
1 Operational notes







































2 Document updated on **August 8, 2022**.

3 The following colors are **not** part of the final product, but serve as highlights in the edit-
4 ing/review process:

- 5 • text that needs attention from the Subject Matter Experts: Mirco, Anna,& Jan
- 6 • terms that have not yet been defined in the book
- 7 • things that need to be checked only at the very final typesetting stage
8 (and it doesn't make sense to do them before)
- 9 • text that needs advice from the communications/marketing team: Aaron & Shane
- 10 • text that needs to be completed or otherwise edited (by Sylvia)

11 NB: This PDF only includes the following chapter(s): Algebra.

12 Todo list

13	 Clarinet	5
14	 zero-knowledge proofs	5
15	 played with	6
16	 Update reference when content is finalized	6
17	 methatical	6
18	 numerical	6
19	 a list of additional exercises	6
20	 think about them	6
21	 Pluralize chapter title	13
22	 check if this is already introduced in intro	13
23	 unify addressing the reader	13
24	 formality of addressing the reader	13
25	 Move content on binary representation	14
26	 simplify Sage ex.	14
27	 To see that	15
28	 let's	15
29	 "themselves" is more common?	15
30	 you	15
31	 a bit overused in the text?	16
32	 readers	18
33	 Commonwealth countries	18
34	 Add explanation	18
35	 unify with example 4	19
36	 check additional explanation	20
37	 check if extended table is understandable	20
38	 the interested reader	21
39	 SB: let's separate these two steps in the equivalence below	23
40	 check reference	24
41	 Readers who	24
42	 check algorithm floating	25
43	 add more explanation	25
44	 one chooses	26
45	 type	27
46	 let's	27
47	 modulo/ modulus/ modular? unify throughout	28
48	 expand on this	29
49	 way	29
50	 add explanation on why this is important	32
51	why is this ref here?	32

52	what does this imply?	32
53	check reference	33
54	why is this ref here?	33
55	check reference	34
56	the task could be defined more clearly	35
57	subtrahend	35
58	minuend	35
59	algorithm-floating	37
60	add reference	37
61	check algorithm floating	40
62	Sylvia: I would like to have a separate counter for definitions	43
63	complicated	45
64	and/or	45
65	unify title of Alg with text	46
66	move 3.4 here	46
67	check algorithm floating	46
68	doesn't	46
69	should be 2?	47
70	images	48
71	b	50
72	we say that	50
73	mention a few examples	51
74	a few examples?	55
75	pseudorandom function family	56
76	Check change of wording	65
77	jubjub	78
78	Check if following Alg is floated too far	91
79	TODO:Rewrite intro together with Sven	103
80	check floating of algorithm	109
81	add references	109
82	check if the algorithm is floated properly	111
83	circuit	114
84	signature schemes	114
85	add reference	114
86	add references	114
87	add reference	115
88	algebraic closures	115
89	check reference	116
90	check reference	116
91	disambiguate	117
92	check reference	119
93	add reference	121
94	check reference	121
95	check reference	121
96	check reference	121
97	add reference	122
98	check reference	122
99	check reference	123

100	check reference	123
101	what does this mean? Maybe just delete it	124
102	write up this part	125
103	add reference	125
104	check reference	125
105	cyclotomic polynomial	125
106	Pholaard-rho attack	126
107	todo	126
108	why? Because in this book elliptic curves are only defined for fields of chracteristic > 3	126
109	check reference	126
110	check reference	126
111	what does this mean?	126
112	add reference	126
113	add reference	127
114	check reference	127
115	check reference	127
116	add reference	128
117	add exercise	128
118	check reference	129
119	add reference	129
120	add reference	129
121	add reference	129
122	check reference	130
123	check reference	130
124	add reference	130
125	add reference	131
126	add reference	132
127	check reference	132
128	add reference	132
129	add reference	132
130	finish writing this up	132
131	add reference	132
132	correct computations	133
133	fill in missing parts	133
134	add reference	133
135	check equation	133
136	Chapter 1?	134
137	add reference	136
138	add reference	137
139	jubjub	137
140	add reference	139
141	Schur/Hadamard product	142
142	add reference	142
143	add references to these languages?	166
144	can we rotate this by 90° ? Good question. IDK	181
145	check reference	196

146

MoonMath manual

147

TechnoBob and the Least Scruples crew

148

August 8, 2022

Contents

150	1 Introduction	5
151	1.1 Aims and target audience	5
152	1.2 The Zoo of Zero-Knowledge Proofs	7
153	To Do List	9
154	Points to cover while writing	9
155	2 Preliminaries	10
156	2.1 Preface and Acknowledgements	10
157	2.2 Purpose of the book	10
158	2.3 How to read this book	11
159	2.4 Cryptological Systems	11
160	2.5 SNARKS	11
161	2.6 complexity theory	11
162	2.6.1 Runtime complexity	11
163	2.7 Software Used in This Book	12
164	2.7.1 Sagemath	12
165	3 Arithmetics	13
166	3.1 Introduction	13
167	3.2 Integer arithmetic	13
168	3.2.1 Integers, natural numbers and rational numbers	13
169	3.2.2 Euclidean Division	16
170	3.2.3 The Extended Euclidean Algorithm	19
171	3.2.4 Coprime Integers	21
172	3.3 Modular arithmetic	21
173	3.3.1 Congruence	21
174	3.3.2 Computational Rules	22
175	3.3.3 The Chinese Remainder Theorem	25
176	3.3.4 Remainder Class Representation	26
177	3.3.5 Modular Inverses	28
178	3.4 Polynomial arithmetic	31
179	3.4.1 Polynomial arithmetic	35
180	3.4.2 Euclidean Division with polynomials	36
181	3.4.3 Prime Factors	39
182	3.4.4 Lagrange interpolation	40

183	4 Algebra	43
184	4.1 Commutative Groups	43
185	4.1.1 Finite groups	45
186	4.1.2 Generators	45
187	4.1.3 The exponential map	46
188	4.1.4 Factor Groups	48
189	4.1.5 Pairings	49
190	4.1.6 Cryptographic Groups	50
191	4.1.6.1 The discrete logarithm assumption	51
192	4.1.6.2 The decisional Diffie–Hellman assumption	51
193	4.1.6.3 The computational Diffie–Hellman assumption	52
194	4.1.7 Hashing to Groups	52
195	4.1.8 Hash functions	52
196	4.1.9 Hashing to cyclic groups	55
197	4.1.10 Pedersen Hashes	56
198	4.1.11 Pseudorandom Function Families in DDH-secure groups	57
199	4.2 Commutative Rings	57
200	4.2.1 Hashing into Modular Arithmetic	61
201	4.3 Fields	64
202	4.3.1 Prime fields	66
203	4.3.2 Square Roots	68
204	4.3.2.1 Hashing into prime fields	69
205	4.3.3 Prime Field Extensions	69
206	4.4 Projective Planes	73
207	5 Elliptic Curves	75
208	5.1 Short Weierstrass Curves	75
209	5.1.1 Affine Short Weierstrass form	76
210	Isomorphic affine short Weierstrass curves	80
211	Affine compressed representation	81
212	5.1.2 Affine Group Law	82
213	Scalar multiplication	86
214	Logarithmic Ordering	86
215	5.1.3 Projective short Weierstrass form	89
216	Projective Group law	91
217	Coordinate Transformations	91
218	5.2 Montgomery Curves	93
219	Affine Montgomery coordinate transformation	95
220	5.2.1 Montgomery group law	97
221	5.3 Twisted Edwards Curves	98
222	5.3.1 Twisted Edwards group law	100
223	5.4 Elliptic Curve Pairings	101
224	Embedding Degrees	101
225	Elliptic Curves over extension fields	103
226	Full torsion groups	104
227	Pairing groups	107
228	The Weil pairing	109
229	5.5 Hashing to Curves	111

230		Try-and-increment hash functions	111
231	5.6	Constructing elliptic curves	114
232		The Trace of Frobenius	114
233		The j -invariant	115
234		The Complex Multiplication Method	116
235	5.6.1	The <i>BLS6_6</i> pen-and-paper curve	125
236		Hashing to pairing groups	132
237	6	Statements	134
238	6.1	Formal Languages	134
239		Decision Functions	135
240		Instance and Witness	138
241		Modularity	141
242	6.2	Statement Representations	142
243	6.2.1	Rank-1 Quadratic Constraint Systems	142
244		R1CS representation	142
245		R1CS Satisfiability	145
246		Modularity	146
247	6.2.2	Algebraic Circuits	146
248		Algebraic circuit representation	147
249		Circuit Execution	152
250		Circuit Satisfiability	153
251		Associated Constraint Systems	154
252	6.2.3	Quadratic Arithmetic Programs	159
253		QAP representation	160
254		QAP Satisfiability	162
255	7	Circuit Compilers	166
256	7.1	A Pen-and-Paper Language	166
257	7.1.1	The Grammar	166
258	7.1.2	The Execution Phases	168
259		The Setup Phase	168
260		The Prover Phase	170
261	7.2	Common Programing concepts	170
262	7.2.1	Primitive Types	170
263		The base-field type	171
264		The Subtraction Constraint System	174
265		The Inversion Constraint System	175
266		The Division Constraint System	176
267		The boolean Type	177
268		The boolean Constraint System	177
269		The AND operator constraint system	178
270		The OR operator constraint system	179
271		The NOT operator constraint system	179
272		Modularity	180
273		Arrays	183
274		The Unsigned Integer Type	184
275		The uN Constraint System	184

276		The Unsigned Integer Operators	185
277	7.2.2	Control Flow	186
278		The Conditional Assignment	186
279		Loops	189
280	7.2.3	Binary Field Representations	190
281	7.2.4	Cryptographic Primitives	191
282		Twisted Edwards curves	191
283		Twisted Edwards curve constraints	191
284		Twisted Edwards curve addition	192
285	8	Zero Knowledge Protocols	194
286	8.1	Proof Systems	194
287	8.2	The “Groth16” Protocol	196
288		The Setup Phase	197
289		The Prover Phase	202
290		The Verification Phase	205
291		Proof Simulation	207
292	9	Exercises and Solutions	211

Chapter 3

Arithmetics

S: This chapter talks about different types of arithmetic, so I suggest using “Arithmetics” as the chapter title.

Pluralize chapter title

3.1 Introduction

The goal of this chapter is to bring a reader with only basic school-level algebra up to speed in arithmetics. We start with a brief recapitulation of basic integer arithmetics, discussing long division, the greatest common divisor and Euclidean Division. After that, we introduce modular arithmetics as **the most important** skill to compute our **pen-and-paper examples**. We then introduce polynomials, compute their analogs to integer arithmetics and introduce the important concept of Lagrange interpolation.

check if this is already introduced in intro

3.2 Integer arithmetic

In a sense, integer arithmetic is at the heart of large parts of modern cryptography. Fortunately, most readers will probably remember integer arithmetic from school. It is, however, important that you can confidently apply those concepts to understand and execute computations in the many pen-and-paper examples that form an integral part of the MoonMath Manual. We will therefore recapitulate basic arithmetic concepts to refresh your memory and fill any knowledge gaps.

unify addressing the reader

formality of addressing the reader

Even though the terms and concepts in this chapter might not appear in the literature on zero-knowledge proofs directly, understanding them is necessary to follow subsequent chapters and beyond: terms like **groups** or **fields** also crop up very frequently in academic papers on zero-knowledge cryptography.

3.2.1 Integers, natural numbers and rational numbers

Integers are also known as **whole numbers**, that is, numbers that can be written without fractional parts. Examples of numbers that are **not** integers are $\frac{2}{3}$, 1.2 and -1280.006 .

Throughout this book, we use the symbol \mathbb{Z} as a shorthand for the set of all **integers**:

$$\mathbb{Z} := \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\} \quad (3.1)$$

If $a \in \mathbb{Z}$ is an integer, then we write $|a|$ for the **absolute value** of a , that is, the non-negative value of a without regard to its sign:

$$|4| = 4 \quad (3.2)$$

$$|-4| = 4 \quad (3.3)$$

We use the symbol \mathbb{N} for the set of all positive integers, usually called the set of **natural numbers**. Furthermore, we use \mathbb{N}_0 for the set of all non-negative integers. This means that \mathbb{N} does not contain the number 0, while \mathbb{N}_0 does:

$$\mathbb{N} := \{1, 2, 3, \dots\} \quad \mathbb{N}_0 := \{0, 1, 2, 3, \dots\}$$

SB: Talking about the binary representation seems way to complex at this stage, and the concepts introduced here are not used for several chapters. Let $n \in \mathbb{N}_0$ be a non-negative integer and (b_0, b_1, \dots, b_k) a string of **bits** $b_j \in \{0, 1\} \subset \mathbb{N}_0$ for some non negative integer $k \in \mathbb{N}$, such that the following equation holds:

$$n = \sum_{j=0}^k b_j \cdot 2^j \quad (3.4)$$

In this case, we call $\text{Bits}(n) := \langle b_0, b_1, \dots, b_k \rangle$ the **binary representation** of n , say that n is a k -bit number and call $k := |n|_2$ the **bit length** of n . It can be shown, that the binary representation of any non negative integer is unique. We call b_0 the **least significant bit** and b_k the **most significant bit** and define the **Hamming weight** of an integer as the number of 1s in its binary representation.

In addition, we use the symbol \mathbb{Q} for the set of all **rational numbers**, which can be represented as the set of all fractions $\frac{n}{m}$, where $n \in \mathbb{Z}$ is an integer and $m \in \mathbb{N}$ is a natural number, such that there is no other fraction $\frac{n'}{m'}$ and natural number $k \in \mathbb{N}$ with $k \neq 1$ and

$$\frac{n}{m} = \frac{k \cdot n'}{k \cdot m'} \quad (3.5)$$

The sets \mathbb{N} , \mathbb{Z} and \mathbb{Q} have a notion of addition and multiplication defined on them. Most of us are probably able to do many integer computations in our head, but this gets more and more difficult as these increase in complexity. We will frequently invoke the SageMath system (2.7.1) for more complicated computations (we define rings and fields later in this book): **SB:** I would delete lines 12-18 from the Sage example below, unnecessarily confusing at this point

```
sage: ZZ # Sage notation for the set of integers
Integer Ring
sage: NN # Sage notation for the set of natural numbers
Non negative integer semiring
sage: QQ # Sage notation for the set of rational numbers
Rational Field
sage: ZZ(5) # Get an element from the set of integers
5
sage: ZZ(5) + ZZ(3)
8
sage: ZZ(5) * NN(3)
```

Move
content
on binary
representation

simplify
Sage ex.

```

590 15 12
591 sage: ZZ.random_element(10**50) 13
592 56499816515073682539212958182454768228311127749717 14
593 sage: ZZ(27713).str(2) # Binary string representation 15
594 110110001000001 16
595 sage: NN(27713).str(2) # Binary string representation 17
596 110110001000001 18
597 sage: ZZ(27713).str(16) # Hexadecimal string representation 19
598 6c41 20

```

A set of numbers of particular interest to us is the set of **prime numbers**, which are natural numbers $p \in \mathbb{N}$ with $p \geq 2$ that are only divisible by themselves and by 1. All prime numbers apart from the number 2 are called **odd** prime numbers. We use \mathbb{P} for the set of all prime numbers and $\mathbb{P}_{\geq 3}$ for the set of all odd prime numbers. The set of prime numbers \mathbb{P} is an infinite set, and it can be ordered according to size. This means that, for any prime number $p \in \mathbb{P}$, one can always find another prime number $p' \in \mathbb{P}$ with $p < p'$. Consequently, there is no largest prime number. Since prime numbers can be ordered by size, we can write them as follows:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, \dots \quad (3.6)$$

As the **fundamental theorem of arithmetic** tells us, prime numbers are, in a certain sense, the basic building blocks from which all other natural numbers are composed. To see that, let $n \in \mathbb{N}$ be any natural number with $n > 1$. Then there are always prime numbers $p_1, p_2, \dots, p_k \in \mathbb{P}$, such that the following equation holds:

To see that

$$n = p_1 \cdot p_2 \cdot \dots \cdot p_k \quad (3.7)$$

This representation is unique for each natural number (except for the order of the **factors** p_1, p_2, \dots, p_k) and is called the **prime factorization** of n .

Example 1 (Prime Factorization). To see what we mean by the prime factorization of a number, let's look at the number $504 \in \mathbb{N}$. To get its prime factors, we can successively divide it by all prime numbers in ascending order starting with 2:

let's

$$504 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot 3 \cdot 7$$

We can double check our findings invoking Sage, which provides an algorithm for factoring natural numbers:

```

615 sage: n = NN(504) 21
616 sage: factor(n) 22
617 2^3 * 3^2 * 7 23

```

The computation from the previous example reveals an important observation: computing the factorization of an integer is computationally expensive, because we have to divide repeatedly by all prime numbers smaller than the number itself until all factors are prime numbers themselves. From this, an important question arises: how fast can we compute the prime factorization of a natural number? This question is the famous **integer factorization problem** and, as far as we know, there is currently no known method that can factor integers much faster than the naive approach of just dividing the given number by all prime numbers in ascending order.

“themselves is more common?”

On the other hand, computing the product of a given set of prime numbers is fast: you just multiply all factors. This simple observation implies that the two processes “prime number

you

multiplication” on the one side and its inverse process “natural number factorization” have very different computational costs. The factorization problem is therefore an example of a **so-called one-way function**: an invertible function that is easy to compute in one direction, but hard to compute in the other direction.¹

a bit overused in the text?

Exercise 1. What is the absolute value of the integers -123 , 27 and 0 ?

Exercise 2. Compute the factorization of 30030 and double check your results using Sage.

Exercise 3. Consider the following equation:

$$4 \cdot x + 21 = 5.$$

Compute the set of all solutions for x under the following alternative assumptions:

1. The equation is defined over the set of natural numbers.

2. The equation is defined over the set of integers.

Exercise 4. Consider the following equation:

$$2x^3 - x^2 - 2x = -1.$$

Compute the set of all solutions x under the following assumptions:

1. The equation is defined over the set of natural numbers.

2. The equation is defined over the set of integers.

3. The equation is defined over the set of rational numbers.

3.2.2 Euclidean Division

As we know from high school mathematics, integers can be added, subtracted and multiplied, and the result of these operations is guaranteed to always be an integer as well. On the contrary, division (in the commonly understood sense) is not defined for integers, as, for example, 7 divided by 3 will not result in an integer. However, it is always possible to divide any two integers if we consider **division with a remainder**. For example, 7 divided by 3 is equal to 2 with a remainder of 1 , since $7 = 2 \cdot 3 + 1$.

This section introduces division with a remainder for integers, usually called **Euclidean Division**. It is an essential technique underlying many concepts in this book. The precise definition is as follows:

Let $a \in \mathbb{Z}$ and $b \in \mathbb{Z}$ be two integers with $b \neq 0$. Then there is always another integer $m \in \mathbb{Z}$ and a natural number $r \in \mathbb{N}$, with $0 \leq r < |b|$ such that the following holds:

$$a = m \cdot b + r \tag{3.8}$$

This decomposition of a given b is called **Euclidean Division**, where a is called the **dividend**, b is called the **divisor**, m is called the **quotient** and r is called the **remainder**. It can be shown that both the quotient and the remainder always exist and are unique, as long as the divisor is different from 0 .

¹It should be pointed out, however, that the American mathematician Peter W. Shor developed an algorithm in 1994, which can calculate the prime factorization of a natural number in polynomial time on a quantum computer. The consequence of this is that cryptosystems, which are based on the prime factor problem, are unsafe as soon as practically usable quantum computers become available.

Notation and Symbols 1. Suppose that the numbers a , b , m and r satisfy equation (3.8). We can then describe the quotient and the remainder of the Euclidean Division as follows:

$$a \operatorname{div} b := m, \quad a \operatorname{mod} b := r \quad (3.9)$$

We also say that an integer a is **divisible** by another integer b if $a \operatorname{mod} b = 0$ holds. In this case, we write $b|a$, and call the integer $a \operatorname{div} b$ the **cofactor** of b in a .

So, in a nutshell, Euclidean Division is the process of dividing one integer by another in a way that produces a quotient and a non-negative remainder, the latter of which is smaller than the absolute value of the divisor.

Example 2. Applying Euclidean Division and the notation defined in 3.9 to the dividend -17 and the divisor 4 , we get the following:

$$-17 \operatorname{div} 4 = -5, \quad -17 \operatorname{mod} 4 = 3 \quad (3.10)$$

$-17 = -5 \cdot 4 + 3$ is the Euclidean Division of -17 by 4 . The remainder, by definition, is a non-negative number. In this case, 4 does not divide -17 , as the remainder is not zero. The truth value of the expression $4|-17$ therefore is **FALSE**. On the other hand, the truth value of $4|12$ is **TRUE**, since 4 divides 12 , as $12 \operatorname{mod} 4 = 0$. If we invoke Sage to do the computation for us, we get the following:

```
sage: ZZ(-17) // ZZ(4) # Integer quotient      24
-5                                              25
sage: ZZ(-17) % ZZ(4) # remainder              26
3                                              27
sage: ZZ(4).divides(ZZ(-17)) # self divides other 28
False                                         29
sage: ZZ(4).divides(ZZ(12))                   30
True                                          31
```

Remark 1. In 3.9, we defined the notation of $a \operatorname{div} b$ and $a \operatorname{mod} b$ in terms of Euclidean Division. It should be noted, however, that many programming languages (like Python and Sage) implement both the operator $(/)$ and the operator $(\%)$ differently. Programmers should be aware of this, as the discrepancy between the mathematical notation and the implementation in programming languages might become the source of subtle bugs in implementations of cryptographic primitives.

To give an example, consider the the dividend -17 and the divisor -4 . Note that, in contrast to the previous example 2, we now have a negative divisor. According to our definition we have the following:

$$-17 \operatorname{div} -4 = 5, \quad -17 \operatorname{mod} -4 = 3 \quad (3.11)$$

$-17 = 5 \cdot (-4) + 3$ is the Euclidean Division of -17 by -4 (the remainder is, by definition, a non-negative number). However, using the operators $(/)$ and $(\%)$ in Sage, we get a different result:

```
sage: ZZ(-17) // ZZ(-4) # Integer quotient      32
4                                              33
sage: ZZ(-17) % ZZ(-4) # remainder              34
-1                                             35
sage: ZZ(-17).quo_rem(ZZ(-4)) # not Euclidean Division 36
```

(4, -1)

Methods to compute Euclidean Division for integers are called **integer division algorithms**. Probably the best known algorithm is the so-called **long division**, which most of us might have learned in school.

In a nutshell, the long division algorithm loops through the digits of the dividend from the left to right, subtracting the largest possible multiple of the divisor (at the digit level) at each stage. The multiples then become the digits of the quotient, and the remainder is the first digit of the dividend.

As long division is the standard method used for pen-and-paper division of multi-digit numbers expressed in decimal notation, we use it throughout this book when we do simple pen-and-paper computations, so readers should become familiar with it. However, instead of defining the algorithm formally, we provide some examples instead, as this will hopefully make the process more clear.

