + G)$ and $fg = O(FG)$.

Exercise 1.12.13
Let $a_1, \ldots, a_k$ be integers. Prove the following assertions:

1. $\gcd(a_1, \ldots, a_k) = \gcd(a_1, \gcd(a_2, \ldots, a_k))$.
2. $a_1\mathbb{Z} + \ldots + a_k\mathbb{Z} = \gcd(a_1, \ldots, a_k)\mathbb{Z}$.
3. The equation $x_1a_1 + \ldots + x_ka_k = n$ has integer solutions $x_1, \ldots, x_k$ if $\gcd(a_1, \ldots, a_k)$ divides n.
4. There are integers $x_1, \ldots, x_k$ with $a_1x_1 + \ldots + a_kx_k = \gcd(a_1, \ldots, a_k)$.
5. The greatest common divisor of $a_1, \ldots, a_k$ is the uniquely determined nonnegative common divisor of $a_1, \ldots, a_k$ which is divisible by all common divisors of $a_1, \ldots, a_k$.

Exercise 1.12.14
Show that the euclidean algorithm also works if the division with remainder is modified as in Exercise 1.12.4.

Exercise 1.12.15
Use the euclidean algorithm to compute gcd(235, 124) including its representation.

Exercise 1.12.16
Use the modified euclidean algorithm from Exercise 1.12.14 to compute gcd(235, 124) including its representation. Compare this computation with the computation in Exercise 1.12.15.

Exercise 1.12.17
Prove Lemma 1.10.4.

Exercise 1.12.18
Let $a > b > 0$. Prove that the modified euclidean algorithm from Exercise 1.12.14 requires $O(\log b)$ iterations to compute $\gcd(a, b)$.

Exercise 1.12.19
Let $a > b > 0$. Prove that the number of iterations that the euclidean algorithm needs to compute $\gcd(a, b)$ depends only on the ratio a/b.

Exercise 1.12.20
Find a sequence $(a_i)_{i\geq 1}$ of positive integers such that the euclidean algorithm needs exactly i iterations to compute $\gcd(a_{i+1}, a_i)$.

Exercise 1.12.21
Prove that $\gcd(a, m) = 1$ and $\gcd(b, m) = 1$ implies $\gcd(ab, m) = 1$.

Exercise 1.12.22
Compute the prime factorization of 37800.

Exercise 1.12.23
Prove that a composite integer n, $n > 1$ has a prime divisor p with $p \leq \sqrt{n}$.

Exercise 1.12.24
The *sieve of Eratosthenes* determines all prime numbers p below a given bound C. It works as follows. Write the list of integers $2, 3, 4, 5, \ldots, \lfloor C \rfloor$. Then iterate the following procedure for $i =$

$2, 3, \ldots, \lfloor \sqrt{C} \rfloor$. If i is still in the list, delete all proper multiples $2i, 3i, 4i, \ldots$ in the list. The numbers remaining in the list are the prime numbers $\leq C$. Prove the correctness of this algorithm. Write a program that implements it.