Example 3 (Integer Long Division). To give an example of integer long division algorithm, let's divide the integer $a = 143785$ by the number $b = 17$. Our goal is therefore to find solutions to equation 3.8, that is, we need to find the quotient $m \in \mathbb{Z}$ and the remainder $r \in \mathbb{N}$ such that $143785 = m \cdot 17 + r$. Using a notation that is mostly used in Commonwealth countries, we compute as follows:

$$\begin{array}{r}
 8457 \\
 17 \overline{) 143785} \\
 \underline{136} \\
 77 \\
 \underline{68} \\
 98 \\
 \underline{85} \\
 135 \\
 \underline{119} \\
 16
 \end{array}
 \tag{3.12}$$

We calculated $m = 8457$ and $r = 16$, and, indeed, the equation $143785 = 8457 \cdot 17 + 16$ holds. We can double check this invoking Sage:

```

sage: ZZ(143785).quo_rem(ZZ(17))
(8457, 16)
sage: ZZ(143785) == ZZ(8457)*ZZ(17) + ZZ(16) # check
True

```

Exercise 5 (Integer Long Division). Find an $m \in \mathbb{Z}$ and an $r \in \mathbb{N}$ with $0 \leq r < |b|$ such that $a = m \cdot b + r$ holds for the following pairs:

- $(a, b) = (27, 5)$
- $(a, b) = (27, -5)$
- $(a, b) = (127, 0)$
- $(a, b) = (-1687, 11)$
- $(a, b) = (0, 7)$

In which cases are your solutions unique?

Exercise 6 (Long Division Algorithm). Using the programming language of your choice, write an algorithm that computes integer long division and handles all edge cases properly.

Exercise 7 (Binary Representation). Using the programming language of your choice, write an algorithm that computes the binary representation 3.4 of any non-negative integer.

3.2.3 The Extended Euclidean Algorithm

One of the most critical parts of this book is the modular arithmetic, defined in section 3.3, and its application in the computations of **prime fields**, defined in section 4.3.1. To be able to do computations in modular arithmetic, we have to get familiar with the so-called **Extended Euclidean Algorithm**, used to calculate the **greatest common divisor** (GCD) of integers.

The greatest common divisor of two non-zero integers a and b is defined as the largest non-zero natural number d such that d divides both a and b , that is, $d|a$ as well as $d|b$ are true. We use the notation $\gcd(a, b) := d$ for this number. Since the natural number 1 divides any other integer, 1 is always a common divisor of any two non-zero integers, but it is not necessarily the greatest.

A common method for computing the greatest common divisor is the so-called Euclidean Algorithm. However, since we don't need that algorithm in this book, we will introduce the Extended Euclidean Algorithm, which is a method for calculating the greatest common divisor of two natural numbers a and $b \in \mathbb{N}$, as well as two additional integers $s, t \in \mathbb{Z}$, such that the following equation holds:

$$\gcd(a, b) = s \cdot a + t \cdot b \quad (3.13)$$

The pseudocode in algorithm 1 shows in detail how to calculate the greatest common divisor and the numbers s and t with the Extended Euclidean Algorithm: **In example 4, the computation stops when $r_k = 0$ (at k_4), not when $r_{k-1} = 0$ (which would be k_5). Also the GCD is $r_3 = r_{k-1} = 1$, not r_{k-2} . Same for s and t .**

unify with example 4

Algorithm 1 Extended Euclidean Algorithm

Require: $a, b \in \mathbb{N}$ with $a \geq b$

procedure EXT-EUCLID(a, b)

$r_0 \leftarrow a$ and $r_1 \leftarrow b$

$s_0 \leftarrow 1$ and $s_1 \leftarrow 0$

$t_0 \leftarrow 0$ and $t_1 \leftarrow 1$

$k \leftarrow 2$

while $r_{k-1} \neq 0$ **do**

$q_k \leftarrow r_{k-2} \text{ div } r_{k-1}$

$r_k \leftarrow r_{k-2} \text{ mod } r_{k-1}$

$s_k \leftarrow s_{k-2} - q_k \cdot s_{k-1}$

$t_k \leftarrow t_{k-2} - q_k \cdot t_{k-1}$

$k \leftarrow k + 1$

end while

return $\gcd(a, b) \leftarrow r_{k-2}$, $s \leftarrow s_{k-2}$ and $t \leftarrow t_{k-2}$

end procedure

Ensure: $\gcd(a, b) = s \cdot a + t \cdot b$

The algorithm is simple enough to be used effectively in pen-and-paper examples. It is commonly written as a table where the rows represent the while-loop and the columns

represent the values of the the array r, s and t with index k . The following example provides a simple execution.

Example 4. To illustrate algorithm 1, we apply it to the numbers $a = 12$ and $b = 5$. Since $12, 5 \in \mathbb{N}$ and $12 \geq 5$, all requirements are met, and we compute as follows: [check if extended table is understandable](#)

check additional explanation

check if extended table is understandable

k	r_k	s_k	t_k	
0	12	1	0	
1	5	0	1	
2	2	1	-2	
3	1	-2	5	
4	0			

k	r_k	s_k	t_k	q_k	
0	12	1	0	-	$r_0 \leftarrow a = 12$ $s_0 \leftarrow 1$ $t_0 \leftarrow 0$
1	5	0	1	-	$r_1 \leftarrow b = 5$ $s_1 \leftarrow 0$ $t_1 \leftarrow 1$
2	2	1	-2	2	$r_2 \leftarrow r_0 \bmod r_1 = 12 \bmod 5 = 2$ $s_2 \leftarrow s_0 - q_2 \cdot s_1 = 1 - 2 \cdot 0 = 1$ $t_2 \leftarrow t_0 - q_2 \cdot t_1 = 0 - 2 \cdot 1 = -2$ $q_2 \leftarrow r_0 \operatorname{div} r_1 = 12 \operatorname{div} 5 = 2$
3	1	-2	5	2	$r_3 \leftarrow r_1 \bmod r_2 = 5 \bmod 2 = 1$ $s_3 \leftarrow s_1 - q_3 \cdot s_2 = 0 - 2 \cdot 1 = -2$ $t_3 \leftarrow t_1 - q_3 \cdot t_2 = 1 - 2 \cdot -2 = 5$ $q_3 \leftarrow r_1 \operatorname{div} r_2 = 5 \operatorname{div} 2 = 2$
4	0				$r_4 \leftarrow r_2 \bmod r_3 = 2 \bmod 1 = 0$

From this we can see that the greatest common divisor of 12 and 5 is $\gcd(12, 5) = 1$ and that the equation $1 = (-2) \cdot 12 + 5 \cdot 5$ holds. We can also invoke Sage to double check our findings:

```
sage: ZZ(12).xgcd(ZZ(5)) # (gcd(a,b), s, t)
(1, -2, 5)
```

42

43

Exercise 8 (Extended Euclidean Algorithm). Find integers $s, t \in \mathbb{Z}$ such that $\gcd(a, b) = s \cdot a + t \cdot b$ holds for the following pairs:

- $(a, b) = (45, 10)$
- $(a, b) = (13, 11)$
- $(a, b) = (13, 12)$

Exercise 9 (Towards Prime fields). Let $n \in \mathbb{N}$ be a natural number and p a prime number, such that $n < p$. What is the greatest common divisor $\gcd(p, n)$?

Exercise 10. Find all numbers $k \in \mathbb{N}$ with $0 \leq k \leq 100$ such that $\gcd(100, k) = 5$.

Exercise 11. Show that $\gcd(n, m) = \gcd(n + m, m)$ for all $n, m \in \mathbb{N}$.

3.2.4 Coprime Integers

Coprime integers are integers that do not share a prime number as a factor. As we will see in 3.3, coprime integers are important for our purposes, because, in modular arithmetic, computations that involve coprime numbers are substantially different from computations on non-coprime numbers 3.3.2.

The naive way to decide if two integers are coprime would be to divide both numbers successively by all prime numbers smaller than those numbers, to see if they share a common prime factor. However, two integers are coprime if and only if their greatest common divisor is 1, which is why computing the *gcd* is the preferred method.

Example 5. Consider example 4 again. As we have seen, the greatest common divisor of 12 and 5 is 1. This implies that the integers 12 and 5 are coprime, since they share no divisor other than 1, which is not a prime number.

Exercise 12. Consider exercise 8 again. Which pairs (a, b) from that exercise are coprime?

3.3 Modular arithmetic

Modular arithmetic is a system of integer arithmetic where numbers “wrap around” when reaching a certain value, much like calculations on a clock wrap around whenever the value exceeds the number 12. For example, if the clock shows that it is 11 o’clock, then 20 hours later it will be 7 o’clock, not 31 o’clock. The number 31 has no meaning on a normal clock that shows hours.

The number at which the wrap occurs is called the **modulus**. Modular arithmetic generalizes the clock example to arbitrary moduli, and studies equations and phenomena that arise in this new kind of arithmetic. It is of central importance for understanding most modern cryptographic systems, in large parts because modular arithmetic provides the computational infrastructure for algebraic types that have cryptographically useful examples of one-way functions.

Although modular arithmetic appears very different from ordinary integer arithmetic that we are all familiar with, we encourage [the interested reader to work through the examples and discover that](#), once they get used to the idea that this is a new kind of calculation, it will seem much less daunting.

the in-
terested
reader

3.3.1 Congruence

In what follows, let $n \in \mathbb{N}$ with $n \geq 2$ be a fixed natural number that we will call the **modulus** of our modular arithmetic system. With such an n given, we can then group integers into classes: two integers are in the same class whenever their Euclidean Division (3.2.2) by n will give the same remainder. We two numbers that are in the same class are called **congruent**.

Example 6. If we choose $n = 12$ as in our clock example, then the integers -7 , 5 , 17 and 29 are all congruent with respect to 12, since all of them have the remainder 5 if we perform Euclidean Division on them by 12. Imagining the picture of an analog 12-hour clock, starting at 5 o’clock and adding 12 hours, we are at 5 o’clock again, representing the number 17. Indeed, in many countries, 5:00 in the afternoon is written as 17:00. On the other hand, when we subtract 12 hours, we are at 5 o’clock again, representing the number -7 .

We can formalize this intuition of what congruence should be into a proper definition utilizing Euclidean Division (as explained previously in 3.2). Let $a, b \in \mathbb{Z}$ be two integers, and

$n \in \mathbb{N}$ be a natural number such that $n \geq 2$. The integers a and b are said to be **congruent with respect to the modulus n** if and only if the following equation holds:

$$a \bmod n = b \bmod n \quad (3.14)$$

If, on the other hand, two numbers are not congruent with respect to a given modulus n , we call them **incongruent** w.r.t. n .

In other words, **congruence** is an equation “up to congruence”, which means that the equation only needs to hold if we take the modulus of both sides. In which case we write

$$a \equiv b \pmod{n} \quad (3.15)$$

Exercise 13. Which of the following pairs of numbers are congruent with respect to the modulus 13:

- (5, 19)
- (13, 0)
- (−4, 9)
- (0, 0)

Exercise 14. Find all integers x , such that the congruence $x \equiv 4 \pmod{6}$ is satisfied.

3.3.2 Computational Rules

Having defined the notion of a congruence as an equation “up to a modulus”, a follow-up question is if we can manipulate a congruence similarly to an equation. Indeed, we can almost apply the same substitution rules to a congruency as to an equation, with the main difference being that, for some non-zero integer $k \in \mathbb{Z}$, the congruence $a \equiv b \pmod{n}$ is equivalent to the congruence $k \cdot a \equiv k \cdot b \pmod{n}$ only if k and the modulus n are coprime (see 3.2.4).

Suppose that integers $a_1, a_2, b_1, b_2, k \in \mathbb{Z}$ are given. Then the following arithmetic rules hold for congruences:

- $a_1 \equiv b_1 \pmod{n} \Leftrightarrow a_1 + k \equiv b_1 + k \pmod{n}$ (compatibility with translation)
- $a_1 \equiv b_1 \pmod{n} \Rightarrow k \cdot a_1 \equiv k \cdot b_1 \pmod{n}$ (compatibility with scaling)
- $\gcd(k, n) = 1$ and $k \cdot a_1 \equiv k \cdot b_1 \pmod{n} \Rightarrow a_1 \equiv b_1 \pmod{n}$
- $k \cdot a_1 \equiv k \cdot b_1 \pmod{k \cdot n} \Rightarrow a_1 \equiv b_1 \pmod{n}$
- $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n} \Rightarrow a_1 + a_2 \equiv b_1 + b_2 \pmod{n}$ (compatibility with addition)
- $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n} \Rightarrow a_1 \cdot a_2 \equiv b_1 \cdot b_2 \pmod{n}$ (compatibility with multiplication)

Other rules, such as compatibility with subtraction, follow from the rules above. For example, compatibility with subtraction follows from compatibility with scaling by $k = -1$ and compatibility with addition.

Another property of congruences not found in the traditional arithmetic of integers is **Fermat’s Little Theorem**. Simply put, it states that, in modular arithmetic, every number raised to

the power of a prime number modulus is congruent to the number itself. Or, to be more precise, if $p \in \mathbb{P}$ is a prime number and $k \in \mathbb{Z}$ is an integer, then the following holds:

$$k^p \equiv k \pmod{p} \quad (3.16)$$

If k is coprime to p , then we can divide both sides of this congruence by k and rewrite the expression into the following equivalent form:

$$k^{p-1} \equiv 1 \pmod{p} \quad (3.17)$$

The Sage code below computes examples of Fermat's Little Theorem and highlights the effects of the exponent k being coprime to p (as in the case of 137 and 64) and not coprime to p (as in the case of 1918 and 137):

```

sage: ZZ(137).gcd(ZZ(64))
1
sage: ZZ(64)^ZZ(137) % ZZ(137) == ZZ(64) % ZZ(137)
True
sage: ZZ(64)^ZZ(137-1) % ZZ(137) == ZZ(1) % ZZ(137)
True
sage: ZZ(1918).gcd(ZZ(137))
137
sage: ZZ(1918)^ZZ(137) % ZZ(137) == ZZ(1918) % ZZ(137)
True
sage: ZZ(1918)^ZZ(137-1) % ZZ(137) == ZZ(1) % ZZ(137)
False

```

The following example contains most of the concepts described in this section.

Example 7. Let us solve the following congruence for $x \in \mathbb{Z}$ in modular 6 arithmetic:

$$7 \cdot (2x + 21) + 11 \equiv x - 102 \pmod{6}$$

As many rules for congruences are more or less same as for equations, we can proceed in a similar way as we would if we had an equation to solve. Since both sides of a congruence contain ordinary integers, we can rewrite the left side as follows:

$$7 \cdot (2x + 21) + 11 = 14x + 147 + 11 = 14x + 158$$

We can therefore rewrite the congruence into the equivalent form:

$$14x + 158 \equiv x - 102 \pmod{6}$$

In the next step, we want to shift all instances of x to the left and every other term to the right. So we apply the “compatibility with translation” rules twice. In the first step, we choose $k = -x$, and in a second step, we choose $k = -158$. separate steps 1 and 2 Since “compatibility with translation” transforms a congruence into an equivalent form, the solution set will not change, and we get the following:

$$14x + 158 \equiv x - 102 \pmod{6} \Leftrightarrow$$

$$14x - x + 158 - 158 \equiv x - x - 102 - 158 \pmod{6} \Leftrightarrow$$

$$13x \equiv -260 \pmod{6}$$

SB: let's separate these two steps in the equivalence below

If our congruence was just a regular integer equation, we would divide both sides by 13 to get $x = -20$ as our solution. However, in case of a congruence, we need to make sure that the modulus and the number we want to divide by are coprime to ensure that we get an equivalent expression (see rule 3.17). Consequently, we need to find the greatest common divisor $\gcd(13, 6)$. Since 13 is prime and 6 is not a multiple of 13, we know that $\gcd(13, 6) = 1$, so these numbers are indeed coprime. We therefore compute as follows:

check
reference

$$13x \equiv -260 \pmod{6} \Leftrightarrow x \equiv -20 \pmod{6}$$

Our task now is to find all integers x such that x is congruent to -20 with respect to the modulus 6. In other words, we have to find all x such that the following equation holds:

$$x \bmod 6 = -20 \bmod 6$$

Since $-4 \cdot 6 + 4 = -20$, we know that $-20 \bmod 6 = 4$, and hence we know that $x = 4$ is a solution to this congruence. However, 22 is another solution, since $22 \bmod 6 = 4$ as well. Another solution is -20 . In fact, there are infinitely many solutions given by the following set:

$$\{\dots, -8, -2, 4, 10, 16, \dots\} = \{4 + k \cdot 6 \mid k \in \mathbb{Z}\}$$

Putting all this together, we have shown that every x from the set $\{x = 4 + k \cdot 6 \mid k \in \mathbb{Z}\}$ is a solution to the congruence $7 \cdot (2x + 21) + 11 \equiv x - 102 \pmod{6}$. We double check for two arbitrary numbers from this set, $x = 4$ and $x = 4 + 12 \cdot 6 = 76$ using Sage:

```

873 sage: (ZZ(7) * (ZZ(2) * ZZ(4) + ZZ(21)) + ZZ(11)) % ZZ(6) == (ZZ(
874 4) - ZZ(102)) % ZZ(6)
875
876 True
877
878 sage: (ZZ(7) * (ZZ(2) * ZZ(76) + ZZ(21)) + ZZ(11)) % ZZ(6) == (
879 ZZ(76) - ZZ(102)) % ZZ(6)
880
881 True

```

Readers who had not been familiar with modular arithmetic until now and who might be discouraged by how complicated modular arithmetic seems at this point should keep two things in mind. First, computing congruences in modular arithmetic is not really more complicated than computations in more familiar number systems (e.g. rational numbers), it is just a matter of getting used to it. Second, once we introduce the idea of remainder class representations in 3.3.4, computations become conceptually cleaner and easier to handle.

Readers
who

Exercise 15. Consider the modulus 13 and find all solutions $x \in \mathbb{Z}$ to the following congruence:

$$5x + 4 \equiv 28 + 2x \pmod{13}$$

Exercise 16. Consider the modulus 23 and find all solutions $x \in \mathbb{Z}$ to the following congruence:

$$69x \equiv 5 \pmod{23}$$

Exercise 17. Consider the modulus 23 and find all solutions $x \in \mathbb{Z}$ to the following congruence:

$$69x \equiv 46 \pmod{23}$$

Exercise 18. Let a, b, k be integers, such that $a \equiv b \pmod{n}$ holds. Show $a^k \equiv b^k \pmod{n}$.

Exercise 19. Let a, n be integers, such that a and n are not coprime. For which $b \in \mathbb{Z}$ does the congruence $a \cdot x \equiv b \pmod{n}$ have a solution x and how does the solution set look in that case?

3.3.3 The Chinese Remainder Theorem

We have seen how to solve congruences in modular arithmetic. In this section, we look at how to solve systems of congruences with different moduli using the **Chinese Remainder Theorem**. This states that, for any $k \in \mathbb{N}$ and coprime natural numbers $n_1, \dots, n_k \in \mathbb{N}$, as well as integers $a_1, \dots, a_k \in \mathbb{Z}$, the so-called **simultaneous congruences** (in 3.18 below) have a solution, and all possible solutions of this congruence system are congruent modulo the product $N = n_1 \cdot \dots \cdot n_k$.²

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\dots \\ x &\equiv a_k \pmod{n_k} \end{aligned} \tag{3.18}$$

The following algorithm computes the solution set:

check
algorithm
floating

Algorithm 2 Chinese Remainder Theorem

Require: $k \in \mathbb{Z}$, $j \in \mathbb{N}_0$ and $n_0, \dots, n_{k-1} \in \mathbb{N}$ coprime

procedure CONGRUENCE-SYSTEMS-SOLVER(a_0, \dots, a_{k-1})

$N \leftarrow n_0 \cdot \dots \cdot n_{k-1}$

while $j < k$ **do**

$N_j \leftarrow N/n_j$

$(_, s_j, t_j) \leftarrow EXT-EUCLID(N_j, n_j)$

$$\triangleright 1 = s_j \cdot N_j + t_j \cdot n_j$$

end while

$x' \leftarrow \sum_{j=0}^{k-1} a_j \cdot s_j \cdot N_j$

$x \leftarrow x' \bmod N$

return $\{x + m \cdot N \mid m \in \mathbb{Z}\}$

end procedure

Ensure: $\{x + m \cdot N \mid m \in \mathbb{Z}\}$ is the complete solution set to 3.18.

Example 8. To illustrate how to solve simultaneous congruences using the Chinese Remainder Theorem, let's look at the following system of congruences:

$$\begin{aligned} x &\equiv 4 \pmod{7} \\ x &\equiv 1 \pmod{3} \\ x &\equiv 3 \pmod{5} \\ x &\equiv 0 \pmod{11} \end{aligned}$$

Clearly, all moduli are coprime (since they are all prime numbers). Now we calculate as follows:

$$\begin{aligned} N &= 7 \cdot 3 \cdot 5 \cdot 11 = 1155 \\ N_1 &= 1155/7 = 165 \\ N_2 &= 1155/3 = 385 \\ N_3 &= 1155/5 = 231 \\ N_4 &= 1155/11 = 105 \end{aligned}$$

From this, we calculate with the Extended Euclidean Algorithm:

add more
explana-
tion

²This is the classical Chinese Remainder Theorem as it was already known in ancient China. Under certain circumstances, the theorem can be extended to non-coprime moduli n_1, \dots, n_k but this is beyond the scope of this book. Interested readers should consult XXX [add references](#)

$$\begin{aligned}
1 &= 2 \cdot 165 + -47 \cdot 7 \\
1 &= 1 \cdot 385 + -128 \cdot 3 \\
1 &= 1 \cdot 231 + -46 \cdot 5 \\
1 &= 2 \cdot 105 + -19 \cdot 11
\end{aligned}$$

Consequently, we get $x = 4 \cdot 2 \cdot 165 + 1 \cdot 1 \cdot 385 + 3 \cdot 1 \cdot 231 + 0 \cdot 2 \cdot 105 = 2398$ as one solution. Because $2398 \bmod 1155 = 88$, the set of all solutions is $\{\dots, -2222, -1067, 88, 1243, 2398, \dots\}$. We can invoke Sage's computation of the Chinese Remainder Theorem (CRT) to double check our findings:

```

903 sage: CRT_list([4,1,3,0], [7,3,5,11])
904      88

```

60
61

3.3.4 Remainder Class Representation

As we have seen in various examples before, computing congruences can be cumbersome, and solution sets are large in general. It is therefore advantageous to find some kind of simplification for modular arithmetic.

Fortunately, this is possible and relatively straightforward once we identify each set of numbers that have equal remainders with that remainder itself, and call this set the **remainder class** or **residue class** representation in modulo n arithmetic.

It then follows from the properties of Euclidean Division that there are exactly n different remainder classes for every modulus n , and that integer addition and multiplication can be projected to a new kind of addition and multiplication on those classes.

Informally speaking, the new rules for addition and multiplication are then computed by taking any element of the first remainder class and some element of the second remainder class, then add or multiply them in the usual way and see which remainder class the result is contained in. The following example makes this abstract description more concrete.

Example 9 (Arithmetic modulo 6). Choosing the modulus $n = 6$, we have six remainder classes of integers which are congruent modulo 6, that is, they have the same remainder when divided by 6. When we identify each of those remainder classes with the remainder, we get the following identification:

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

To compute the new addition law of those remainder class representatives, say $2 + 5$, one chooses an arbitrary element from each class, say 14 and -1 , adds those numbers in the usual way, and then looks at the remainder class of the result.

one
chooses

Adding 14 and (-1) , we get 13, and 13 is in the remainder class (of) 1. Hence, we find that $2 + 5 = 1$ in modular 6 arithmetic, which is a more readable way to write the congruence $2 + 5 \equiv 1 \pmod{6}$.

Applying the same reasoning to all remainder classes, addition and multiplication can be transferred to the representatives of the remainder classes. The results for modulus 6 arithmetic

are summarized in the following addition and multiplication tables:

+	0	1	2	3	4	5		·	0	1	2	3	4	5
0	0	1	2	3	4	5		0	0	0	0	0	0	0
1	1	2	3	4	5	0		1	0	1	2	3	4	5
2	2	3	4	5	0	1		2	0	2	4	0	2	4
3	3	4	5	0	1	2		3	0	3	0	3	0	3
4	4	5	0	1	2	3		4	0	4	2	0	4	2
5	5	0	1	2	3	4		5	0	5	4	3	2	1

(3.19)

This way, we have defined a new arithmetic system that contains just 6 numbers and comes with its own definition of addition and multiplication. We call it **modular 6 arithmetic** and write the associated **type** as \mathbb{Z}_6 .

To see why identifying a remainder class with its remainder is useful and actually simplifies congruence computations significantly, let's go back to the congruence from example 7:

$$7 \cdot (2x + 21) + 11 \equiv x - 102 \pmod{6} \quad (3.20)$$

As shown in example 7, the arithmetic of congruences can deviate from ordinary arithmetic: for example, division needs to check whether the modulus and the dividend are coprimes, and solutions are not unique in general.

We can rewrite the congruence in (3.20) as an **equation** over our new arithmetic type \mathbb{Z}_6 by **projecting onto the remainder classes**: since $7 \bmod 6 = 1$, $21 \bmod 6 = 3$, $11 \bmod 6 = 5$ and $102 \bmod 6 = 0$, we get the following:

$$\begin{aligned} 7 \cdot (2x + 21) + 11 &\equiv x - 102 \pmod{6} \text{ over } \mathbb{Z} \\ &\Leftrightarrow 1 \cdot (2x + 3) + 5 = x \text{ over } \mathbb{Z}_6 \end{aligned}$$

We can use the multiplication and addition table in (3.19) above to solve the equation on the right like we would solve normal integer equations:

$$\begin{aligned} 1 \cdot (2x + 3) + 5 &= x \\ 2x + 3 + 5 &= x && \# \text{ addition table: } 3 + 5 = 2 \\ 2x + 2 &= x && \# \text{ add 4 and } -x \text{ on both sides} \\ 2x + 2 + 4 - x &= x + 4 - x && \# \text{ addition table: } 2 + 4 = 0 \\ x &= 4 \end{aligned}$$

As we can see, despite the somewhat unfamiliar rules of addition and multiplication, solving congruences this way is very similar to solving normal equations. And, indeed, the solution set is identical to the solution set of the original congruence, since 4 is identified with the set $\{4 + 6 \cdot k \mid k \in \mathbb{Z}\}$.

We can invoke Sage to do computations in our modular 6 arithmetic type. This is particularly useful to double-check our computations:

```

sage: Z6 = Integers(6)
sage: Z6(2) + Z6(5)
1
sage: Z6(7) * (Z6(2) * Z6(4) + Z6(21)) + Z6(11) == Z6(4) - Z6(102)
True

```


Remark 2 (k -bit modulus). In cryptographic papers, we sometimes read phrases like “[...] using a 4096-bit modulus”. This means that the underlying modulus n of the modular arithmetic used in the system has a binary representation with a length of 4096 bits. In contrast, the number 6 has the binary representation 110 and hence our example 9 describes a 3-bit modulus arithmetic system.

Exercise 20. Define \mathbb{Z}_{13} as the arithmetic modulo 13 analogously to example 9. Then consider the congruence from exercise 15 and rewrite it into an equation in \mathbb{Z}_{13} .

modulo/
modulus/
modu-
lar? unify
through-
out

3.3.5 Modular Inverses

As we know, integers can be added, subtracted and multiplied so that the result is also an integer, but this is not true for the division of integers in general: for example, $3/2$ is not an integer. To see why this is so from a more theoretical perspective, let us consider the definition of a multiplicative inverse first. When we have a set that has some kind of multiplication defined on it, and we have a distinguished element of that set that behaves neutrally with respect to that multiplication (doesn't change anything when multiplied with any other element), then we can define **multiplicative inverses** in the following way:

Definition 3.3.5.1. Let S be our set that has some notion $a \cdot b$ of multiplication and a **neutral element** $1 \in S$, such that $1 \cdot a = a$ for all elements $a \in S$. Then a **multiplicative inverse** a^{-1} of an element $a \in S$ is defined as follows:

$$a \cdot a^{-1} = 1 \quad (3.21)$$

Informally speaking, the definition of a multiplicative inverse means that it “cancels” the original element, so that multiplying the two results in 1.

Numbers that have multiplicative inverses are of particular interest, because they immediately lead to the definition of division by those numbers. In fact, if a is number such that the multiplicative inverse a^{-1} exists, then we define **division** by a simply as multiplication by the inverse:

$$\frac{b}{a} := b \cdot a^{-1} \quad (3.22)$$

Example 10. Consider the set of rational numbers, also known as fractions, \mathbb{Q} . For this set, the neutral element of multiplication is 1, since $1 \cdot a = a$ for all rational numbers. For example, $1 \cdot 4 = 4$, $1 \cdot \frac{1}{4} = \frac{1}{4}$, or $1 \cdot 0 = 0$ and so on.

Every rational number $a \neq 0$ has a multiplicative inverse, given by $\frac{1}{a}$. For example, the multiplicative inverse of 3 is $\frac{1}{3}$, since $3 \cdot \frac{1}{3} = 1$, the multiplicative inverse of $\frac{5}{7}$ is $\frac{7}{5}$, since $\frac{5}{7} \cdot \frac{7}{5} = 1$, and so on.

Example 11. Looking at the set \mathbb{Z} of integers, we see that the neutral element of multiplication is the number 1. We can also see that no integer other than 1 or -1 has a multiplicative inverse, since the equation $a \cdot x = 1$ has no integer solutions for $a \neq 1$ or $a \neq -1$.

The definition of multiplicative inverse has a parallel for addition called the **additive inverse**. In the case of integers, the neutral element with respect to addition is 0, since $a + 0 = 0$ for all integers $a \in \mathbb{Z}$. The additive inverse always exists, and is given by the negative number $-a$, since $a + (-a) = 0$.

Example 12. Looking at the set \mathbb{Z}_6 of residue classes modulo 6 from example 9, we can use the multiplication table in (3.19) to find multiplicative inverses. To do so, we look at the row of the element and find the entry equal to 1. If such an entry exists, the element of that column is the

989 multiplicative inverse. If, on the other hand, the row has no entry equal to 1, we know that the
 990 element has no multiplicative inverse.

991 For example in, \mathbb{Z}_6 , the multiplicative inverse of 5 is 5 itself, since $5 \cdot 5 = 1$. We can also
 992 see that 5 and 1 are the only elements that have multiplicative inverses in \mathbb{Z}_6 .

Now, since 5 has a multiplicative inverse in modulo 6 arithmetic, we can divide by 5 in \mathbb{Z}_6 ,
 since we have a notation of multiplicative inverse and division is nothing but multiplication by
 the multiplicative inverse:

$$\frac{4}{5} = 4 \cdot 5^{-1} = 4 \cdot 5 = 2$$

993 From the last example, we can make the interesting observation that, while 5 has no multi-
 994 plicative inverse as an integer, it has a multiplicative inverse in modular 6 arithmetic.

995 This raises the question of which numbers have multiplicative inverses in modular arith-
 996 metic. The answer is that, in modular n arithmetic, a number r has a multiplicative inverse if
 997 and only if n and r are coprime. Since $\gcd(n, r) = 1$ in that case, we know from the Extended
 998 Euclidean Algorithm that there are numbers s and t , such that the following equation holds:

$$1 = s \cdot n + t \cdot r \quad (3.23)$$

999 If we take the modulus n on both sides, the term $s \cdot n$ vanishes, which tells us that $t \bmod n$ is the
 1000 multiplicative inverse of r in modular n arithmetic.

1001 *Example 13* (Multiplicative inverses in \mathbb{Z}_6). In the previous example, we looked up multiplica-
 1002 tive inverses in \mathbb{Z}_6 from the lookup table in (3.19). In real-world examples, it is usually impossi-
 1003 ble to write down those lookup tables, as the modulus is way too large, and the sets occasionally
 1004 contain more elements than there are atoms in the observable universe.

1005 Now, trying to determine that $2 \in \mathbb{Z}_6$ has no multiplicative inverse in \mathbb{Z}_6 without using the
 1006 lookup table, we immediately observe that 2 and 6 are not coprime, since their greatest common
 1007 divisor is 2. It follows that equation 3.23 has no solutions s and t , which means that 2 has no
 1008 multiplicative inverse in \mathbb{Z}_6 .

1009 The same reasoning works for 3 and 4, as neither of these are coprime with 6. The case
 1010 of 5 is different, since $\gcd(6, 5) = 1$. To compute the multiplicative inverse of 5, we use the
 1011 Extended Euclidean Algorithm and compute the following:

k	r_k	s_k	$t_k = (r_k - s_k \cdot a) \bmod b$
0	6	1	0
1	5	0	1
2	1	1	-1
3	0	.	.

1012 We get $s = 1$ as well as $t = -1$ and have $1 = 1 \cdot 6 - 1 \cdot 5$. From this, it follows that $-1 \bmod 6 =$
 1013 5 is the multiplicative inverse of 5 in modular 6 arithmetic. We can double check using Sage:

1014 **sage:** `ZZ(6).xgcd(ZZ(5))`
 1015 `(1, 1, -1)`

67

68

At this point, the attentive reader might notice that the situation where the modulus is a
 prime number is of particular interest, because we know from exercise 9 that, in these cases, all
 remainder classes must have modular inverses, since $\gcd(r, n) = 1$ for prime n and any $r < n$. In
 fact, Fermat's Little Theorem (3.16) provides a way to compute multiplicative inverses in this

situation, since, in case of a prime modulus p and $r < p$, we get the following:

$$\begin{aligned} r^p &\equiv r \pmod{p} \Leftrightarrow \\ r^{p-1} &\equiv 1 \pmod{p} \Leftrightarrow \\ r \cdot r^{p-2} &\equiv 1 \pmod{p} \end{aligned}$$

1017 This tells us that the multiplicative inverse of a residue class r in modular p arithmetic is pre-
1018 cisely r^{p-2} .

Example 14 (Modular 5 arithmetic). To see the unique properties of modular arithmetic when the modulus is a prime number, we will replicate our findings from example 9, but this time for the prime modulus 5. For $p = 5$ we have five equivalence classes of integers which are congruent modulo 5. We write this as follows:

$$\begin{aligned} 0 &:= \{\dots, -5, 0, 5, 10, \dots\} \\ 1 &:= \{\dots, -4, 1, 6, 11, \dots\} \\ 2 &:= \{\dots, -3, 2, 7, 12, \dots\} \\ 3 &:= \{\dots, -2, 3, 8, 13, \dots\} \\ 4 &:= \{\dots, -1, 4, 9, 14, \dots\} \end{aligned}$$

1019 Addition and multiplication can be transferred to the equivalence classes, in a way exactly
1020 parallel to Example 9. This results in the following addition and multiplication tables:

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

(3.24)

1021 Calling the set of remainder classes in modular 5 arithmetic with this addition and multiplication
1022 \mathbb{Z}_5 , we see some subtle but important differences to the situation in \mathbb{Z}_6 . In particular, we see
1023 that in the multiplication table, every remainder $r \neq 0$ has the entry 1 in its row and therefore
1024 has a multiplicative inverse. In addition, there are no non-zero elements such that their product
1025 is zero.

1026 To use Fermat's Little Theorem in \mathbb{Z}_5 for computing multiplicative inverses (instead of using
1027 the multiplication table), let's consider $3 \in \mathbb{Z}_5$. We know that the multiplicative inverse is given
1028 by the remainder class that contains $3^{5-2} = 3^3 = 3 \cdot 3 \cdot 3 = 4 \cdot 3 = 2$. And indeed $3^{-1} = 2$, since
1029 $3 \cdot 2 = 1$ in \mathbb{Z}_5 .

1030 We can invoke Sage to do computations in our modular 5 arithmetic type to double-check
1031 our computations:

1032	<code>sage: Z5 = Integers(5)</code>	69
1033	<code>sage: Z5(3) ** (5-2)</code>	70
1034	<code>2</code>	71
1035	<code>sage: Z5(3) ** (-1)</code>	72
1036	<code>2</code>	73
1037	<code>sage: Z5(3) ** (5-2) == Z5(3) ** (-1)</code>	74
1038	<code>True</code>	75

Example 15. To understand one of the principal differences between prime number modular arithmetic and non-prime number modular arithmetic, consider the linear equation $a \cdot x + b = 0$ defined over both types \mathbb{Z}_5 and \mathbb{Z}_6 . Since every non-zero element has a multiplicative inverse in \mathbb{Z}_5 , we can always solve these equations in \mathbb{Z}_5 , which is not true in \mathbb{Z}_6 . To see that, consider the equation $3x + 3 = 0$. In \mathbb{Z}_5 we have the following:

$$\begin{array}{ll}
 3x + 3 = 0 & \# \text{ add 2 and on both sides} \\
 3x + 3 + 2 = 2 & \# \text{ addition-table: } 2 + 3 = 0 \\
 3x = 2 & \# \text{ divide by 3 (which equals multiplication by 2)} \\
 2 \cdot (3x) = 2 \cdot 2 & \# \text{ multiplication-table: } 2 \cdot 2 = 4 \\
 x = 4 &
 \end{array}$$

So in the case of our prime number modular arithmetic, we get the unique solution $x = 4$. Now consider \mathbb{Z}_6 :

$$\begin{array}{ll}
 3x + 3 = 0 & \# \text{ add 3 and on both sides} \\
 3x + 3 + 3 = 3 & \# \text{ addition-table: } 3 + 3 = 0 \\
 3x = 3 & \# \text{ division not possible (no multiplicative inverse of 3 exists)}
 \end{array}$$

So, in this case, we cannot solve the equation for x by dividing by 3. And, indeed, when we look at the multiplication table of \mathbb{Z}_6 (example 9), we find that there are three solutions $x \in \{1, 3, 5\}$, such that $3x + 3 = 0$ holds true for all of them.

Exercise 21. Consider the modulus $n = 24$. Which of the integers 7, 1, 0, 805, -4255 have multiplicative inverses in modular 24 arithmetic? Compute the inverses, in case they exist.

Exercise 22. Find the set of all solutions to the congruence $17(2x + 5) - 4 \equiv 2x + 4 \pmod{5}$. Then project the congruence into \mathbb{Z}_5 and solve the resulting equation in \mathbb{Z}_5 . Compare the results.

Exercise 23. Find the set of all solutions to the congruence $17(2x + 5) - 4 \equiv 2x + 4 \pmod{6}$. Then project the congruence into \mathbb{Z}_6 and try to solve the resulting equation in \mathbb{Z}_6 .

3.4 Polynomial arithmetic

A polynomial is an expression consisting of variables (also-called indeterminates) and coefficients that involves only the operations of addition, subtraction and multiplication. All coefficients of a polynomial must have the same type, e.g. they must all be integers or they must all be rational numbers, etc.

To be more precise, an **univariate**³ **polynomial** is an expression as shown below:

$$P(x) := \sum_{j=0}^m a_j x^j = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0, \quad (3.25)$$

In (3.25) x is called the **variable**, and each a is called a **coefficient**. If \mathbb{R} is the type of the coefficients, then the set of all **univariate polynomials with coefficients in \mathbb{R}** is written as $\mathbb{R}[x]$. Univariate polynomials are often simply called polynomials, and written as $P(x) \in \mathbb{R}[x]$. The constant term a_0 as is also written as $P(0)$.

A polynomial is called the **zero polynomial** if all its coefficients are zero. A polynomial is called the **one polynomial** if the constant term is 1 and all other coefficients are zero.

³In our context, the term univariate means that the polynomial contains a single variable only.

Given a univariate polynomial $P(x) = \sum_{j=0}^m a_j x^j$ that is not the zero polynomial, we call the non-negative integer $\deg(P) := m$ the *degree* of P , and define the degree of the zero polynomial to be $-\infty$, where $-\infty$ (negative infinity) is a symbol with the properties that $-\infty + m = -\infty$ and $-\infty < m$ for all non-negative integers $m \in \mathbb{N}_0$.

In addition, we denote the coefficient of the term with the highest degree, called **leading coefficient**, of the polynomial P as follows:

$$Lc(P) := a_m \quad (3.26)$$

We can restrict the set $\mathbb{R}[x]$ of **all** polynomials with coefficients in \mathbb{R} to the set of all such polynomials that have a degree that does not exceed a certain value. If m is the maximum degree allowed, we write $\mathbb{R}_{\leq m}[x]$ for the set of all polynomials with a degree less than or equal to m .

Example 16 (Integer Polynomials). The coefficients of a polynomial must all have the same type. The set of polynomials with integer coefficients is written as $\mathbb{Z}[x]$. Some examples of such polynomials are listed below:

$P_1(x) = 2x^2 - 4x + 17$	# with $\deg(P_1) = 2$ and $Lc(P_1) = 2$
$P_2(x) = x^{23}$	# with $\deg(P_2) = 23$ and $Lc(P_2) = 1$
$P_3(x) = x$	# with $\deg(P_3) = 1$ and $Lc(P_3) = 1$
$P_4(x) = 174$	# with $\deg(P_4) = 0$ and $Lc(P_4) = 174$
$P_5(x) = 1$	# with $\deg(P_5) = 0$ and $Lc(P_5) = 1$
$P_6(x) = 0$	# with $\deg(P_6) = -\infty$ and $Lc(P_6) = 0$
$P_7(x) = (x-2)(x+3)(x-5)$	

Every integer can be seen as an integer polynomial of degree zero. P_7 is a polynomial, because we can expand its definition into $P_7(x) = x^3 - 4x^2 - 11x + 30$, which is a polynomial of degree 3 and leading coefficient 1.

The following expressions are not integer polynomials:

$$\begin{aligned} Q_1(x) &= 2x^2 + 4 + 3x^{-2} \\ Q_2(x) &= 0.5x^4 - 2x \\ Q_3(x) &= 2^x \end{aligned}$$

Q_1 is not an integer polynomial because the expression x^{-2} has a negative exponent. Q_2 is not an integer polynomial because the coefficient 0.5 is not an integer. Q_3 is not an integer polynomial because the variable appears in the exponent of a coefficient.

We can invoke Sage to do computations with polynomials. To do so, we have to specify the symbol for the variable and the type for the coefficients. (For the definition of rings see 4.2.) **Note, however, that Sage defines the degree of the zero polynomial to be -1 .**

```
sage: Zx = ZZ['x'] # integer polynomials with variable x
sage: Zt.<t> = ZZ[] # integer polynomials with variable t
sage: Zx
Univariate Polynomial Ring in x over Integer Ring
sage: Zt
Univariate Polynomial Ring in t over Integer Ring
sage: p1 = Zx([17,-4,2])
sage: p1
```

add explanation on why this is important

why is this ref here?

what does this imply?

1086	<code>2*x^2 - 4*x + 17</code>	84
1087	<code>sage: p1.degree()</code>	85
1088	<code>2</code>	86
1089	<code>sage: p1.leading_coefficient()</code>	87
1090	<code>2</code>	88
1091	<code>sage: p2 = Zt(t^23)</code>	89
1092	<code>sage: p2</code>	90
1093	<code>t^23</code>	91
1094	<code>sage: p6 = Zx([0])</code>	92
1095	<code>sage: p6.degree()</code>	93
1096	<code>-1</code>	94

Example 17 (Polynomials over \mathbb{Z}_6). Recall the definition of modular 6 arithmetics \mathbb{Z}_6 from example 9. The set of all polynomials with variable x and coefficients in \mathbb{Z}_6 is symbolized as $\mathbb{Z}_6[x]$. Some examples of polynomials from $\mathbb{Z}_6[x]$ are given below:

$$\begin{aligned}
 P_1(x) &= 2x^2 - 4x + 5 && \# \text{ with } \deg(P_1) = 2 \text{ and } Lc(P_1) = 2 \\
 P_2(x) &= x^{23} && \# \text{ with } \deg(P_2) = 23 \text{ and } Lc(P_2) = 1 \\
 P_3(x) &= x && \# \text{ with } \deg(P_3) = 1 \text{ and } Lc(P_3) = 1 \\
 P_4(x) &= 3 && \# \text{ with } \deg(P_4) = 0 \text{ and } Lc(P_4) = 3 \\
 P_5(x) &= 1 && \# \text{ with } \deg(P_5) = 0 \text{ and } Lc(P_5) = 1 \\
 P_6(x) &= 0 && \# \text{ with } \deg(P_6) = -\infty \text{ and } Lc(P_6) = 0 \\
 P_7(x) &= (x-2)(x+3)(x-5)
 \end{aligned}$$

Just like in the previous example, P_7 is a polynomial. However, since we are working with coefficients from \mathbb{Z}_6 now, the expansion of P_7 is computed differently, as we have to invoke addition and multiplication in \mathbb{Z}_6 as defined in (3.19). We get the following:

$$\begin{aligned}
 (x-2)(x+3)(x-5) &= (x+4)(x+3)(x+1) && \# \text{ additive inverses in } \mathbb{Z}_6 \\
 &= (x^2 + 4x + 3x + 3 \cdot 4)(x+1) && \# \text{ bracket expansion} \\
 &= (x^2 + 1x + 0)(x+1) && \# \text{ computation in } \mathbb{Z}_6 \\
 &= x^3 + x^2 + x^2 + x && \# \text{ bracket expansion} \\
 &= x^3 + 2x^2 + x
 \end{aligned}$$

check
reference

1097 Again, we can use Sage to do computations with polynomials that have their coefficients in \mathbb{Z}_6 .
 1098 (For the definition of rings see 4.2.) To do so, we have to specify the symbol for the variable
 1099 and the type for the coefficients:

1100	<code>sage: Z6 = Integers(6)</code>	95
1101	<code>sage: Z6x = Z6['x']</code>	96
1102	<code>sage: Z6x</code>	97
1103	<code>Univariate Polynomial Ring in x over Ring of integers modulo 6</code>	98
1104	<code>sage: p1 = Z6x([5,-4,2])</code>	99
1105	<code>sage: p1</code>	100
1106	<code>2*x^2 + 2*x + 5</code>	101
1107	<code>sage: p1 = Z6x([17,-4,2])</code>	102
1108	<code>sage: p1</code>	103

why is
this ref
here?

```

1109 2*x^2 + 2*x + 5 104
1110 sage: Z6x(x-2)*Z6x(x+3)*Z6x(x-5) == Z6x(x^3 + 2*x^2 + x) 105
1111 True 106

```

1112 Given some element from the same type as the coefficients of a polynomial, the polynomial can be evaluated at that element, which means that we insert the given element for every occurrence of the variable x in the polynomial expression.

1115 To be more precise, let $P \in \mathbb{R}[x]$, with $P(x) = \sum_{j=0}^m a_j x^j$ be a polynomial with a coefficient of type \mathbb{R} and let $b \in \mathbb{R}$ be an element of that type. Then the **evaluation** of P at b is given as follows:

$$P(b) = \sum_{j=0}^m a_j b^j \quad (3.27)$$

Example 18. Consider the integer polynomials from example 16 again. To evaluate them at given points, we have to insert the point for all occurrences of x in the polynomial expression. Inserting arbitrary values from \mathbb{Z} , we get the following:

$$\begin{aligned}
 P_1(2) &= 2 \cdot 2^2 - 4 \cdot 2 + 17 = 17 \\
 P_2(3) &= 3^{23} = 94143178827 \\
 P_3(-4) &= -4 = -4 \\
 P_4(15) &= 174 \\
 P_5(0) &= 1 \\
 P_6(1274) &= 0 \\
 P_7(-6) &= (-6-2)(-6+3)(-6-5) = -264
 \end{aligned}$$

1118 Note, however, that it is not possible to evaluate any of those polynomial on values of different type. For example, it is not strictly correct to write $P_1(0.5)$, since 0.5 is not an integer. We can verify our computations using Sage:

```

1121 sage: Zx = ZZ['x'] 107
1122 sage: p1 = Zx([17, -4, 2]) 108
1123 sage: p7 = Zx(x-2)*Zx(x+3)*Zx(x-5) 109
1124 sage: p1(ZZ(2)) 110
1125 17 111
1126 sage: p7(ZZ(-6)) == ZZ(-264) 112
1127 True 113

```

Example 19. Consider the polynomials with coefficients in \mathbb{Z}_6 from example 17 again. To evaluate them at given values from \mathbb{Z}_6 , we have to insert the point for all occurrences of x in the polynomial expression. We get the following:

$$\begin{aligned}
 P_1(2) &= 2 \cdot 2^2 - 4 \cdot 2 + 5 = 2 - 2 + 5 = 5 \\
 P_2(3) &= 3^{23} = 3 \\
 P_3(-4) &= P_3(2) = 2 \\
 P_5(0) &= 1 \\
 P_6(4) &= 0
 \end{aligned}$$

check
reference

1128

```

1129 sage: Z6 = Integers(6) 114
1130 sage: Z6x = Z6['x'] 115
1131 sage: p1 = Z6x([5, -4, 2]) 116
1132 sage: p1(Z6(2)) == Z6(5) 117
1133 True 118

```

Exercise 24. Compare both expansions of P_7 from $\mathbb{Z}[x]$ in example 16 and from $\mathbb{Z}_6[x]$ in example 17, and consider the definition of \mathbb{Z}_6 as given in example 9. Can you see how the definition of P_7 over \mathbb{Z} projects to the definition over \mathbb{Z}_6 if you consider the residue classes of \mathbb{Z}_6 ?

the task
could be
defined
more
clearly

3.4.1 Polynomial arithmetic

Polynomials behave like integers in many ways. In particular, they can be added, subtracted and multiplied. In addition, they have their own notion of Euclidean Division. Informally speaking, we can add two polynomials by simply adding the coefficients of the same index, and we can multiply them by applying the distributive property, that is, by multiplying every term of the left factor with every term of the right factor and adding the results together.

To be more precise, let $\sum_{n=0}^{m_1} a_n x^n$ and $\sum_{n=0}^{m_2} b_n x^n$ be two polynomials from $\mathbb{R}[x]$. Then the **sum** and the **product** of these polynomials is defined as follows:

$$\sum_{n=0}^{m_1} a_n x^n + \sum_{n=0}^{m_2} b_n x^n = \sum_{n=0}^{\max\{m_1, m_2\}} (a_n + b_n) x^n \quad (3.28)$$

$$\left(\sum_{n=0}^{m_1} a_n x^n \right) \cdot \left(\sum_{n=0}^{m_2} b_n x^n \right) = \sum_{n=0}^{m_1+m_2} \sum_{i=0}^n a_i b_{n-i} x^n \quad (3.29)$$

A rule for polynomial subtraction can be deduced from these two rules by first multiplying the **subtrahend** with (the polynomial) -1 and then add the result to the **minuend**.

subtrahend

Regarding the definition of the degree of a polynomial, we see that the degree of the sum is always the maximum of the degrees of both summands, and the degree of the product is always the degree of the sum of the factors, since we defined $-\infty + m = -\infty$ for every integer $m \in \mathbb{Z}$.

minuend

Example 20. To give an example of how polynomial arithmetic works, consider the following two integer polynomials $P, Q \in \mathbb{Z}[x]$ with $P(x) = 5x^2 - 4x + 2$ and $Q(x) = x^3 - 2x^2 + 5$. The sum of these two polynomials is computed by adding the coefficients of each term with equal exponent in x . This gives the following:

$$\begin{aligned} (P + Q)(x) &= (0 + 1)x^3 + (5 - 2)x^2 + (-4 + 0)x + (2 + 5) \\ &= x^3 + 3x^2 - 4x + 7 \end{aligned}$$

The product of these two polynomials is computed by multiplying each term in the first factor with each term in the second factor. We get the following:

$$\begin{aligned} (P \cdot Q)(x) &= (5x^2 - 4x + 2) \cdot (x^3 - 2x^2 + 5) \\ &= (5x^5 - 10x^4 + 25x^2) + (-4x^4 + 8x^3 - 20x) + (2x^3 - 4x^2 + 10) \\ &= 5x^5 - 14x^4 + 10x^3 + 21x^2 - 20x + 10 \end{aligned}$$


```

1153 sage: Zx = ZZ['x'] 119
1154 sage: P = Zx([2, -4, 5]) 120
1155 sage: Q = Zx([5, 0, -2, 1]) 121
1156 sage: P+Q == Zx(x^3 + 3*x^2 - 4*x + 7) 122
1157 True 123
1158 sage: P*Q == Zx(5*x^5 - 14*x^4 + 10*x^3 + 21*x^2 - 20*x + 10) 124
1159 True 125

```

Example 21. Let us consider the polynomials of the previous example 20, but interpreted in modular 6 arithmetic. So we consider $P, Q \in \mathbb{Z}_6[x]$ again with $P(x) = 5x^2 - 4x + 2$ and $Q(x) = x^3 - 2x^2 + 5$. This time we get the following:

$$\begin{aligned}
 (P+Q)(x) &= (0+1)x^3 + (5-2)x^2 + (-4+0)x + (2+5) \\
 &= (0+1)x^3 + (5+4)x^2 + (2+0)x + (2+5) \\
 &= x^3 + 3x^2 + 2x + 1
 \end{aligned}$$

$$\begin{aligned}
 (P \cdot Q)(x) &= (5x^2 - 4x + 2) \cdot (x^3 - 2x^2 + 5) \\
 &= (5x^2 + 2x + 2) \cdot (x^3 + 4x^2 + 5) \\
 &= (5x^5 + 2x^4 + 1x^2) + (2x^4 + 2x^3 + 4x) + (2x^3 + 2x^2 + 4) \\
 &= 5x^5 + 4x^4 + 4x^3 + 3x^2 + 4x + 4
 \end{aligned}$$

```

1160
1161 sage: Z6x = Integers(6)['x'] 126
1162 sage: P = Z6x([2, -4, 5]) 127
1163 sage: Q = Z6x([5, 0, -2, 1]) 128
1164 sage: P+Q == Z6x(x^3 + 3*x^2 + 2*x + 1) 129
1165 True 130
1166 sage: P*Q == Z6x(5*x^5 + 4*x^4 + 4*x^3 + 3*x^2 + 4*x + 4) 131
1167 True 132

```

Exercise 25. Compare the sum $P+Q$ and the product $P \cdot Q$ from the previous two examples 20 and 21, and consider the definition of \mathbb{Z}_6 as given in example 9. How can we derive the computations in $\mathbb{Z}_6[x]$ from the computations in $\mathbb{Z}[x]$?

1171 3.4.2 Euclidean Division with polynomials

1172 The arithmetic of polynomials shares a lot of properties with the arithmetic of integers. As
 1173 a consequence, the concept of Euclidean Division and the algorithm of long division is also
 1174 defined for polynomials. Recalling the Euclidean Division of integers 3.2.2, we know that,
 1175 given two integers a and $b \neq 0$, there is always another integer m and a natural number r with
 1176 $r < |b|$ such that $a = m \cdot b + r$ holds.

1177 We can generalize this to polynomials whenever the leading coefficient of the dividend
 1178 polynomial has a notion of multiplicative inverse. In fact, given two polynomials A and $B \neq 0$
 1179 from $\mathbb{R}[x]$ such that $Lc(B)^{-1}$ exists in \mathbb{R} , there exist two polynomials Q (the quotient) and P (the
 1180 remainder), such that the following equation holds and $\deg(P) < \deg(B)$:

$$A = Q \cdot B + P \quad (3.30)$$

Similarly to integer Euclidean Division, both Q and P are uniquely defined by these relations.

Notation and Symbols 2. Suppose that the polynomials A, B, Q and P satisfy equation 3.30. We often use the following notation to describe the quotient and the remainder polynomials of the Euclidean Division:

$$A \operatorname{div} B := Q, \quad A \operatorname{mod} B := P \quad (3.31)$$

We also say that a polynomial A is divisible by another polynomial B if $A \operatorname{mod} B = 0$ holds. In this case, we also write $B|A$ and call B a *factor* of A .

Analogously to integers, methods to compute Euclidean Division for polynomials are called **polynomial division algorithms**. Probably the best known algorithm is the so-called **polynomial long division**.

algorithm-
floating

Algorithm 3 Polynomial Euclidean Algorithm

Require: $A, B \in R[x]$ with $B \neq 0$, such that $Lc(B)^{-1}$ exists in R

procedure POLY-LONG-DIVISION(A, B)

$Q \leftarrow 0$

$P \leftarrow A$

$d \leftarrow \deg(B)$

$c \leftarrow Lc(B)$

while $\deg(P) \geq d$ **do**

$S := Lc(P) \cdot c^{-1} \cdot x^{\deg(P)-d}$

$Q \leftarrow Q + S$

$P \leftarrow P - S \cdot B$

end while

return (Q, P)

end procedure

Ensure: $A = Q \cdot B + P$

This algorithm works only when there is a notion of division by the leading coefficient of B . It can be generalized, but we will only need this somewhat simpler method in what follows.

Example 22 (Polynomial Long Division). To give an example of how the previous algorithm works, let us divide the integer polynomial $A(x) = x^5 + 2x^3 - 9 \in \mathbb{Z}[x]$ by the integer polynomial $B(x) = x^2 + 4x - 1 \in \mathbb{Z}[x]$. Since B is not the zero polynomial, and the leading coefficient of B is 1, which is invertible as an integer, we can apply algorithm 1. Our goal is to find solutions to equation XXX, that is, we need to find the quotient polynomial $Q \in \mathbb{Z}[x]$ and the remainder polynomial $P \in \mathbb{Z}[x]$ such that $x^5 + 2x^3 - 9 = Q(x) \cdot (x^2 + 4x - 1) + P(x)$. Using a the long

add refer-
ence

1199 division notation that is mostly used in anglophone countries, we compute as follows:

$$\begin{array}{r}
 X^3 - 4X^2 + 19X - 80 \\
 X^2 + 4X - 1 \overline{) } \\
 \underline{X^3 + 2X^3 } \\
 -X^5 - 4X^4 + X^3 \\
 \underline{-4X^4 + 3X^3} \\
 4X^4 + 16X^3 - 4X^2 \\
 \underline{19X^3 - 4X^2} \\
 -19X^3 - 76X^2 + 19X \\
 \underline{-80X^2 + 19X - 9} \\
 80X^2 + 320X - 80 \\
 \underline{339X - 89}
 \end{array} \tag{3.32}$$

1200 We therefore get $Q(x) = x^3 - 4x^2 + 19x - 80$ and $P(x) = 339x - 89$, and indeed, the equation
 1201 $A = Q \cdot B + P$ is true with these valude, since $x^5 + 2x^3 - 9 = (x^3 - 4x^2 + 19x - 80) \cdot (x^2 + 4x -$
 1202 $1) + (339x - 89)$. We can double check this invoking Sage:

```

1203 sage: Zx = ZZ['x']                                     133
1204 sage: A = Zx([-9, 0, 0, 2, 0, 1])                       134
1205 sage: B = Zx([-1, 4, 1])                                 135
1206 sage: Q = Zx([-80, 19, -4, 1])                         136
1207 sage: P = Zx([-89, 339])                                137
1208 sage: A == Q*B + P                                     138
1209 True                                                  139

```

1210 *Example 23.* In the previous example, polynomial division gave a non-trivial (non-vanishing,
 1211 i.e non-zero) remainder. Divisions that don't give a remainder are of special interest. In these
 1212 cases, divisors are called **factors of the dividend**.

1213 For example, consider the integer polynomial P_7 from example 16 again. As we have shown,
 1214 it can be written both as $x^3 - 4x^2 - 11x + 30$ and as $(x - 2)(x + 3)(x - 5)$. From this, we can
 1215 see that the polynomials $F_1(x) = (x - 2)$, $F_2(x) = (x + 3)$ and $F_3(x) = (x - 5)$ are all factors of
 1216 $x^3 - 4x^2 - 11x + 30$, since division of P_7 by any of these factors will result in a zero remainder.

1217 *Exercise 26.* Consider the polynomial expressions $A(x) := -3x^4 + 4x^3 + 2x^2 + 4$ and $B(x) =$
 1218 $x^2 - 4x + 2$. Compute the Euclidean Division of A by B in the following types:

- 1219 1. $A, B \in \mathbb{Z}[x]$
- 1220 2. $A, B \in \mathbb{Z}_6[x]$
- 1221 3. $A, B \in \mathbb{Z}_5[x]$

1222 Now consider the result in $\mathbb{Z}[x]$ and in $\mathbb{Z}_6[x]$. How can we compute the result in $\mathbb{Z}_6[x]$ from the
 1223 result in $\mathbb{Z}[x]$?

1224 *Exercise 27.* Show that the polynomial $B(x) = 2x^4 - 3x + 4 \in \mathbb{Z}_5[x]$ is a factor of the polynomial
 1225 $A(x) = x^7 + 4x^6 + 4x^5 + x^3 + 2x^2 + 2x + 3 \in \mathbb{Z}_5[x]$, that is, show that $B|A$. What is $B \text{ div } A$?

3.4.3 Prime Factors

Recall that the fundamental theorem of arithmetic 3.7 tells us that every natural number is the product of prime numbers. In this chapter, we will see that something similar holds for univariate polynomials $R[x]$, too.⁴

The polynomial analog to a prime number is a so-called **irreducible polynomial**, which is defined as a polynomial that cannot be factored into the product of two non-constant polynomials using Euclidean Division. Irreducible polynomials are to polynomials what prime numbers are to integers: they are the basic building blocks from which all other polynomials can be constructed.

To be more precise, let $P \in \mathbb{R}[x]$ be any polynomial. Then there always exist irreducible polynomials $F_1, F_2, \dots, F_k \in \mathbb{R}[x]$, such that the following holds:

$$P = F_1 \cdot F_2 \cdot \dots \cdot F_k. \quad (3.33)$$

This representation is unique (except for permutations in the factors) and is called the **prime factorization** of P . Moreover, each factor F_i is called a **prime factor** of P .

Example 24. Consider the polynomial expression $P = x^2 - 3$. When we interpret P as an integer polynomial $P \in \mathbb{Z}[x]$, we find that this polynomial is irreducible, since any factorization other than $1 \cdot (x^2 - 3)$, must look like $(x - a)(x + a)$ for some integer a , but there is no integers a with $a^2 = 3$.

```

sage: Zx = ZZ['x']
sage: p = Zx(x^2-3)
sage: p.factor()
x^2 - 3

```

On the other hand, interpreting P as a polynomial $P \in \mathbb{Z}_6[x]$ in modulo 6 arithmetic, we see that P has two factors $F_1 = (x - 3)$ and $F_2 = (x + 3)$, since $(x - 3)(x + 3) = x^2 - 3x + 3x - 3 \cdot 3 = x^2 - 3$.

Points where a polynomial evaluates to zero are called **roots** of the polynomial. To be more precise, let $P \in \mathbb{R}[x]$ be a polynomial. Then a root is a point $x_0 \in \mathbb{R}$ with $P(x_0) = 0$ and the set of all roots of P is defined as follows:

$$R_0(P) := \{x_0 \in \mathbb{R} \mid P(x_0) = 0\} \quad (3.34)$$

The roots of a polynomial are of special interest with respect to its prime factorization, since it can be shown that, for any given root x_0 of P , the polynomial $F(x) = (x - x_0)$ is a prime factor of P .

Finding the roots of a polynomial is sometimes called **solving the polynomial**. It is a difficult problem that has been the subject of much research throughout history.

It can be shown that if m is the degree of a polynomial P , then P cannot have more than m roots. However, in general, polynomials can have less than m roots.

Example 25. Consider the integer polynomial $P_7(x) = x^3 - 4x^2 - 11x + 30$ from example 16 again. We know that its set of roots is given by $R_0(P_7) = \{-3, 2, 5\}$.

On the other hand, we know from example 24 that the integer polynomial $x^2 - 3$ is irreducible. It follows that it has no roots, since every root defines a prime factor.

⁴Strictly speaking, this is not true for polynomials over arbitrary types \mathbb{R} . However, in this book, we assume \mathbb{R} to be a so-called unique factorization domain for which the content of this section holds.

1264 *Example 26.* To give another example, consider the integer polynomial $P = x^7 + 3x^6 + 3x^5 +$
 1265 $x^4 - x^3 - 3x^2 - 3x - 1$. We can invoke Sage to compute the roots and prime factors of P :

```

1266 sage: Zx = ZZ['x'] 144
1267 sage: p = Zx(x^7 + 3*x^6 + 3*x^5 + x^4 - x^3 - 3*x^2 - 3*x - 1) 145
1268 )
1269 sage: p.roots() 146
1270 [(1, 1), (-1, 4)] 147
1271 sage: p.factor() 148
1272 (x - 1) * (x + 1)^4 * (x^2 + 1) 149

```

We see that P has the root 1, and that the associated prime factor $(x - 1)$ occurs once in P . We can also see that P has the root -1 , where the associated prime factor $(x + 1)$ occurs 4 times in P . This gives the following prime factorization:

$$P = (x - 1)(x + 1)^4(x^2 + 1)$$

1273 *Exercise 28.* Show that if a polynomial $P \in \mathbb{R}[x]$ of degree $\deg(P) = m$ has less than m roots, it
 1274 must have a prime factor F of degree $\deg(F) > 1$.

1275 *Exercise 29.* Consider the polynomial $P = x^7 + 3x^6 + 3x^5 + x^4 - x^3 - 3x^2 - 3x - 1 \in \mathbb{Z}_6[x]$.
 1276 Compute the set of all roots of $R_0(P)$ and then compute the prime factorization of P .

1277 3.4.4 Lagrange interpolation

1278 One particularly useful property of polynomials is that a polynomial of degree m is completely
 1279 determined on $m + 1$ evaluation points, which implies that we can uniquely derive a polynomial
 1280 of degree m from a set S :

$$S = \{(x_0, y_0), (x_1, y_1), \dots, (x_m, y_m) \mid x_i \neq x_j \text{ for all indices } i \text{ and } j\} \quad (3.35)$$

1281 Polynomials therefore have the property that $m + 1$ pairs of points (x_i, y_i) for $x_i \neq x_j$ are enough
 1282 to determine the set of pairs $(x, P(x))$ for all $x \in \mathbb{R}$. This “few too many” property of polynomials
 1283 is widely used, including in SNARKs. Therefore, we need to understand the method to actually
 1284 compute a polynomial from a set of points.

1285 If the coefficients of the polynomial we want to find have a notion of multiplicative inverse,
 1286 it is always possible to find such a polynomial using a method called **Lagrange interpolation**,
 1287 which works as follows. Given a set like 3.35, a polynomial P of degree m with $P(x_i) = y_i$ for
 1288 all pairs (x_i, y_i) from S is given by the following algorithm:

Example 27. Let us consider the set $S = \{(0, 4), (-2, 1), (2, 3)\}$. Our task is to compute a polynomial of degree 2 in $\mathbb{Q}[x]$ with coefficients from the set of rational numbers \mathbb{Q} . Since \mathbb{Q} has multiplicative inverses, we can use method of Lagrange interpolation from Algorithm 4 to

check
algorithm
floating

Algorithm 4 Lagrange Interpolation**Require:** R must have multiplicative inverses**Require:** $S = \{(x_0, y_0), (x_1, y_1), \dots, (x_m, y_m) \mid x_i, y_i \in R, x_i \neq x_j \text{ for all indices } i \text{ and } j\}$ **procedure** LAGRANGE-INTERPOLATION(S) **for** $j \in (0 \dots m)$ **do**

$$l_j(x) \leftarrow \prod_{i=0; i \neq j}^m \frac{x - x_i}{x_j - x_i} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_m)}{(x_j - x_m)}$$

end for

$$P \leftarrow \sum_{j=0}^m y_j \cdot l_j$$

return P **end procedure****Ensure:** $P \in R[x]$ with $\deg(P) = m$ **Ensure:** $P(x_j) = y_j$ for all pairs $(x_j, y_j) \in S$

compute the polynomial:

$$\begin{aligned} l_0(x) &= \frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_2} = \frac{x + 2}{0 + 2} \cdot \frac{x - 2}{0 - 2} = -\frac{(x + 2)(x - 2)}{4} \\ &= -\frac{1}{4}(x^2 - 4) \\ l_1(x) &= \frac{x - x_0}{x_1 - x_0} \cdot \frac{x - x_2}{x_1 - x_2} = \frac{x - 0}{-2 - 0} \cdot \frac{x - 2}{-2 - 2} = \frac{x(x - 2)}{8} \\ &= \frac{1}{8}(x^2 - 2x) \\ l_2(x) &= \frac{x - x_0}{x_2 - x_0} \cdot \frac{x - x_1}{x_2 - x_1} = \frac{x - 0}{2 - 0} \cdot \frac{x + 2}{2 + 2} = \frac{x(x + 2)}{8} \\ &= \frac{1}{8}(x^2 + 2x) \\ P(x) &= 4 \cdot \left(-\frac{1}{4}(x^2 - 4)\right) + 1 \cdot \frac{1}{8}(x^2 - 2x) + 3 \cdot \frac{1}{8}(x^2 + 2x) \\ &= -x^2 + 4 + \frac{1}{8}x^2 - \frac{1}{4}x + \frac{3}{8}x^2 + \frac{3}{4}x \\ &= -\frac{1}{2}x^2 + \frac{1}{2}x + 4 \end{aligned}$$

1289 And, indeed, evaluation of P on the x -values of S gives the correct points, since $P(0) = 4$,
 1290 $P(-2) = 1$ and $P(2) = 3$. Sage confirms this result:

1291	sage: <code>Qx = QQ['x']</code>	150
1292	sage: <code>S=[(0,4), (-2,1), (2,3)]</code>	151
1293	sage: <code>Qx.lagrange_polynomial(S)</code>	152
1294	<code>-1/2*x^2 + 1/2*x + 4</code>	153

Example 28. To give another example more relevant to the topics of this book, let us consider the same set as in the previous example, $S = \{(0, 4), (-2, 1), (2, 3)\}$. This time, the task is to compute a polynomial $P \in \mathbb{Z}_5[x]$ from this data. Since we know from example 14 that multiplicative inverses exist in \mathbb{Z}_5 , algorithm 4 is applicable and we can compute a unique polynomial of degree 2 in $\mathbb{Z}_5[x]$ from S . We can use the lookup tables from (3.24) for computations in \mathbb{Z}_5

and get the following:

$$l_0(x) = \frac{x-x_1}{x_0-x_1} \cdot \frac{x-x_2}{x_0-x_2} = \frac{x+2}{0+2} \cdot \frac{x-2}{0-2} = \frac{(x+2)(x-2)}{-4} = \frac{(x+2)(x+3)}{1} \\ = x^2 + 1$$

$$l_1(x) = \frac{x-x_0}{x_1-x_0} \cdot \frac{x-x_2}{x_1-x_2} = \frac{x-0}{-2-0} \cdot \frac{x-2}{-2-2} = \frac{x}{3} \cdot \frac{x+3}{1} = 2(x^2 + 3x) \\ = 2x^2 + x$$

$$l_2(x) = \frac{x-x_0}{x_2-x_0} \cdot \frac{x-x_1}{x_2-x_1} = \frac{x-0}{2-0} \cdot \frac{x+2}{2+2} = \frac{x(x+2)}{3} = 2(x^2 + 2x) \\ = 2x^2 + 4x$$

$$P(x) = 4 \cdot (x^2 + 1) + 1 \cdot (2x^2 + x) + 3 \cdot (2x^2 + 4x) \\ = 4x^2 + 4 + 2x^2 + x + x^2 + 2x \\ = 2x^2 + 3x + 4$$

1295 And, indeed, evaluation of P on the x -values of S gives the correct points, since $P(0) = 4$,
1296 $P(-2) = 1$ and $P(2) = 3$. We can double check our findings using Sage:

```
1297 sage: F5 = GF(5) 154
1298 sage: F5x = F5['x'] 155
1299 sage: S = [(0, 4), (-2, 1), (2, 3)] 156
1300 sage: F5x.lagrange_polynomial(S) 157
1301 2*x^2 + 3*x + 4 158
```

1302 *Exercise 30.* Consider modular 5 arithmetic from example 14, and the set $S = \{(0, 0), (1, 1), (2, 2), (3, 2)\}$.
1303 Find a polynomial $P \in \mathbb{Z}_5[x]$ such that $P(x_i) = y_i$ for all $(x_i, y_i) \in S$.

1304 *Exercise 31.* Consider the set S from the previous example. Why is it not possible to apply
1305 algorithm 4 to construct a polynomial $P \in \mathbb{Z}_6[x]$ such that $P(x_i) = y_i$ for all $(x_i, y_i) \in S$?

Chapter 4

Algebra

In the previous chapter, we gave an introduction to the basic computational tools needed for a pen-and-paper approach to SNARKs. In this chapter, we provide a more abstract clarification of relevant mathematical terminology such as **groups**, **rings** and **fields**.

Scientific literature on cryptography frequently contain such terms, and it is necessary to get at least some understanding of these terms to be able to follow the literature.

4.1 Commutative Groups

Commutative groups are abstractions that capture the essence of mathematical phenomena, like addition and subtraction, or multiplication and division.

To understand commutative groups, let us think back to when we learned about the addition and subtraction of integers in school. We have learned that, whenever we add two integers, the result is guaranteed to be an integer as well. We have also learned that adding zero to any integer means that “nothing happens”, that is, the result of the addition is the same integer we started with. Furthermore, we have learned that the order in which we add two (or more) integers does not matter, that brackets have no influence on the result of addition, and that, for every integer, there is always another integer (the negative) such that we get zero when we add them together.

These conditions are the defining properties of a commutative group, and mathematicians have realized that the exact same set of rules can be found in very different mathematical structures. It therefore makes sense to abstract from integers and to give a formal definition of what a group should be, detached from any concrete examples. This lets us handle entities of very different mathematical origins in a flexible way, while retaining essential structural aspects of many objects in abstract algebra and beyond.

Distilling these rules to the smallest independent list of properties and making them abstract, we arrive at the following definition of a commutative group:

Sylvia: I would like to have a separate counter for definitions

Definition 4.1.0.1. A **commutative group** (\mathbb{G}, \cdot) consists of a set \mathbb{G} and a **map** $\cdot : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$. The map is called the **group law**, and it combines two elements of the set \mathbb{G} into a third one such that the following properties hold:

- **Commutativity:** For all $g_1, g_2 \in \mathbb{G}$, the equation $g_1 \cdot g_2 = g_2 \cdot g_1$ holds.
- **Associativity:** For every $g_1, g_2, g_3 \in \mathbb{G}$ the equation $g_1 \cdot (g_2 \cdot g_3) = (g_1 \cdot g_2) \cdot g_3$ holds.
- **Existence of a neutral element:** For every $g \in \mathbb{G}$, there is an $e \in \mathbb{G}$ such that $e \cdot g = g$.

Sylvia: I would like to have a separate counter for definitions

- **Existence of an inverse:** For every $g \in \mathbb{G}$, there is a $g^{-1} \in \mathbb{G}$ such that $g \cdot g^{-1} = e$.

If (\mathbb{G}, \cdot) is a group, and $\mathbb{G}' \subset \mathbb{G}$ is a subset of \mathbb{G} such that the **restriction** of the group law $\cdot : \mathbb{G}' \times \mathbb{G}' \rightarrow \mathbb{G}'$ is a group law on \mathbb{G}' , then (\mathbb{G}', \cdot) is called a **subgroup** of (\mathbb{G}, \cdot) .

Rephrasing the abstract definition in layman's terms, a group is something where we can do computations in a way that resembles the behavior of the addition of integers. Specifically, this means we can combine some element with another element into a new element in a way that is reversible and where the order of combining elements doesn't matter.

Notation and Symbols 3. Since we are exclusively concerned with commutative groups in this book, we often just call them groups, keeping the notation of commutativity implicit.¹

If there is no risk of ambiguity (about what the group law of a group is), we frequently drop the symbol \cdot and simply write \mathbb{G} as notation for the group, keeping the group law implicit. In this case we also say that \mathbb{G} is of group type, indicating that \mathbb{G} is not simply a set but a set together with a group law.

Notation and Symbols 4 (Additive notation). For commutative groups (\mathbb{G}, \cdot) , we sometimes use the so-called **additive notation** $(\mathbb{G}, +)$, that is, we write $+$ instead of \cdot for the group law, 0 for the neutral element and $-g := g^{-1}$ for the inverse of an element $g \in \mathbb{G}$.

As we will see in the following chapters, groups are heavily used in cryptography and in SNARKs. But let us look at some more familiar examples first.

Example 29 (Integer Addition and Subtraction). The set $(\mathbb{Z}, +)$ of integers with integer addition is the archetypical example of a commutative group, where the group law is traditionally written in additive notation (Notation and Symbols 4).

To compare integer addition against the abstract axioms of a commutative group, we first note that integer addition is **commutative and associative**, since $a + b = b + a$ as well as $(a + b) + c = a + (b + c)$ for all integers $a, b, c \in \mathbb{Z}$. The **neutral element** e is the number 0 , since $a + 0 = a$ for all integers $a \in \mathbb{Z}$. Furthermore, the **inverse** of a number is its negative counterpart, since $a + (-a) = 0$ for all $a \in \mathbb{Z}$. This implies that integers with addition are indeed a commutative group in the abstract sense.

To give an example of a subgroup of the group of integers, consider the set of even numbers, including 0 .

$$\mathbb{Z}_{\text{even}} := \{\dots, -4, -2, 0, 2, 4, \dots\}$$

We can see that this set is a subgroup of $(\mathbb{Z}, +)$, since the sum of two even numbers is always an even number again, since the neutral element 0 is a member of \mathbb{Z}_{even} and since the negative of an even number is itself an even number.

Example 30 (The trivial group). The most basic example of a commutative group is the group with just one element $\{\bullet\}$ and the group law $\bullet \cdot \bullet = \bullet$. We call it the **trivial group**.

The trivial group is a subgroup of any group. To see that, let (\mathbb{G}, \cdot) be a group with the neutral element $e \in \mathbb{G}$. Then $e \cdot e = e$ as well as $e^{-1} = e$ both hold. Consequently, the set $\{e\}$ is a subgroup of \mathbb{G} . In particular, $\{0\}$ is a subgroup of $(\mathbb{Z}, +)$, since $0 + 0 = 0$.

Example 31. Consider addition in modulo 6 arithmetics $(\mathbb{Z}_6, +)$, as defined in in example 9. As we see, the remainder 0 is the neutral element in modulo 6 addition, and the inverse of a remainder r is given by $6 - r$, because $r + (6 - r) = 6$. 6 is congruent to 0 since $6 \bmod 6 = 0$. Moreover, $r_1 + r_2 = r_2 + r_1$ as well as $(r_1 + r_2) + r_3 = r_1 + (r_2 + r_3)$ are inherited from integer addition. We therefore see that $(\mathbb{Z}_6, +)$ is a group.

¹Commutative groups are also called **Abelian groups**. A set \mathbb{G} with a map \cdot that satisfies all previously mentioned rules except for the commutativity law is called a **non-commutative group**.

The previous example of a commutative group is a very important one for this book. Abstracting from this example and considering residue classes $(\mathbb{Z}_n, +)$ for arbitrary moduli n , it can be shown that $(\mathbb{Z}_n, +)$ is a commutative group with the neutral element 0 and the additive inverse $n - r$ for any element $r \in \mathbb{Z}_n$. We call such a group the **remainder class group** of modulus n .

Exercise 32. Consider example 14 again, and let \mathbb{Z}_5^* be the set of all remainder classes from \mathbb{Z}_5 without the class 0. Then $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$. Show that (\mathbb{Z}_5^*, \cdot) is a commutative group.

Exercise 33. Generalizing the previous exercise, consider the general modulus n , and let \mathbb{Z}_n^* be the set of all remainder classes from \mathbb{Z}_n without the class 0. Then $\mathbb{Z}_n^* = \{1, 2, \dots, n-1\}$. Provide a counter-example to show that (\mathbb{Z}_n^*, \cdot) is not a group in general.

Find a condition such that (\mathbb{Z}_n^*, \cdot) is a commutative group, compute the neutral element, give a closed form for the inverse of any element and prove the commutative group axioms.

4.1.1 Finite groups

As we have seen in the previous examples, groups can either contain infinitely many elements (such as integers) or finitely many elements (as for example the remainder class groups $(\mathbb{Z}_n, +)$). To capture this distinction, a group is called a **finite group** if the underlying set of elements is finite. In that case, the number of elements of that group is called its **order**.

Notation and Symbols 5. Let \mathbb{G} be a finite group. We write $\text{ord}(\mathbb{G})$ or $|\mathbb{G}|$ for the order of \mathbb{G} .

Example 32. Consider the remainder class groups $(\mathbb{Z}_6, +)$ from example 9, the group $(\mathbb{Z}_5, +)$ from example 14, and the group (\mathbb{Z}_5^*, \cdot) from exercise 32. We can easily see that the order of $(\mathbb{Z}_6, +)$ is 6, the order of $(\mathbb{Z}_5, +)$ is 5 and the order of (\mathbb{Z}_5^*, \cdot) is 4.

Exercise 34. Let $n \in \mathbb{N}$ with $n \geq 2$ be some modulus. What is the order of the remainder class group $(\mathbb{Z}_n, +)$?

4.1.2 Generators

The set of elements of a group can be complicated, and it is not always obvious how to actually compute elements of a given group. From a practical point of view, it is therefore desirable to have groups with a **generator set**. This is a small subset of elements from which all other elements can be generated by applying the group law repeatedly to only the elements of the generator set or their inverses.

Of course, every group \mathbb{G} has a trivial set of generators, when we just consider every element of the group to be in the generator set. The more interesting question is to find smallest possible generator set for a given group. Of particular interest in this regard are groups that have a generator set that contains a single element only. In this case, there exists a (not necessarily unique) element $g \in \mathbb{G}$ such that every other element from \mathbb{G} can be computed by the repeated combination of g and its inverse g^{-1} only.

Definition 4.1.2.1 (Cyclic groups). Groups with single, not necessarily unique, generators are called **cyclic groups** and any element $g \in \mathbb{G}$ that is able to generate \mathbb{G} is called a **generator**.

Example 33. The most basic example of a cyclic group is the group of integers with integer addition $(\mathbb{Z}, +)$. In this case, the number 1 is a generator of \mathbb{Z} , since every integer can be obtained by repeatedly adding either 1 or its inverse -1 to itself. For example, -4 is generated by 1, since $-4 = -1 + (-1) + (-1) + (-1)$. Another generator of \mathbb{Z} is the number -1 .

Example 34. Consider the group (\mathbb{Z}_5^*, \cdot) from exercise 32. Since $2^1 = 2$, $2^2 = 4$, $2^3 = 3$ and $2^4 = 1$, the element 2 is a generator of (\mathbb{Z}_5^*, \cdot) . Moreover since $3^1 = 3$, $3^2 = 4$, $3^3 = 2$ and $3^4 = 1$, the element 3 is another generator of (\mathbb{Z}_5^*, \cdot) . Cyclic groups can therefore have more than one generator. However since $4^1 = 4$, $4^2 = 1$, $4^3 = 4$ and in general $4^k = 4$ for k odd and $4^k = 1$ for k even the element 4 is not a generator of (\mathbb{Z}_5^*, \cdot) . It follows that in general not every element of a finite cyclic group is a generator.

Example 35. Consider a modulus n and the remainder class groups $(\mathbb{Z}_n, +)$ from exercise 34. These groups are cyclic, with generator 1, since every other element of that group can be constructed by repeatedly adding the remainder class 1 to itself. Since \mathbb{Z}_n is also finite, we know that $(\mathbb{Z}_n, +)$ is a finite cyclic group of order n .

Exercise 35. Consider the group $(\mathbb{Z}_6, +)$ of modular 6 addition from example 9. Show that $5 \in \mathbb{Z}_6$ is a generator, and then show that $2 \in \mathbb{Z}_6$ is not a generator.

Exercise 36. Let $p \in \mathbb{P}$ be prime number and (\mathbb{Z}_p^*, \cdot) the finite group from exercise 33. Show that (\mathbb{Z}_p^*, \cdot) is cyclic.

4.1.3 The exponential map

Observe that, when \mathbb{G} is a cyclic group of order n and $g \in \mathbb{G}$ is a generator of \mathbb{G} , then there exists a so-called **exponential map**, which maps the additive group law of the remainder class group $(\mathbb{Z}_n, +)$ onto the group law of \mathbb{G} in a one-to-one correspondence. The exponential map can be formalized as in (4.1) below (where g^x means “multiply g by itself x times” and $g^0 = e_{\mathbb{G}}$).

$$g^{(\cdot)} : \mathbb{Z}_n \rightarrow \mathbb{G} \quad x \mapsto g^x \quad (4.1)$$

To see how the exponential map works, first observe that, since $g^0 := e_{\mathbb{G}}$ by definition, the neutral element of \mathbb{Z}_n is mapped to the neutral element of \mathbb{G} . Furthermore, since $g^{x+y} = g^x \cdot g^y$, the map respects the group law.

Notation and Symbols 6 (Scalar multiplication). If a group $(\mathbb{G}, +)$ is written in additive notation (Notation and Symbols 4), then the exponential map is often called **scalar multiplication**, and written as follows:

$$(\cdot) \cdot g : \mathbb{Z}_n \rightarrow \mathbb{G} \quad ; \quad x \mapsto x \cdot g \quad (4.2)$$

In this notation, the symbol $x \cdot g$ is defined as “add the generator g to itself x times” and the symbol $0 \cdot g$ is defined to be the neutral element in \mathbb{G} .

Cryptographic applications often utilize finite cyclic groups of a very large order n , which means that computing the exponential map by repeated multiplication of the generator with itself is infeasible for very large remainder classes. Algorithm 5, called **square and multiply**, solves this problem by computing the exponential map in approximately k steps, where k is the bit length of the exponent (3.4):

Because the exponential map respects the group law, it doesn't matter if we do our computation in \mathbb{Z}_n before we write the result into the exponent of g or afterwards: the result will be the same in both cases. The latter method is usually referred to as doing computations “in the exponent”. In cryptography in general, and in SNARK development in particular, we often perform computations “in the exponent” of a generator.

unify title
of Alg
with text

move 3.4
here

check
algorithm
floating

doesn't

Algorithm 5 Cyclic Group Exponentiation**Require:** g group generator of order n **Require:** $x \in \mathbb{Z}_n$ **procedure** EXPONENTIATION(g, x)Let (b_0, \dots, b_k) be a binary representation of x

▷ see example XXX

 $h \leftarrow g$ $y \leftarrow e_{\mathbb{G}}$ **for** $0 \leq j < k$ **do** **if** $b_j = 1$ **then** $y \leftarrow y \cdot h$

▷ multiply

end if $h \leftarrow h \cdot h$

▷ square

end for**return** y **end procedure****Ensure:** $y = g^x$

Example 36. Consider the multiplicative group (\mathbb{Z}_5^*, \cdot) from exercise 32. We know from 36 that \mathbb{Z}_5^* is a cyclic group of order 4, and that the element $3 \in \mathbb{Z}_5^*$ is a generator. This means that we also know that the following map respects the group law of addition in \mathbb{Z}_4 and the group law of multiplication in \mathbb{Z}_5^* :

$$3^{(\cdot)} : \mathbb{Z}_4 \rightarrow \mathbb{Z}_5^* ; x \mapsto 3^x$$

To do an example computation “in the exponent” of 3, let’s perform the calculation $1 + 3 + 2$ in the exponent of the generator 3:

$$3^{1+3+2} = 3^2 \tag{4.3}$$

$$= 4 \tag{4.4}$$

1460 In (4.3) above, we first performed the computation $1 + 3 + 2 = 1$ in the remainder class group
 1461 $(\mathbb{Z}_4, +)$ and then applied the exponential map $3^{(\cdot)}$ to the result in (4.4).

 should be
2?

1462 However, since the exponential map (4.1) respects the group law, we also could map each
 1463 summand into (\mathbb{Z}_5^*, \cdot) first and then apply the group law of (\mathbb{Z}_5^*, \cdot) . The result is guaranteed to be
 1464 the same:

$$\begin{aligned} 3^1 \cdot 3^3 \cdot 3^2 &= 3 \cdot 2 \cdot 4 \\ &= 1 \cdot 4 \\ &= 4 \end{aligned}$$

1465 Since the exponential map (4.1) is a one-to-one correspondence that respects the group law,
 1466 it can be shown that this map has an inverse with respect to the base g , called the **base g discrete**
 1467 **logarithm map**:

$$\log_g(\cdot) : \mathbb{G} \rightarrow \mathbb{Z}_n \ x \mapsto \log_g(x) \tag{4.5}$$

1468 Discrete logarithms are highly important in cryptography, because there are finite cyclic groups
 1469 where the exponential map and its inverse, the discrete logarithm map, are believed to be one-
 1470 way functions, which informally means that computing the exponential map is fast, while com-
 1471 puting the logarithm map is slow (We will look into a more precise definition in 4.1.6).

Example 37. Consider the exponential map $3^{(\cdot)}$ from example 36. Its inverse is the discrete logarithm to the base 3, given by the map below:

$$\log_3(\cdot) : \mathbb{Z}_5^* \rightarrow \mathbb{Z}_4 \quad x \mapsto \log_3(x)$$

In contrast to the exponential map $3^{(\cdot)}$, we have no way to actually compute this map, other than by trying all elements of the group until we find the correct one. For example, in order to compute $\log_3(4)$, we have to find some $x \in \mathbb{Z}_4$ such that $3^x = 4$, and all we can do is repeatedly insert elements x into the exponent of 3 until the result is 4. To do this, let's write down all the images of $3^{(\cdot)}$:

$$3^0 = 1, \quad 3^1 = 3, \quad 3^2 = 4, \quad 3^3 = 2$$

Since the discrete logarithm $\log_3(\cdot)$ is defined as the inverse to this function, we can use those images to compute the discrete logarithm:

$$\log_3(1) = 0, \quad \log_3(2) = 3, \quad \log_3(3) = 1, \quad \log_3(4) = 2$$

Note that this computation was only possible, because we were able to write down all images of the exponential map. However, in real world applications the groups in consideration are too large to write down the images of the exponential map.

Exercise 37 (Efficient Scalar Multiplication). Let $(\mathbb{G}, +)$ be a finite cyclic group of order n . Consider algorithm 5 and define its analog for groups in additive notation.

4.1.4 Factor Groups

As we know from the fundamental theorem of arithmetic (3.7), every natural number n is a product of factors, the most basic of which are prime numbers. This parallels subgroups of finite cyclic groups in an interesting way.

Definition 4.1.4.1 (The fundamental theorem of finite cyclic groups). If \mathbb{G} is a finite cyclic group of order n , then every subgroup \mathbb{G}' of \mathbb{G} is finite and cyclic, and the order of \mathbb{G}' is a factor of n . Moreover for each factor k of n , \mathbb{G} has exactly one subgroup of order k . This is known as the **fundamental theorem of finite cyclic groups**.

Notation and Symbols 7. If \mathbb{G} is a finite cyclic group of order n and k is a factor of n , then we write $\mathbb{G}[k]$ for the unique finite cyclic group which is the order k subgroup of \mathbb{G} , and call it a **factor group** of \mathbb{G} .

One particularly interesting situation occurs if the order of a given finite cyclic group is a prime number. As we know from the fundamental theorem of arithmetics (3.7), prime numbers have only two factors: the number 1 and the prime number itself. It then follows from the fundamental theorem of finite cyclic groups (definition 4.1.4.1) that those groups have no subgroups other than the trivial group (example 30) and the group itself.

Cryptographic protocols often assume the existence of finite cyclic groups of prime order. However some real-world implementations of those protocols are not defined on prime order groups, but on groups where the order consist of a (usually large) prime number that has small cofactors (see Notation and Symbols 1). In this case, a method called **cofactor clearing** has to be applied to ensure that the computations are not done in the group itself but in its (large) prime order subgroup.

To understand cofactor clearing in detail, let \mathbb{G} be a finite cyclic group of order n , and let k be a factor of n with associated factor group $\mathbb{G}[k]$. We can project any element $g \in \mathbb{G}[k]$ onto the neutral element e of \mathbb{G} by multiplying g k -times with itself:

$$g^k = e \tag{4.6}$$

Consequently, if $c := n \operatorname{div} k$ is the cofactor of k in n , then any element from the full group $g \in \mathbb{G}$ can be projected into the factor group $\mathbb{G}[k]$ by multiplying g c -times with itself. This defines the following map, which is often called **cofactor clearing** in cryptographic literature:

$$(\cdot)^c : \mathbb{G} \rightarrow \mathbb{G}[k] : g \mapsto g^c \quad (4.7)$$

Example 38. Consider the finite cyclic group (\mathbb{Z}_5^*, \cdot) from example 34. Since the order of \mathbb{Z}_5^* is 4, and 4 has the factors 1, 2 and 4, it follows from the fundamental theorem of finite cyclic groups (Definition 4.1.4.1) that \mathbb{Z}_5^* has 3 unique subgroups. In fact, the unique subgroup $\mathbb{Z}_5^*[1]$ of order 1 is given by the trivial group $\{1\}$ that contains only the multiplicative neutral element 1. The unique subgroup $\mathbb{Z}_5^*[4]$ of order 4 is \mathbb{Z}_5^* itself, since, by definition, every group is trivially a subgroup of itself. The unique subgroup $\mathbb{Z}_5^*[2]$ of order 2 is more interesting, and is given by the set $\mathbb{Z}_5^*[2] = \{1, 4\}$.

Since \mathbb{Z}_5^* is not a prime order group, and, since the only prime factor of 4 is 2, the “large” prime order subgroup of \mathbb{Z}_5^* is $\mathbb{Z}_5^*[2]$. Moreover, since the cofactor of 2 in 4 is also 2, we get the cofactor clearing map $(\cdot)^2 : \mathbb{Z}_5^* \rightarrow \mathbb{Z}_5^*[2]$. As expected, when we apply this map to all elements of \mathbb{Z}_5^* , we see that it maps onto the elements of $\mathbb{Z}_5^*[2]$ only:

$$1^2 = 1 \quad 2^2 = 4 \quad 3^2 = 4 \quad 4^2 = 1 \quad (4.8)$$

We can therefore use this map to “clear the cofactor” of any element from \mathbb{Z}_5^* , which means that the element is projected onto the “large” prime order subgroup $\mathbb{Z}_5^*[2]$.

Exercise 38. Consider the previous example 38, and show that $\mathbb{Z}_5^*[2]$ is a commutative group.

Exercise 39. Consider the finite cyclic group $(\mathbb{Z}_6, +)$ of modular 6 addition from example 35. Describe all subgroups of $(\mathbb{Z}_6, +)$. Identify the large prime order subgroup of \mathbb{Z}_6 , define its cofactor clearing map and apply that map to all elements of \mathbb{Z}_6 .

Exercise 40. Let (\mathbb{Z}_p^*, \cdot) be the cyclic group from exercise 36. Show that, for $p \geq 5$, not every element $x \in \mathbb{F}_p^*$ is a generator of \mathbb{F}_p^* .

4.1.5 Pairings

Of particular importance for the development of SNARKs are so-called **pairing maps** on commutative groups, defined below.

Definition 4.1.5.1 (Pairing map). Let \mathbb{G}_1 , \mathbb{G}_2 and \mathbb{G}_3 be three commutative groups. Then a **pairing map** is a function

$$e(\cdot, \cdot) : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3 \quad (4.9)$$

This function takes pairs (g_1, g_2) of elements from \mathbb{G}_1 and \mathbb{G}_2 , and maps them to elements from \mathbb{G}_3 such that the **bilinearity** property holds, which means that for all $g_1, g'_1 \in \mathbb{G}_1$ and $g_2, g'_2 \in \mathbb{G}_2$ the following two identities are satisfied:

$$e(g_1 \cdot g'_1, g_2) = e(g_1, g_2) \cdot e(g'_1, g_2) \quad \text{and} \quad e(g_1, g_2 \cdot g'_2) = e(g_1, g_2) \cdot e(g_1, g'_2) \quad (4.10)$$

Informally speaking, bilinearity means that it doesn’t matter if we first execute the group law on one side and then apply the bilinear map, or if we first apply the bilinear map and then apply the group law in \mathbb{G}_3 .

A pairing map is called **non-degenerate** if, whenever the result of the pairing is the neutral element in \mathbb{G}_3 , one of the input values is the neutral element of \mathbb{G}_1 or \mathbb{G}_2 . To be more precise, $e(g_1, g_2) = e_{\mathbb{G}_3}$ implies $g_1 = e_{\mathbb{G}_1}$ or $g_2 = e_{\mathbb{G}_2}$.

1538 *Example 39.* One of the most basic examples of a non-degenerate pairing involves the groups
 1539 \mathbb{G}_1 , \mathbb{G}_2 and \mathbb{G}_3 all being groups of integers with addition $(\mathbb{Z}, +)$. In this case, the following
 1540 map defines a non-degenerate pairing:

$$e(\cdot, \cdot) : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z} \quad (a, b) \mapsto a \cdot b \quad (4.11)$$

1541 Note that bilinearity follows from the distributive law of integers, meaning that, for $a, b, c \in$
 1542 \mathbb{Z} , the equation $e(a + b, c) = (a + b) \cdot c = a \cdot c + b \cdot c = e(a, c) + e(b, c)$ holds (and the same
 1543 reasoning is true for the second argument b). b

1544 To see that $e(\cdot, \cdot)$ is non-degenerate, assume that $e(a, b) = 0$. Then $a \cdot b = 0$ implies that a or
 1545 b must be zero.

1546 *Exercise 41* (Arithmetic laws for pairing maps). Let \mathbb{G}_1 , \mathbb{G}_2 and \mathbb{G}_3 be finite cyclic groups of
 1547 the same order n , and let $e(\cdot, \cdot) : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ be a pairing map. Show that, for given $g_1 \in \mathbb{G}_1$,
 1548 $g_2 \in \mathbb{G}_2$ and all $a, b \in \mathbb{Z}_n$, the following identity holds:

$$e(g_1^a, g_2^b) = e(g_1, g_2)^{a \cdot b} \quad (4.12)$$

1549 *Exercise 42.* Consider the remainder class groups $(\mathbb{Z}_n, +)$ from example 34 for some modulus
 1550 n . Show that the following map is a pairing map.

$$e(\cdot, \cdot) : \mathbb{Z}_n \times \mathbb{Z}_n \rightarrow \mathbb{Z}_n \quad (a, b) \mapsto a \cdot b \quad (4.13)$$

1551 Why is the pairing not non-degenerate in general, and what condition must be imposed on
 1552 n such that the pairing will be non-degenerate?

1553 4.1.6 Cryptographic Groups

1554 In this section, we look at classes of groups that are believed to satisfy certain **computational**
 1555 **hardness assumptions**, meaning that it is not feasible to compute them in polynomial time.

1556 *Example 40.* To give an example for a well-known computational hardness assumption, con-
 1557 sider the problem of factorization, i.e. computing the prime factors of a composite integer (see
 1558 example 1). If the prime factors are very large, this is infeasible to do, and is expected to remain
 1559 infeasible. We assume the problem is **computationally hard** or **infeasible**.

1560 Note that, in example 40, we say that the problem is infeasible to solve if the prime factors
 1561 are large enough. Naturally, this is made more precise in the cryptographic standard model,
 1562 where we have a security parameter, and we say that “there exists a security parameter such that
 1563 it is not feasible to compute a solution to the problem”. In the following examples, the security
 1564 parameter roughly correlates with the order of the group in consideration. In this book, we
 1565 do not include the security parameter in our definitions, since we only aim to provide an intu-
 1566 itive understanding of the cryptographic assumptions, not teach the ability to perform rigorous
 1567 analysis.

1568 Furthermore, understand that these are **assumptions**. Academics have been looking for
 1569 efficient prime factorization algorithms for a long time, and they have been getting better and
 1570 better while computers have become faster and faster – but there always was a higher security
 1571 parameter for which the problem still was infeasible.

1572 In what follows, we describe a few problems arising in the context of groups in cryptography
 1573 that are assumed to be infeasible. We will refer to them throughout the book.

we say
that

4.1.6.1 The discrete logarithm assumption

The so-called **discrete logarithm problem (DLP)** is one of the most fundamental assumptions in cryptography.

Definition 4.1.6.1. Let \mathbb{G} be a finite cyclic group of order r and let g be a generator of \mathbb{G} . We know from (4.1) that there is an exponential map $g^{(\cdot)} : \mathbb{Z}_r \rightarrow \mathbb{G} ; x \mapsto g^x$ that maps the residue classes from modulo r arithmetic onto the group in a 1 : 1 correspondence. The **discrete logarithm problem** is the task of finding an inverse to this map, that is, to find a solution $x \in \mathbb{Z}_r$ to the following equation for some given $h, g \in \mathbb{G}$:

$$h = g^x \quad (4.14)$$

There are groups in which the DLP is assumed to be infeasible to solve, and there are groups in which it isn't. We call the former group **DL-secure** groups.

Rephrasing the previous definition, it is believed that, in DL-secure groups, there is a number n such that it is infeasible to compute some number x that solves the equation $h = g^x$ for a given h and g , assuming that the order of the group n is large enough. The number n here corresponds to the security parameter discussed above.

Example 41 (Public key cryptography). One the most basic examples of an application for DL-secure groups is in public key cryptography, where the parties publicly agree on some pair (\mathbb{G}, g) such that \mathbb{G} is a finite cyclic group of appropriate order n , believed to be a DL-secure group, and g is a generator of \mathbb{G} .

In this setting, a secret key is some number $sk \in \mathbb{Z}_r$ and the associated public key pk is the group element $pk = g^{sk}$. Since discrete logarithms are assumed to be hard, it is infeasible for an attacker to compute the secret key from the public key, as this would involve finding solutions x to the following equation (which is believed to be infeasible):

$$pk = g^x \quad (4.15)$$

As the example 41 shows, identifying DL-secure groups is an important practical problem. Unfortunately, it is easy to see that it does not make sense to assume the hardness of the discrete logarithm problem in all finite cyclic groups: counterexamples are common and easy to construct.

mention a few examples

4.1.6.2 The decisional Diffie–Hellman assumption

Let \mathbb{G} be a finite cyclic group of order n and let g be a generator of \mathbb{G} . The decisional Diffie–Hellman (DDH) problem is to distinguish (g^a, g^b, g^{ab}) from the triple (g^a, g^b, g^c) for uniformly random values $a, b, c \in \mathbb{Z}_r$. If we assume the DDH problem is infeasible to solve in \mathbb{G} , we call \mathbb{G} a **DDH-secure** group.

DDH-security is a stronger assumption than DL-security 4.1.6.1, in the sense that if the DDH problem is infeasible, so is the DLP, but not necessarily the other way around.

To see why this is the case, assume that the discrete logarithm assumption does not hold. In that case, given a generator g and a group element h , it is easy to compute some element $x \in \mathbb{Z}_p$ with $h = g^x$. Then the decisional Diffie–Hellman assumption cannot hold, since given some triple (g^a, g^b, z) , one could efficiently decide whether $z = g^{ab}$ is true by first computing the discrete logarithm b of g^b , then computing $g^{ab} = (g^a)^b$ and deciding whether or not $z = g^{ab}$.

On the other hand, the following example shows that there are groups where the discrete logarithm assumption holds but the decisional Diffie–Hellman assumption does not.

Example 42 (Efficiently computable bilinear pairings). Let \mathbb{G} be a DL-secure, finite, cyclic group of order r with generator g and \mathbb{G}_T another group such that there is an efficiently computable pairing map $e(\cdot, \cdot) : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ that is bilinear and non degenerate 4.9.

In a setting like this, it is easy to show that solving DDH cannot be infeasible, since given some triple (g^a, g^b, z) , it is possible to efficiently check whether $z = g^{ab}$ by making use of the pairing:

$$e(g^a, g^b) \stackrel{?}{=} e(g, z) \quad (4.16)$$

Since the bilinearity properties of $e(\cdot, \cdot)$ imply $e(g^a, g^b) = e(g, g)^{ab} = e(g, g^{ab})$, and $e(g, y) = e(g, y')$ implies $y = y'$ due to the non-degenerate property, the equality means $z = g^{ab}$.

It follows that the DDH assumption is indeed stronger than the discrete log assumption, and groups with efficient pairings cannot be DDH-secure groups.

4.1.6.3 The computational Diffie–Hellman assumption

Let \mathbb{G} be a finite cyclic group of order n and let g be a generator of \mathbb{G} . The computational Diffie–Hellman assumption stipulates that, given randomly and independently chosen elements $a, b \in \mathbb{Z}_r$, it is not possible to compute g^{ab} if only g, g^a and g^b (but not a and b) are known. If this is the case for \mathbb{G} , we call \mathbb{G} a **CDH-secure** group.

In general, we don't know if CDH-security is a stronger assumption than DL-security, or if both assumptions are equivalent. We know that DL-security is necessary for CDH-security, but the other direction is currently not well understood. In particular, there are no DL-security groups known that are not also CDH-secure.

To see why the discrete logarithm assumption is necessary, assume that it does not hold. So, given a generator g and a group element h , it is easy to compute some element $x \in \mathbb{Z}_p$ with $h = g^x$. In that case, the computational Diffie–Hellman assumption cannot hold, since, given g, g^a and g^b , one can efficiently compute b and hence is able to compute $g^{ab} = (g^a)^b$ from this data.

The computational Diffie–Hellman assumption is a weaker assumption than the decisional Diffie–Hellman assumption. This means that there are groups where CDH holds and DDH does not hold, while there cannot be groups in which DDH holds but CDH does not hold. To see that, assume that it is efficiently possible to compute g^{ab} from g, g^a and g^b . Then, given (g^a, g^b, z) it is easy to decide whether $z = g^{ab}$ holds or not.

Several variations and special cases of CDH exist. For example, the **square computational Diffie–Hellman assumption** assumes that, given g and g^x , it is computationally hard to compute g^{x^2} . The **inverse computational Diffie–Hellman assumption** assumes that, given g and g^x , it is computationally hard to compute $g^{x^{-1}}$.

4.1.7 Hashing to Groups

4.1.7.1 Hash functions

Generally speaking, a hash function is any function that can be used to map data of arbitrary size to fixed-size values. Since binary strings of arbitrary length are a way to represent data in general, we can understand a **hash function** as the following map where $\{0, 1\}^*$ represents the set of all binary strings of arbitrary but finite length and $\{0, 1\}^k$ represents the set of all binary strings that have a length of exactly k bits:

$$H : \{0, 1\}^* \rightarrow \{0, 1\}^k \quad (4.17)$$

The **images** of H , that is, the values returned by the hash function H , are called **hash values**, **digests**, or simply **hashes**.

Notation and Symbols 8. In what follows we call an element $b \in \{0, 1\}$ a **bit**. If $s \in \{0, 1\}^*$ is a binary string, we write $|s| = k$ for its **length**, that is for the number of bits in s . We write $\langle \rangle$ for the empty binary string and $s = \langle b_1, b_2, \dots, b_k \rangle$ for a binary string of length k .

If two binary strings $s = \langle b_1, b_2, \dots, b_k \rangle$ and $s' = \langle b'_1, b'_2, \dots, b'_l \rangle$ are given then we write $s||s'$ for the **concatenation** that is the string $s||s' = \langle b_1, b_2, \dots, b_k, b'_1, b'_2, \dots, b'_l \rangle$.

If H is a hash function that maps binary strings of arbitrary length onto binary strings of length k and if $s \in \{0, 1\}^*$ is a binary string, we write $H(s)_j$ for the bit at position j in the image $H(s)$.

Example 43 (k -truncation hash). One of the most basic hash functions $H_k : \{0, 1\}^* \rightarrow \{0, 1\}^k$ is given by simply truncating every binary string s of size $|s| > k$ to a string of size k and by filling any string s' of size $|s'| < k$ with zeros. To make this hash function deterministic, we define that both truncation and filling should happen “on the left”.

For example, if the parameter k is given by $k = 3$ and $s_1 = \langle 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0 \rangle$ as well as $s_2 = 1$, then $H_3(s_1) = \langle 1, 1, 0 \rangle$ and $H_3(s_2) = \langle 0, 0, 1 \rangle$.

A desirable property of a hash function is **uniformity**, which means that it should map input values as evenly as possible over its output range. In mathematical terms, every string of length k from $\{0, 1\}^k$ should be generated with roughly the same probability.

Of particular interest are so-called **cryptographic** hash functions, which are hash functions that are also **one-way functions**, which essentially means that, given a string y from $\{0, 1\}^k$ it is infeasible to find a string $x \in \{0, 1\}^*$ such that $H(x) = y$ holds. This property is usually called **preimage-resistance**.

Moreover, if a string $x_1 \in \{0, 1\}^*$ is given, then it should be infeasible to find another string $x_2 \in \{0, 1\}^*$ with $x_1 \neq x_2$ and $H(x_1) = H(x_2)$.

In addition, it should be infeasible to find two strings $x_1, x_2 \in \{0, 1\}^*$ such that $H(x_1) = H(x_2)$, which is called **collision resistance**. It is important to note, though, that collisions always exist, since a function $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ inevitably maps infinitely many values onto the same hash. In fact, for any hash function with digests of length k , finding a preimage to a given digest can always be done using a brute force search in 2^k evaluation steps. It should just be practically impossible to compute those values, and statistically very unlikely to generate two of them by chance.

A third property of a cryptographic hash function is that small changes in the input string, like changing a single bit, should generate hash values that look completely different from each other. This is called **diffusion** or the avalanche effect.

Because cryptographic hash functions map tiny changes in input values onto large changes in the output, implementation errors that change the outcome are usually easy to spot by comparing them to expected output values. The definitions of cryptographic hash functions are therefore usually accompanied by some test vectors of common inputs and expected digests. Since the empty string $\langle \rangle$ is the only string of length 0, a common test vector is the expected digest of the empty string.

Example 44 (k -truncation hash). Consider the k -truncation hash from example 43. Since the empty string has length 0, it follows that the digest of the empty string is the string of length k that only contains 0's:

$$H_k(\langle \rangle) = \langle 0, 0, \dots, 0, 0 \rangle \quad (4.18)$$

It is pretty obvious from the definition of H_k that this simple hash function is not a cryptographic hash function. In particular, every digest is its own preimage, since $H_k(y) = y$ for every string

of size exactly k . Finding preimages is therefore easy, so the property of preimage resistance does not hold.

In addition, it is easy to construct collisions as all strings s of size $|s| > k$ that share the same k -bits “on the right” are mapped to the same hash value, so this function is not collision resistant, either.

Finally, this hash function does not have a lot of diffusion, as changing bits that are not part of the k -right-most bits don’t change the digest at all.

Computing cryptographically secure hash functions in pen-and-paper style is possible but tedious. Fortunately, Sage can import the **hashlib** library, which is intended to provide a reliable and stable base for writing Python programs that require cryptographic functions. The following examples explain how to use hashlib in Sage.

Example 45. An example of a hash function that is generally believed to be a cryptographically secure hash function is the so-called **SHA256** hash, which, in our notation, is a function that maps binary strings of arbitrary length onto binary strings of length 256:

$$SHA256 : \{0, 1\}^* \rightarrow \{0, 1\}^{256} \quad (4.19)$$

To evaluate a proper implementation of the *SHA256* hash function, the digest of the empty string is supposed to be

$$SHA256(<>) = e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855 \quad (4.20)$$

For better human readability, it is common practice to represent the digest of a string not in its binary form, but in a hexadecimal representation. We can use Sage to compute *SHA256* and freely transit between binary, hexadecimal and decimal representations. To do so, we import hashlib’s implementation of *SHA256*:

```
sage: import hashlib 159
sage: test = 'e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934 160
ca495991b7852b855'
sage: empty_string = "" 161
sage: binary_string = empty_string.encode() 162
sage: hasher = hashlib.sha256(binary_string) 163
sage: result = hasher.hexdigest() 164
sage: type(result) # sage represents digests as strings 165
<class 'str'> 166
sage: d = ZZ('0x'+ result) # conversion to an integer 167
sage: d.str(16) == test # hash is equal to test vector 168
True 169
sage: d.str(16) # hexadecimal representation 170
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b8 171
55
sage: d.str(2) # binary representation 172
11100011101100001100010001000010100110001111110000011100000101 173
001001101011111011111010011001000100110010110111101110010
01001000010011110101110010000011110010001100100100110111001
00110100110010100100100101011001100100011011011110000101001
01011100001010101 174
sage: d.str(10) # decimal representation 174
```

1742 10298733624955409702953521232258132278979990064819803499337939 175
 1743 7001115665086549

1744 4.1.7.2 Hashing to cyclic groups

1745 As we have seen in the previous paragraph, general hash functions map binary strings of ar-
 1746 bitrary length onto binary strings of some fixed length. However, it is desirable in various
 1747 cryptographic primitives to not simply hash to binary strings of fixed length but to hash into al-
 1748 gebraic structures like groups, while keeping (some of) the properties like preimage resistance
 1749 or collision resistance.

1750 Hash functions like this can be defined for various algebraic structures, but, in a sense, the
 1751 most fundamental ones are hash functions that map into groups, because they can be easily
 1752 extended to map into other structures like rings or fields.

1753 To give a more precise definition, let \mathbb{G} be a group and $\{0, 1\}^*$ the set of all finite, binary
 1754 strings, then a **hash-to-group** function is a deterministic map

$$H : \{0, 1\}^* \rightarrow \mathbb{G} \quad (4.21)$$

1755 As the following example shows, hashing to finite cyclic groups can be trivially achieved
 1756 for the price of some undesirable properties of the hash function:

1757 *Example 46* (Naive cyclic group hash). Let \mathbb{G} be a finite cyclic group of order n . If the task is to
 1758 implement a hash-to-group function, one immediate approach can be based on the observation
 1759 that binary strings of size k can be interpreted as integers $z \in \mathbb{Z}$ in the range $0 \leq z < 2^k$ using
 1760 equation 3.4.

1761 To be more precise, let $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ be a hash function for some parameter k , g a
 1762 generator of \mathbb{G} and $s \in \{0, 1\}^*$ a binary string. Using equation 3.4 and notation 8 the following
 1763 expression is a non negative integer:

$$z_{H(s)} = H(s)_0 \cdot 2^0 + H(s)_1 \cdot 2^1 + \dots + H(s)_k \cdot 2^k \quad (4.22)$$

1764 A hash-to-group function for the group \mathbb{G} can then be defined as a composition of the expo-
 1765 nential map $g^{(\cdot)}$ of g with the interpretation of $H(s)$ as an integer:

$$H_g : \{0, 1\}^* \rightarrow \mathbb{G} : s \mapsto g^{z_{H(s)}} \quad (4.23)$$

1766 Constructing a hash-to-group function like this is easy for cyclic groups, and it might be
 1767 good enough in certain applications. It is, however, almost never adequate in cryptographic
 1768 applications, as a discrete log relation might be constructible between some hash values $H_g(s)$
 1769 and $H_g(t)$, regardless of whether or not \mathbb{G} is DL-secure 4.1.6.1.

a few ex-
amples?

To be more precise a discrete log relation between the group elements $H_g(s)$ and $H_g(t)$ is
 any element $x \in \mathbb{Z}_n$ such that $H_g(s) = H_g(t)^x$. To see how such an x can be constructed, assume
 that $z_{H(s)}$ has a multiplicative inverse in \mathbb{Z}_n . In this case, the element $x = z_{H(t)} \cdot z_{H(s)}^{-1}$ from \mathbb{Z}_n is
 a discrete log relation between $H_g(s)$ and $H_g(t)$ since:

$$\begin{aligned} g^{z_{H(t)}} &= g^{z_{H(t)}} && \Leftrightarrow \\ g^{z_{H(t)}} &= g^{z_{H(t)} \cdot z_{H(s)} \cdot z_{H(s)}^{-1}} && \Leftrightarrow \\ g^{z_{H(t)}} &= g^{z_{H(s)} \cdot x} && \Leftrightarrow \\ H_g(t) &= (H_g(s))^x \end{aligned}$$

Therefore applications where discrete log relations between hash values are undesirable need different approaches. Many of these approaches start with a way to hash into the set \mathbb{Z}_r of modular r arithmetics.

4.1.7.3 Pedersen Hashes

The so-called **Pedersen hash function** [Pedersen, 1992] provides a way to map fixed size tuples of elements from modular arithmetics onto elements of finite cyclic groups in such a way that discrete log relations between different images are avoidable. Compositions of a Pedersen hash with a general hash function 4.17, then provide hash-to-group functions that maps strings of arbitrary length onto group elements.

To be more precise, let j be an integer, \mathbb{G} a finite cyclic group of order n and $\{g_1, \dots, g_j\} \subset \mathbb{G}$ a uniform and randomly generated set of generators of \mathbb{G} . Then **Pedersen's hash function** is defined as follows:

$$H_{\{g_1, \dots, g_j\}} : (\mathbb{Z}_r)^j \rightarrow \mathbb{G} : (x_1, \dots, x_j) \mapsto \prod_{i=1}^j g_i^{x_i} \quad (4.24)$$

It can be shown that Pedersen's hash function is collision-resistant under the assumption that \mathbb{G} is DL-secure 4.1.6.1. It is important to note though, that the following familie of functions does not qualify as a **pseudorandom function family**.

$$\{H_{\{g_1, \dots, g_j\}} \mid g_1, \dots, g_j \in \mathbb{G}\} \quad (4.25)$$

From an implementation perspective, it is important to derive the set of generators $\{g_1, \dots, g_k\}$ in such a way that they are as uniform and random as possible. In particular, any known discrete log relation between two generators, that is, any known $x \in \mathbb{Z}_n$ with $g_h = (g_i)^x$ must be avoided.

Example 47. To compute an actual Pedersen's hash, consider the cyclic group \mathbb{Z}_5^* from example 34. We know from example 38, that the elements 2 and 3 are generators of \mathbb{Z}_5^* and it follows that the following map is a Pedersen's hash function:

$$H_{\{2,3\}} : \mathbb{Z}_4 \times \mathbb{Z}_4 \rightarrow \mathbb{Z}_5^* ; (x, y) \mapsto 2^x \cdot 3^y$$

To see how this map can be calculated, we choose the input value $(1, 3)$ from $\mathbb{Z}_4 \times \mathbb{Z}_4$. Then $H_{\{2,3\}}(1, 3) = 2^1 \cdot 3^3 = 2 \cdot 2 = 4$.

To see how the composition of a hash function with $H_{\{2,3\}}$ defines a hash-to-group function, consider the *SHA256* hash function from example 45. Given some binary string $s \in \{0, 1\}^*$, we can insert the two least significant bits $SHA256(s)_0$ and $SHA256(s)_1$ from the image $SHA256(s)$ into $H_{\{2,3\}}$ to get an element in \mathbb{F}_5^* . This defines the following hash-to-group function

$$SHA256_H_{\{2,3\}} : \{0, 1\}^* \rightarrow \mathbb{Z}_5^* ; s \mapsto 2^{SHA256(s)_0} \cdot 3^{SHA256(s)_1}$$

To see how this hash function can be calculated, consider the empty string $\langle \rangle$. Since we know from the sage computation in example 45, that $SHA256(\langle \rangle)_0 = 1$ as well as $SHA256(\langle \rangle)_1 = 0$, we get $SHA_256H_{\{2,3\}}(\langle \rangle) = 2^1 \cdot 3^0 = 2$.

Of course computing $SHA256_H_{\{2,3\}}$ in a pen and paper style is difficult. However we can easily implement this function in sage in the following way:

```
sage: import hashlib
```

pseudorandom
function
family

```

1796 sage: def SHA256_H(x) : 177
1797 .....:     Z5 = Integers(5) # define the group type 178
1798 .....:     hasher = hashlib.sha256(x) # Compute SHA256 179
1799 .....:     digest = hasher.hexdigest() 180
1800 .....:     z = ZZ(digest, 16) # cast into integer 181
1801 .....:     z_bin = z.digits(base=2, padto=256) # cast to 256 182
1802         bits
1803 .....:     return Z5(2)^z_bin[0] * Z5(3)^z_bin[1] 183
1804 sage: SHA256_H(b"") # evaluate on empty string 184
1805 2 185
1806 sage: SHA256_H(b"SHA") # possible images are {1,2,3} 186
1807 3 187
1808 sage: SHA256_H(b"Math") 188
1809 1 189

```

1810 *Exercise 43.* Consider the multiplicative group \mathbb{Z}_{13}^* of modular 13 arithmetic from example 33.
1811 Choose a set of 3 generators of \mathbb{Z}_{13}^* , define its associated Pedersen hash function and compute
1812 the Pedersen hash of $(3, 7, 11) \in \mathbb{Z}_{12}$.

1813 *Exercise 44.* Consider the Pedersen hash from exercise 43. Compose it with the *SHA256* hash
1814 function from example 45 to define a hash-to-group function. Implement that function in sage.

1815 4.1.7.4 Pseudorandom Function Families in DDH-secure groups

1816 As noted in 4.1.10, the family of Pederson's hash functions, parameterized by a set of generators
1817 $\{g_1, \dots, g_j\}$ does not qualify as a family of pseudorandom functions and should therefore not be
1818 instantiated as such. To see an example of a proper family of pseudorandom functions in groups
1819 where the decisional Diffie–Hellman assumption 4.1.6.2 is assumed to hold true, let \mathbb{G} be a DDH-
1820 secure cyclic group of order n with generator g and $\{a_0, a_1, \dots, a_k\} \subset \mathbb{Z}_n^*$ a uniform randomly
1821 generated set of numbers invertible in modular n arithmetics. Then a family of pseudorandom
1822 functions, parameterized by the seed $\{a_0, a_1, \dots, a_k\}$ is given as follows:

$$F_{\{a_0, a_1, \dots, a_k\}} : \{0, 1\}^{k+1} \rightarrow \mathbb{G} : (b_0, \dots, b_k) \mapsto g^{b_0 \cdot \prod_{i=1}^k a_i^{b_i}} \quad (4.26)$$

1823 *Exercise 45.* Consider the multiplicative group \mathbb{Z}_{13}^* of modular 13 arithmetic from example 33
1824 and the parameter $k = 3$. Choose a generator of \mathbb{Z}_{13}^* , a seed and instantiate a member of the
1825 familie 4.26 for that seed. Evaluate that member on the binary string $\langle 1, 0, 1 \rangle$.

1826 4.2 Commutative Rings

1827 In the previous section we have seen that integers are a commutative group with respect to
1828 integer addition, but as we know there are in fact two arithmetic operations defined on integers:
1829 addition and multiplication. However, in contrast to addition, multiplication does not define a
1830 group structure, given that integers generally don't have multiplicative inverses. Configurations
1831 like these constitute so-called **commutative rings with unit** and the following definition will
1832 make the structure explicit:

1833 *Definition 4.2.0.1* (Commutative ring with unit). A **commutative ring with unit** $(R, +, \cdot, 1)$
1834 is a set R provided with two maps $+: R \times R \rightarrow R$ and $\cdot: R \times R \rightarrow R$, called **addition** and
1835 **multiplication** and an element $1 \in R$, called the **unit** such that the following conditions hold:

- $(R, +)$ is a commutative group, where the neutral element is denoted with 0.
- **Commutativity of multiplication:** $r_1 \cdot r_2 = r_2 \cdot r_1$ for all $r_1, r_2 \in R$.
- **Multiplicative neutral unit :** $1 \cdot g = g$ for all $g \in R$.
- **Associativity:** For every $g_1, g_2, g_3 \in \mathbb{G}$ the equation $g_1 \cdot (g_2 \cdot g_3) = (g_1 \cdot g_2) \cdot g_3$ holds.
- **Distributivity:** For all $g_1, g_2, g_3 \in R$ the distributive law $g_1 \cdot (g_2 + g_3) = g_1 \cdot g_2 + g_1 \cdot g_3$ holds.

If $(R, +, \cdot, 1)$ is a commutative ring with unit and $R' \subset R$ is a subset of R such that the restriction of addition and multiplication to R' define a commutative ring with addition $+: R' \times R' \rightarrow R'$, multiplication $\cdot: R' \times R' \rightarrow R'$ and unit 1 on R' , then $(R', +, \cdot, 1)$ is called a **subring** of $(R, +, \cdot, 1)$.

Notation and Symbols 9. Since we are exclusively concerned with commutative rings in this book, we often just call them rings, keeping the notation of commutativity implicit. A set R with two maps $+$ and \cdot that satisfies all previously mentioned rules, except for the commutativity law of the multiplication is called a non-commutative ring.

If there is no risk of ambiguity (about what the addition and multiplication maps of a ring are), we frequently drop the symbols $+$ and \cdot and simply write R as notation for the ring, keeping those maps implicit. In this case we also say that R is of ring type, indicating that R is not simply a set but a set together with an addition and a multiplication map.

Example 48 (The ring of integers). The set \mathbb{Z} of integers with the usual addition and multiplication is the archetypical example of a commutative ring with unit 1.

sage: ZZ

190

Integer Ring

191

Example 49 (Underlying commutative group of a ring). Every commutative ring with unit $(R, +, \cdot, 1)$ gives rise to a group, if we disregard multiplication.

The following example is somewhat unusual, but we encourage you to think through it because it helps to detach the mind from familiar styles of computation and concentrate on the abstract algebraic explanation.

Example 50. Let $S := \{\bullet, \star, \odot, \otimes\}$ be a set that contains four elements, and let addition and multiplication on S be defined as follows:

\cup	\bullet	\star	\odot	\otimes
\bullet	\bullet	\star	\odot	\otimes
\star	\star	\odot	\otimes	\bullet
\odot	\odot	\otimes	\bullet	\star
\otimes	\otimes	\bullet	\star	\odot

\circ	\bullet	\star	\odot	\otimes
\bullet	\bullet	\bullet	\bullet	\bullet
\star	\bullet	\star	\odot	\otimes
\odot	\bullet	\odot	\bullet	\odot
\otimes	\bullet	\otimes	\odot	\star

Then (S, \cup, \circ, \star) is a ring with unit \star and zero \bullet . It therefore makes sense to ask for solutions to equations like this one: Find $x \in S$ such that

$$\otimes \circ (x \cup \odot) = \star$$

To see how such a “moonmath equation” can be solved, we have to keep in mind that rings behaves mostly like normal numbers when it comes to bracketing and computation rules. The

only differences are the symbols, and the actual way to add and multiply them. With this in mind, we solve the equation for x in the “usual way”²:

$$\begin{array}{ll}
 \otimes \circ (x \cup \odot) = \star & \# \text{ apply the distributive law} \\
 \otimes \circ x \cup \otimes \circ \odot = \star & \# \otimes \circ \odot = \odot \\
 \otimes \circ x \cup \odot = \star & \# \text{ concatenate the } \cup \text{ inverse of } \odot \text{ to both sides} \\
 \otimes \circ x \cup \odot \cup -\odot = \star \cup -\odot & \# \odot \cup -\odot = \bullet \\
 \otimes \circ x \cup \bullet = \star \cup -\odot & \# \bullet \text{ is the } \cup \text{ neutral element} \\
 \otimes \circ x = \star \cup -\odot & \# \text{ for } \cup \text{ we have } -\odot = \odot \\
 \otimes \circ x = \star \cup \odot & \# \star \cup \odot = \otimes \\
 \otimes \circ x = \otimes & \# \text{ concatenate the } \circ \text{ inverse of } \otimes \text{ to both sides} \\
 (\otimes)^{-1} \circ \otimes \circ x = (\otimes)^{-1} \circ \otimes & \# \text{ multiply with the multiplicative inverse} \\
 \star \circ x = \star & \\
 x = \star &
 \end{array}$$

So, even though this equation looked really alien at first glance, we could solve it basically exactly the way we solve “normal” equations containing numbers.

Note, however, that whenever a multiplicative inverse would be needed to solve an equation in the usual way in a ring, things can be very different than most of us are used to. For example, the following equation cannot be solved for x in the usual way, since there is no multiplicative inverse for \odot in our ring.

$$\odot \circ x = \otimes \quad (4.27)$$

We can confirm this by looking at the multiplication table to see that no such x exists.

As another example, the following equation does not have a single solution but two: $x \in \{\star, \otimes\}$.

$$\odot \circ x = \odot \quad (4.28)$$

Having no solution or two solutions is certainly not something to expect from types like the rational numbers \mathbb{Q} .

Example 51 (Ring of Polynomials). Considering the definition of polynomials from 3.4 again, we notice that what we have informally called the type R of the coefficients must in fact be a commutative ring with a unit, since we need addition, multiplication, commutativity and the existence of a unit for $R[x]$ to have the properties we expect.

In fact if we consider R to be a ring and we define addition and multiplication of polynomials as in 3.28, the set $R[x]$ is a commutative ring with a unit, where the polynomial 1 is the multiplicative unit. We call this ring the **ring of polynomials with coefficients in R** .

sage: `ZZ['x']`

Univariate Polynomial Ring in x over Integer Ring

Example 52 (Ring of modular n arithmetic). Let n be a modulus and $(\mathbb{Z}_n, +, \cdot)$ the set of all remainder classes of integers modulo n , with the projection of integer addition and multiplication as defined in 3.3.4. Then $(\mathbb{Z}_n, +, \cdot)$ is a commutative ring with unit 1.

²Note that there are more efficient ways to solve this equation. The point of our computation is to show how the axioms of a ring can be used to solve the equation

1888 **sage: Integers(6)**

194

1889 **Ring of integers modulo 6**

195

1890 *Example 53* (Binary Representations in Modular Arithmetic). TODO (Non unique)

1891 *Example 54* (Polynomial evaluation in the exponent of group generators). As we show in 6.2.3,
1892 a key insights in many zero knowledge protocols is the ability to encode computations as poly-
1893 nomials and then to hide the information of that computation by evaluating the polynomial “in
1894 the exponent” of certain cryptographic groups 8.2.

To understand the underlying principle of this idea, consider the exponential map 46 again. If \mathbb{G} is a finite cyclic group of order n with generator $g \in \mathbb{G}$, then the ring structure of $(\mathbb{Z}_n, +, \cdot)$ corresponds to the group structure of \mathbb{G} in the following way:

$$g^{x+y} = g^x \cdot g^y \quad g^{x \cdot y} = (g^x)^y \quad \text{for all } x, y \in \mathbb{Z}_n \quad (4.29)$$

This correspondence allows polynomials with coefficients in \mathbb{Z}_n to be evaluated “in the exponent” of a group generator. To see what this means, let $p \in \mathbb{Z}_n[x]$ be a polynomial with $p(x) = a_m \cdot x^m + a_{m-1}x^{m-1} + \dots + a_1x + a_0$ and let $s \in \mathbb{Z}_n$ be an evaluation point. Then the previously defined exponential laws 4.29 imply the following identity:

$$\begin{aligned} g^{p(s)} &= g^{a_m \cdot s^m + a_{m-1}s^{m-1} + \dots + a_1s + a_0} \\ &= \left(g^{s^m}\right)^{a_m} \cdot \left(g^{s^{m-1}}\right)^{a_{m-1}} \cdot \dots \cdot (g^s)^{a_1} \cdot g^{a_0} \end{aligned} \quad (4.30)$$

1895 Utilizing these identities, it is possible to evaluate any polynomial p of degree $\deg(p) \leq m$ at a
1896 “secret” evaluation point s in the exponent of g without any knowledge about s , assuming that
1897 \mathbb{G} is a DL-group. To see this assume that the set $\{g, g^s, g^{s^2}, \dots, g^{s^m}\}$ is given, but s is unknown.
1898 Then $g^{p(s)}$ can be computed using 4.30, however it is not feasible to compute s .

Example 55. To give an example of the evaluation of a polynomial in the exponent of a finite cyclic group, consider the exponential map from example 36:

$$3^{(\cdot)} : \mathbb{Z}_4 \rightarrow \mathbb{Z}_5^* ; x \mapsto 3^x$$

Choosing the polynomial $p(x) = 2x^2 + 3x + 1$ from $\mathbb{Z}_4[x]$, we first evaluate the polynomial at the point $s = 2$ and then write the result into the exponent 3 as follows:

$$\begin{aligned} 3^{p(2)} &= 3^{2 \cdot 2^2 + 3 \cdot 2 + 1} \\ &= 3^{2 \cdot 0 + 2 + 1} \\ &= 3^3 \\ &= 2 \end{aligned}$$

This was possible, because we had access to the evaluation point 2. On the other hand, if we only had access to the set $\{3, 4, 1\}$ and we knew that this set represents the set $\{3, 3^s, 3^{s^2}\}$ for some secret value s , we could evaluate p at the point s in the exponent of 3 as

$$\begin{aligned} 3^{p(s)} &= 1^2 \cdot 4^3 \cdot 3^1 \\ &= 1 \cdot 4 \cdot 3 \\ &= 2 \end{aligned}$$

1899 Both computations agree, since the secret point s was equal to 2 in this example. However the
1900 second evaluation was possible without any knowledge about s .

4.2.1 Hashing into Modular Arithmetic

As we have seen in 4.1.7, various constructions for hashing-to-groups are known and used in applications. As commutative rings are commutative groups, when we disregard the multiplicative structure, hash-to-group constructions can be applied for hashing into commutative rings.

One of the most widely used applications of hash-into-ring constructions are hash functions that map into the ring \mathbb{Z}_n of modular n arithmetics for some modulus n . Different approaches to construct such a function are known, but probably the most widely used ones are based on the insight that the images of general hash functions can be interpreted as binary representations of integers, as explained in example 46.

It follows from this interpretation that one simple method of hashing into \mathbb{Z}_n is constructed by observing that if n is a modulus with a bit-length 3.4 of $k = |n|$, then every binary string $\langle b_0, b_1, \dots, b_{k-2} \rangle$ of length $k - 1$ defines an integer z in the range $0 \leq z \leq 2^{k-1} - 1 < n$:

$$z = b_0 \cdot 2^0 + b_1 \cdot 2^1 + \dots + b_{k-2} \cdot 2^{k-2} \quad (4.31)$$

Now, since $z < n$, we know that z is guaranteed to be in the set $\{0, 1, \dots, n - 1\}$, and hence it can be interpreted as an element of \mathbb{Z}_n . From this it follows that if $H : \{0, 1\}^* \rightarrow \{0, 1\}^{k-1}$ is a hash function, then a hash-to-ring function can be constructed as follows:

$$H_{|n|_2-1} : \{0, 1\}^* \rightarrow \mathbb{Z}_n : s \mapsto H(s)_0 \cdot 2^0 + H(s)_1 \cdot 2^1 + \dots + H(s)_{k-2} \cdot 2^{k-2} \quad (4.32)$$

A drawback of this hash function is that the distribution of the hash values in \mathbb{Z}_n is not necessarily uniform. In fact, if n is larger than 2^{k-1} , then by design $H_{|n|_2-1}$ will never hash onto values $z \geq 2^{k-1}$. Using this hashing method therefore generates approximately uniform hashes only, if n is very close to 2^{k-1} . In the worst case, when $n = 2^k - 1$, it misses almost half of all elements from \mathbb{Z}_n .

An advantage of this approach is that properties like preimage resistance or collision resistance of the original hash function $H(\cdot)$ are preserved.

Example 56. To analyze a particular implementation of a $H_{|n|_2-1}$ hash function, we use a 5-bit truncation of the *SHA256* hash from example 45 and define a hash into \mathbb{Z}_{16} as follows:

$$H_{|16|_2-5} : \{0, 1\}^* \rightarrow \mathbb{Z}_{16} : s \mapsto \text{SHA256}(s)_0 \cdot 2^0 + \text{SHA256}(s)_1 \cdot 2^1 + \dots + \text{SHA256}(s)_4 \cdot 2^4$$

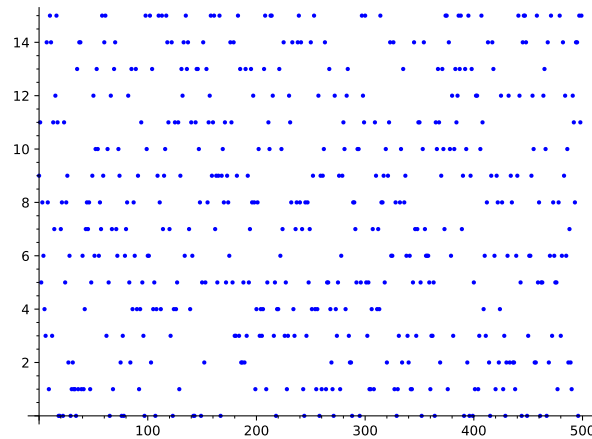
Since $k = |16|_2 = 5$ and $16 - 2^{k-1} = 0$, this hash maps uniformly onto \mathbb{Z}_{16} . We can invoke Sage to implement it:

```

sage: import hashlib
sage: def Hash5(x):
.....:     Z16 = Integers(16)
.....:     hasher = hashlib.sha256(x) # compute SHA56
.....:     digest = hasher.hexdigest()
.....:     d = ZZ(digest, base=16) # cast to integer
.....:     d = d.str(2)[-4:] # keep 5 least significant bits
.....:     d = ZZ(d, base=2) # cast to integer
.....:     return Z16(d) # cast to Z16
sage: Hash5(b' ')
5

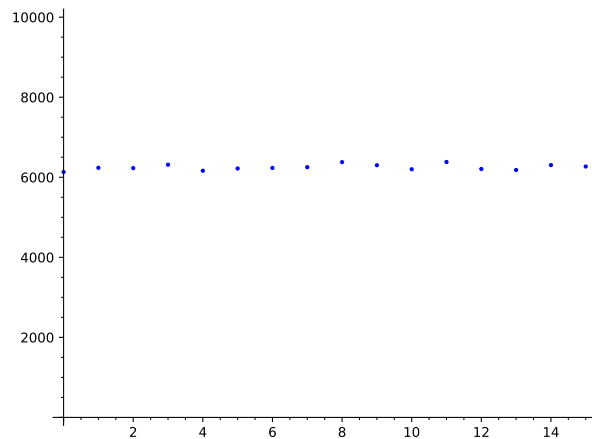
```

We can then use Sage to apply this function to a large set of input values in order to plot a visualization of the distribution over the set $\{0, \dots, 15\}$. Executing over 500 input values gives the following plot:



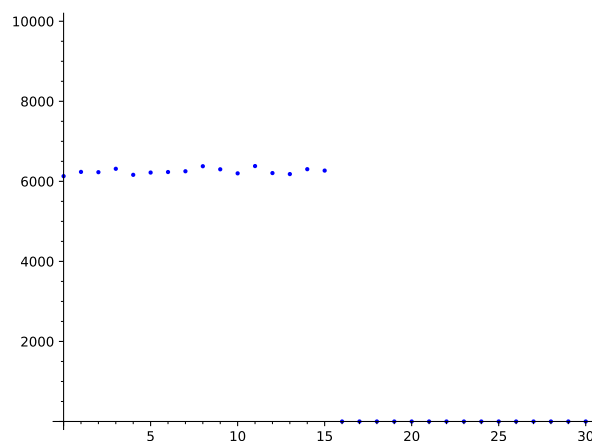
1939

1940 To get an intuition of uniformity, we can count the number of times the hash function $H_{|16|_2-1}$
 1941 maps onto each number in the set $\{0, 1, \dots, 15\}$ in a loop of 100000 hashes, and compare that
 1942 to the ideal uniform distribution, which would map exactly 6250 samples to each element. This
 1943 gives the following result:



1944

1945 The lack of uniformity becomes apparent if we want to construct a similar hash function for \mathbb{Z}_n
 1946 for any other 5 bit integer n in the range $17 \leq n \leq 31$. In this case, the definition of the hash
 1947 function is exactly the same as for \mathbb{Z}_{16} , and hence, the images will not exceed the value 15.
 1948 So, for example, even in the case of hashing to \mathbb{Z}_{31} , the hash function never maps to any value
 1949 larger than 15, leaving almost half of all numbers out of the image range.



1950

A second widely used method of hashing into \mathbb{Z}_n is constructed by observing the following: If n is a modulus with a bit-length of $|n|_2 = k_1$ and $H : \{0, 1\}^* \rightarrow \{0, 1\}^{k_2}$ is a hash function that produces digests of size k_2 , with $k_2 \geq k_1$, then a hash-to-ring function can be constructed by interpreting the image of H as a binary representation of an integer and then taking the modulus by n to map into \mathbb{Z}_n .

$$H'_{mod_n} : \{0, 1\}^* \rightarrow \mathbb{Z}_n : s \mapsto \left(H(s)_0 \cdot 2^0 + H(s)_1 \cdot 2^1 + \dots + H(s)_{k_2} \cdot 2^{k_2} \right) \bmod n \quad (4.33)$$

A drawback of this hash function is that computing the modulus requires some computational effort. In addition, the distribution of the hash values in \mathbb{Z}_n might not be uniform, depending on the number $2^{k_2+1} \bmod n$. An advantage of it is that potential properties of the original hash function $H(\cdot)$ (like preimage resistance or collision resistance) are preserved, and the distribution can be made almost uniform, with only negligible bias depending on what modulus n and images size k_2 are chosen.

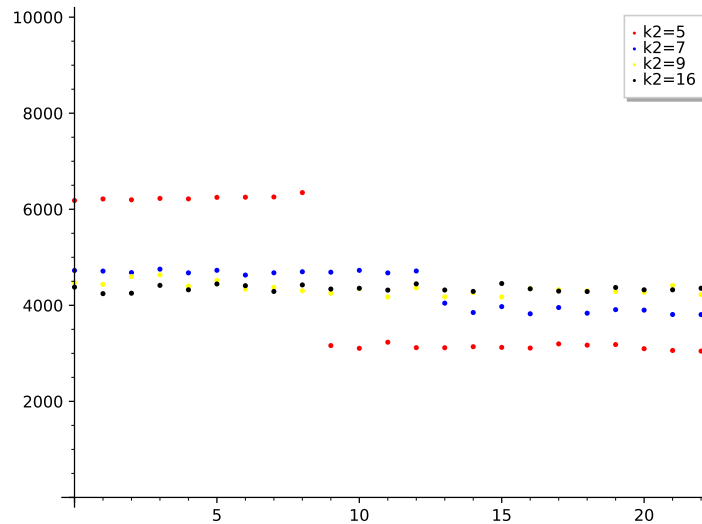
Example 57. To give an implementation of the H_{mod_n} hash function, we use k_2 -bit truncation of the *SHA256* hash from example 45, and define a hash into \mathbb{Z}_{23} as follows:

$$H_{mod_{23}, k_2} : \{0, 1\}^* \rightarrow \mathbb{Z}_{23} : s \mapsto \left(SHA256(s)_0 \cdot 2^0 + SHA256(s)_1 \cdot 2^1 + \dots + SHA256(s)_{k_2} \cdot 2^{k_2} \right) \bmod 23$$

We want to use various instantiations of k_2 to visualize the impact of truncation length on the distribution of the hashes in \mathbb{Z}_{23} . We can invoke Sage to implement it as follows:

```
sage: import hashlib                                207
sage: Z23 = Integers(23)                             208
sage: def Hash_mod23(x, k2):                          209
.....:     hasher = hashlib.sha256(x.encode('utf-8')) # Compute 210
SHA256
.....:     digest = hasher.hexdigest()                211
.....:     d = ZZ(digest, base=16) # cast to integer    212
.....:     d = d.str(2)[-k2:] # keep k2+1 LSB          213
.....:     d = ZZ(d, base=2) # cast to integer         214
.....:     return Z23(d) # cast to Z23                215
```

We can then use Sage to apply this function to a large set of input values in order to plot visualizations of the distribution over the set $\{0, \dots, 22\}$ for various values of k_2 , by counting the number of times it maps onto each number in a loop of 100000 hashes. We get the following plot:



A third method that can sometimes be found in implementations is the so-called **“try-and-increment” method**. To understand this method, we define an integer $z \in \mathbb{Z}$ from any hash value $H(s)$ as we did in the previous methods, that is, we define

$$z = H(s)_0 \cdot 2^0 + H(s)_1 \cdot 2^1 + \dots + H(s)_{k-1} \cdot 2^k$$

Hashing into \mathbb{Z}_n is then achievable by first computing z , and then trying to see if $z \in \mathbb{Z}_n$. If it is, then the hash is done; if not, the string s is modified in a deterministic way and the process is repeated until a suitable element $z \in \mathbb{Z}_n$ is found. A suitable, deterministic modification could be to concatenate the original string by some bit counter. A “try-and-increment” algorithm would then work like in algorithm 6.

Algorithm 6 Hash-to- \mathbb{Z}_n

Require: $n \in \mathbb{Z}$ with $|n|_2 = k$ and $s \in \{0, 1\}^*$

procedure TRY-AND-INCREMENT(n, k, s)

$c \leftarrow 0$

repeat

$s' \leftarrow s || c_bits()$

$z \leftarrow H(s')_0 \cdot 2^0 + H(s')_1 \cdot 2^1 + \dots + H(s')_k \cdot 2^k$

$c \leftarrow c + 1$

until $z < n$

return x

end procedure

Ensure: $z \in \mathbb{Z}_n$

Depending on the parameters, this method can be very efficient. In fact, if k is sufficiently large and n is close to 2^{k+1} , the probability for $z < n$ is very high and the repeat loop will almost always be executed a single time only. A drawback is, however, that the probability of having to execute the loop multiple times is not zero.

4.3 Fields

We started this chapter with the definition of a group 4.1, which we then expanded into the definition of a commutative ring with a unit 4.2. Such rings generalize the behavior of integers.

In this section, we will look at those special cases of commutative rings where every element other than the neutral element of addition has a multiplicative inverse. Those structures behave very much like the rational numbers \mathbb{Q} . Rational numbers are, in a sense, an extension of the ring of integers, that is, they are constructed by including newly defined multiplicative inverses (fractions) to the integers. The following definition makes the definition of a field precise:

Definition 4.3.0.1 (Field). A **field** $(\mathbb{F}, +, \cdot)$ is a set \mathbb{F} provided with two maps $+: \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{F}$ and $\cdot: \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{F}$, called **addition** and **multiplication** such that the following conditions hold:

- $(\mathbb{F}, +)$ is a commutative group, where the neutral element is denoted by 0.
- $(\mathbb{F} \setminus \{0\}, \cdot)$ is a commutative group, where the neutral element is denoted by 1.
- (Distributivity) The equation $g_1 \cdot (g_2 + g_3) = g_1 \cdot g_2 + g_1 \cdot g_3$ holds for all $g_1, g_2, g_3 \in \mathbb{F}$.

If $(\mathbb{F}, +, \cdot)$ is a field and $\mathbb{F}' \subset \mathbb{F}$ is a subset of \mathbb{F} such that the restriction of addition and multiplication to \mathbb{F}' define a field with addition $+: \mathbb{F}' \times \mathbb{F}' \rightarrow \mathbb{F}'$ and multiplication $\cdot: \mathbb{F}' \times \mathbb{F}' \rightarrow \mathbb{F}'$ on \mathbb{F}' , then $(\mathbb{F}', +, \cdot)$ is called a **subfield** of $(\mathbb{F}, +, \cdot)$ and $(\mathbb{F}, +, \cdot)$ is called an **extension field** of $(\mathbb{F}', +, \cdot)$.

Notation and Symbols 10. If there is no risk of ambiguity (about what the addition and multiplication maps of a field are), we frequently drop the symbols $+$ and \cdot and simply write \mathbb{F} as notation for the field, keeping those maps implicit. In this case we also say that \mathbb{F} is of field type, indicating that \mathbb{F} is not simply a set but a set together with an addition and a multiplication map that satisfy the field axioms 4.3.0.1.

We call $(\mathbb{F}, +)$ the **additive group** of the field, write $\mathbb{F}^* := \mathbb{F} \setminus \{0\}$ for the set of all elements excluding the neutral element of addition and call the group (\mathbb{F}^*, \cdot) the **multiplicative group** of the field.

The **characteristic** of a field \mathbb{F} , represented as $\text{char}(\mathbb{F})$, is the smallest natural number $n \geq 1$ for which the n -fold sum of the multiplicative neutral element 1 equals zero, i.e. for which $\sum_{i=1}^n 1 = 0$. If such an $n > 0$ exists, the field is also said to have a **finite characteristic**. If, on the other hand, every finite sum of 1 is such that it is not equal to zero, then the field is defined to have characteristic 0. **S: Tried to disambiguate the scope of negation between 1. “It is true of every finite sum of 1 that it is not equal to 0” and 2. “It is not true of every finite sum of 1 that it is equal to 0” From the example below, it looks like 1. is the intended meaning here, correct?**

Check
change of
wording

Example 58 (Field of rational numbers). Probably the best known example of a field is the set of rational numbers \mathbb{Q} together with the usual definition of addition, subtraction, multiplication and division. Since there is no natural number $n \in \mathbb{N}$ such that $\sum_{j=0}^n 1 = 0$ in the set of rational numbers, the characteristic of the field \mathbb{Q} is given by $\text{char}(\mathbb{Q}) = 0$.

sage: QQ

Rational Field

216

217

Example 59 (Field with two elements). It can be shown that, in any field, the neutral element of addition 0 must be different from the neutral element of multiplication 1, that is, $0 \neq 1$ always holds in a field. From this, it follows that the smallest field must contain at least two elements. As the following addition and multiplication tables show, there is indeed a field with two elements, which is usually called \mathbb{F}_2 :

Let $\mathbb{F}_2 := \{0, 1\}$ be a set that contains two elements and let addition and multiplication on \mathbb{F}_2 be defined as follows:

$+$	0	1	\cdot	0	1
0	0	1	0	0	0
1	1	0	1	0	1

Since $1 + 1 = 0$ in the field \mathbb{F}_2 , we know that the characteristic of \mathbb{F}_2 given by $\text{char}(\mathbb{F}_2) = 2$ and the multiplicative subgroup \mathbb{F}_2^* of \mathbb{F}_2 is given by the trivial group $\{1\}$.

```

sage: F2 = GF(2)
sage: F2(1) # Get an element from GF(2)
1
sage: F2(1) + F2(1) # Addition
0
sage: F2(1) / F2(1) # Division
1

```

Exercise 46. Consider the ring of modular 5 arithmetics $(\mathbb{Z}_5, +, \cdot)$ from example 14. Show that $(\mathbb{Z}_5, +, \cdot)$ is a field. What is the characteristic of \mathbb{Z}_5 ? Proof that the equation $a \cdot x = b$ has only a single solution $x \in \mathbb{Z}_5$ for any given $a, b \in \mathbb{Z}_5^*$.

Exercise 47. Consider the ring of modular 6 arithmetics $(\mathbb{Z}_6, +, \cdot)$ from example 9. Show that $(\mathbb{Z}_6, +, \cdot)$ is not a field.

4.3.1 Prime fields

As we have seen in the various examples of the previous sections, modular arithmetics behaves similarly to the ordinary arithmetics of integers in many ways. This is due to the fact that remainder class sets \mathbb{Z}_n are commutative rings with units 52.

However, we have also seen in 36 that, whenever the modulus is a prime number, every remainder class other than the zero class has a modular multiplicative inverse. This is an important observation, since it immediately implies that, in case the modulus is a prime number, the remainder class set \mathbb{Z}_n is not just a ring but actually a **field**. Moreover, since $\sum_{j=0}^n 1 = 0$ in \mathbb{Z}_n , we know that those fields have the finite characteristic n .

Notation and Symbols 11 (Prime Fields). Let $p \in \mathbb{P}$ be a prime number and $(\mathbb{Z}_p, +, \cdot)$ the ring of modular p arithmetics 52. To distinguish prime fields from arbitrary modular arithmetic rings, we write $(\mathbb{F}_p, +, \cdot)$ for the ring of modular p arithmetics and call it the **prime field** of characteristic p .

Prime fields are the foundation for many of the contemporary algebra-based cryptographic systems, as they have some desirable properties. One of them is that any prime field of characteristic p contain exactly p elements which can be represented on a computer with not more than $\log_2(p)$ many bits. On the other hand fields like the rational numbers, require a potentially unbounded amount of bits for any full-precision representation.

Since prime fields are special cases of modular arithmetic rings, addition and multiplication can be computed by first doing normal integer addition and multiplication, and then considering the remainder in Eucliden division by p as the result. For any prime field element $x \in \mathbb{F}_p$, its additive inverse (the negative) is given by $-x = p - x \bmod p$. For $x \neq 0$ the multiplicative inverse always exists and is given by $x^{-1} = x^{p-2}$. Division is then defined by multiplication with the multiplicative inverse as explained in 3.3.5. Alternative the multiplicative inverse can be computed using the Extended Euclidean Algorithm as explained in 3.23.

Example 60. The smallest field is the field \mathbb{F}_2 of characteristic 2 as we have seen in example 59. It is the prime field of the prime number 2.

2075 *Example 61.* The field \mathbb{F}_5 from example 14 as defined by its addition and multiplication table is
 2076 a prime field.

Example 62. To summarize the basic aspects of computation in prime fields, let us consider the prime field \mathbb{F}_5 and simplify the following expression:

$$\left(\frac{2}{3} - 2\right) \cdot 2$$

A first thing to note is that since \mathbb{F}_5 is a field, all rules are identical to the rules we learned in school when we were dealing with rational, real or complex numbers. This means we can use e.g. bracketing (distributivity) or addition as usual:

$$\begin{aligned} \left(\frac{2}{3} - 2\right) \cdot 2 &= \frac{2}{3} \cdot 2 - 2 \cdot 2 && \# \text{ distributive law} \\ &= \frac{2 \cdot 2}{3} - 2 \cdot 2 && 4 \bmod 5 = 4 \\ &= \frac{4}{3} - 4 && \# \text{ multiplicative inverse of 3 is } 3^{5-2} \bmod 5 = 2 \\ &= 4 \cdot 2 - 4 && \# \text{ additive inverse of 4 is } 5 - 4 = 1 \\ &= 4 \cdot 2 + 1 && 8 \bmod 5 = 3 \\ &= 3 + 1 && 4 \bmod 5 = 4 \\ &= 4 \end{aligned}$$

2077 In this computation, we computed the multiplicative inverse of 3 using the identity $x^{-1} = x^{p-2}$
 2078 in a prime field. This is impractical for large prime numbers. Recall that another way of
 2079 computing the multiplicative inverse is the Extended Euclidean Algorithm (see 3.13). To refresh
 2080 our memory, the algorithm solves the equation $x^{-1} \cdot 3 + t \cdot 5 = 1$, for x^{-1} , even though in this
 2081 case t is irrelevant. We get:

k	r_k	x_k^{-1}	t_k
0	3	1	.
1	5	0	.
2	3	1	.
3	2	-1	.
4	1	2	.

2083 So the multiplicative inverse of 3 in \mathbb{Z}_5 is 2, and, indeed, if we compute the product of 3 with
 2084 its multiplicative inverse 2 we get the neutral element 1 in \mathbb{F}_5 .

2085 *Exercise 48* (Prime field \mathbb{F}_3). Construct the addition and multiplication table of the prime field
 2086 \mathbb{F}_3 .

2087 *Exercise 49* (Prime field \mathbb{F}_{13}). Construct the addition and multiplication table of the prime field
 2088 \mathbb{F}_{13} .

Exercise 50. Consider the prime field \mathbb{F}_{13} from exercise 49. Find the set of all pairs $(x, y) \in \mathbb{F}_{13} \times \mathbb{F}_{13}$ that satisfy the equation

$$x^2 + y^2 = 1 + 7 \cdot x^2 \cdot y^2$$

4.3.2 Square Roots

As we know from integer arithmetics some integers like 4 or 9 are squares of other integers, as for example $4 = 2^2$ and $9 = 3^2$. However we also know that not all integers are squares of other integers, as for example there is no integers $x \in \mathbb{Z}$ such that $x^2 = 2$. If an integer a is square of another integer b , then it make sense to define the square-root of a to be b .

In the context of prime fields an element that is a square of another element is also called a **quadratic residue** and an element that is not a square of another element is called a **quadratic non-residue**. Those notations are of particular importance in our studies on elliptic curves in chapter 5, as only square numbers can actually be points on an elliptic curve.

To make the intuition of quadratic residues and their roots precise, let $p \in \mathbb{P}$ be a prime number and \mathbb{F}_p its associated prime field. Then a number $x \in \mathbb{F}_p$ is called a **square root** of another number $y \in \mathbb{F}_p$, if x is a solution to the following equation:

$$x^2 = y \quad (4.34)$$

In this case, y is called a **quadratic residue**. On the other hand, if y is given and the quadratic equation has no solution x , we call y a **quadratic non-residue**. For any $y \in \mathbb{F}_p$, we denote the set of all square roots of y in the prime field \mathbb{F}_p as follows:

$$\sqrt{y} := \{x \in \mathbb{F}_p \mid x^2 = y\} \quad (4.35)$$

Informally speaking, quadratic residues are numbers such that we can take the square root of them, while quadratic non-residues are numbers that don't have square roots. The situation therefore parallels the familiar case of integers, where some integers like 4 or 9 have square roots and others like 2 or 3 don't (as integers).

If y is a quadratic non-residue, then $\sqrt{y} = \emptyset$ (an empty set), and if $y = 0$, then $\sqrt{y} = \{0\}$. Moreover if $y \neq 0$ is a quadratic residue then it has precisely two roots $\sqrt{y} = \{x, p - x\}$ for some $x \in \mathbb{F}_p$. We adopt the convention to call the smaller one (when interpreted as an integer) as the **positive** square root and the larger one as the **negative** square root. If $p \in \mathbb{P}_{\geq 3}$ is an odd prime number with associated prime field \mathbb{F}_p , then there are precisely $(p+1)/2$ many quadratic residues and $(p-1)/2$ quadratic non residues.

Example 63 (Quadratic residues and roots in \mathbb{F}_5). Let us consider the prime field \mathbb{F}_5 again. All square numbers can be found on the main diagonal of the multiplication table in example 14. As you can see, in \mathbb{F}_5 only the numbers 0, 1 and 4 have square roots and we get $\sqrt{0} = \{0\}$, $\sqrt{1} = \{1, 4\}$, $\sqrt{2} = \emptyset$, $\sqrt{3} = \emptyset$ and $\sqrt{4} = \{2, 3\}$. The numbers 0, 1 and 4 are therefore quadratic residues, while the numbers 2 and 3 are quadratic non-residues.

In order to describe whether an element of a prime field is a square number or not, the so-called **Legendre symbol** can sometimes be found in the literature, which is why we will summarize it here:

Let $p \in \mathbb{P}$ be a prime number and $y \in \mathbb{F}_p$ an element from the associated prime field. Then the *Legendre symbol* of y is defined as follows:

$$\left(\frac{y}{p}\right) := \begin{cases} 1 & \text{if } y \text{ has square roots} \\ -1 & \text{if } y \text{ has no square roots} \\ 0 & \text{if } y = 0 \end{cases} \quad (4.36)$$

Example 64. Looking at the quadratic residues and non residues in \mathbb{F}_5 from example 14 again, we can deduce the following Legendre symbols, from example 63.

$$\left(\frac{0}{5}\right) = 0, \quad \left(\frac{1}{5}\right) = 1, \quad \left(\frac{2}{5}\right) = -1, \quad \left(\frac{3}{5}\right) = -1, \quad \left(\frac{4}{5}\right) = 1.$$

2126 The Legendre symbol provides a criterion to decide whether or not an element from a prime
 2127 field has a quadratic root or not. This, however, is not just of theoretical use: The so-called
 2128 **Euler criterion** provides a compact way to actually compute the Legendre symbol. To see that,
 2129 let $p \in \mathbb{P}_{\geq 3}$ be an odd prime number and $y \in \mathbb{F}_p$. Then the Legendre symbol can be computed
 2130 as follows:

$$\left(\frac{y}{p}\right) = y^{\frac{p-1}{2}}. \quad (4.37)$$

Example 65. Looking at the quadratic residues and non residues in \mathbb{F}_5 from example 63 again, we can compute the following Legendre symbols using the Euler criterion:

$$\begin{aligned} \left(\frac{0}{5}\right) &= 0^{\frac{5-1}{2}} = 0^2 = 0 \\ \left(\frac{1}{5}\right) &= 1^{\frac{5-1}{2}} = 1^2 = 1 \\ \left(\frac{2}{5}\right) &= 2^{\frac{5-1}{2}} = 2^2 = 4 = -1 \\ \left(\frac{3}{5}\right) &= 3^{\frac{5-1}{2}} = 3^2 = 4 = -1 \\ \left(\frac{4}{5}\right) &= 4^{\frac{5-1}{2}} = 4^2 = 1 \end{aligned}$$

2131 *Exercise 51.* Consider the prime field \mathbb{F}_{13} from exercise 49. Compute the Legendre symbol
 2132 $\left(\frac{x}{13}\right)$ and the set of roots \sqrt{x} for all elements $x \in \mathbb{F}_{13}$.

2133 4.3.2.1 Hashing into prime fields

2134 An important problem in cryptography is the ability to hash to (various subsets) of elliptic
 2135 curves. As we will see in chapter 5, those curves are often defined over prime fields, and hashing
 2136 to a curve might start with hashing to the prime field. It is therefore important to understand
 2137 how to hash into prime fields.

2138 In 4.2.1, we looked at a few methods of hashing into the modular arithmetic rings \mathbb{Z}_n for
 2139 arbitrary $n > 1$. As prime fields are just special instances of those rings, all methods for hashing
 2140 into \mathbb{Z}_n functions can be used for hashing into prime fields, too.

2141 4.3.3 Prime Field Extensions

2142 Prime fields, as defined in the previous section, are basic building blocks in cryptography. How-
 2143 ever, as we will see in chapter 8 so-called pairing-based SNARK systems are crucially depen-
 2144 dent on certain group pairings 4.9 defined on elliptic curves over **prime field extensions**. In
 2145 this section we therefore introduce those extensions.

2146 Given some prime number $p \in \mathbb{P}$, a natural number $m \in \mathbb{N}$ and an irreducible polynomial
 2147 $P \in \mathbb{F}_p[x]$ of degree m with coefficients from the prime field \mathbb{F}_p , then a prime field extension
 2148 $(\mathbb{F}_{p^m}, +, \cdot)$ is defined as follows:

2149 The set \mathbb{F}_{p^m} of the prime field extension is given by the set of all polynomials with a degree
 2150 less than m :

$$\mathbb{F}_{p^m} := \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_1x + a_0 \mid a_i \in \mathbb{F}_p\} \quad (4.38)$$

2151 The addition law of the prime field extension \mathbb{F}_{p^m} is given by the usual addition of polynomials
2152 as defined in 3.28:

$$+ : \mathbb{F}_{p^m} \times \mathbb{F}_{p^m} \rightarrow \mathbb{F}_{p^m}, (\sum_{j=0}^m a_j x^j, \sum_{j=0}^m b_j x^j) \mapsto \sum_{j=0}^m (a_j + b_j) x^j \quad (4.39)$$

2153 The multiplication law of the prime field extension \mathbb{F}_{p^m} is given by first multiplying the two
2154 polynomials as defined in 3.29 and then divided the result by the irreducible polynomial p and
2155 keep the remainder:

$$\cdot : \mathbb{F}_{p^m} \times \mathbb{F}_{p^m} \rightarrow \mathbb{F}_{p^m}, (\sum_{j=0}^m a_j x^j, \sum_{j=0}^m b_j x^j) \mapsto (\sum_{n=0}^{2m} \sum_{i=0}^n a_i b_{n-i} x^n) \bmod P \quad (4.40)$$

2156 The neutral element of the additive group $(\mathbb{F}_{p^m}, +)$ is given by the zero polynomial 0 and the
2157 additive inverse is given by the polynomial with all negative coefficients. The neutral element of
2158 the multiplicative group $(\mathbb{F}_{p^m}^*, \cdot)$ is given by the unit polynomial 1 and the multiplicative inverse
2159 can be computed by the extended Euclidean algorithm.

2160 We can see from the definition of \mathbb{F}_{p^m} that the field is of characteristic p , since the mul-
2161 tiplicative neutral element 1 is equivalent to the multiplicative element 1 from the underlying
2162 prime field, and hence $\sum_{j=0}^p 1 = 0$. Moreover, \mathbb{F}_{p^m} is finite and contains p^m many elements,
2163 since elements are polynomials of degree $< m$, and every coefficient a_j can have p many differ-
2164 ent values. In addition, we see that the prime field \mathbb{F}_p is a subfield of \mathbb{F}_{p^m} that occurs when we
2165 restrict the elements of \mathbb{F}_{p^m} to polynomials of degree zero.

2166 One key point is that the construction of \mathbb{F}_{p^m} depends on the choice of an irreducible poly-
2167 nomial, and, in fact, different choices will give different multiplication tables, since the remainders
2168 from dividing a polynomial product by those polynomials will be different.

2169 It can, however, be shown that the fields for different choices of P are **isomorphic**, which
2170 means that there is a one-to-one correspondence between all of them. Consequently, from
2171 an abstract point of view, they are the same thing. From an implementations point of view,
2172 however, some choices are preferable to others because they allow for faster computations.

2173 *Remark 3.* Similar to the way prime fields \mathbb{F}_p are generated by starting with the ring of integers
2174 and then dividing by a prime number p and keeping the remainder, prime field extensions \mathbb{F}_{p^m}
2175 are generated by starting with the ring $\mathbb{F}_p[x]$ of polynomials and then dividing them by an
2176 irreducible polynomial of degree m and keeping the remainder.

2177 In fact it can be shown that \mathbb{F}_{p^m} is the set of all remainders when dividing any polynomial
2178 $Q \in \mathbb{F}_p[x]$ by an irreducible polynomial P of degree m . This is analogous to how \mathbb{F}_p is the set of
2179 all remainders when dividing integers by p .

2180 Any field \mathbb{F}_{p^m} constructed in the above manner is a field extension of \mathbb{F}_p . To be more
2181 general, a field $\mathbb{F}_{p^{m_2}}$ is a field extension of a field $\mathbb{F}_{p^{m_1}}$, if and only if m_1 divides m_2 . From this,
2182 we can deduce that, for any given fixed prime number, there are nested sequences of subfields
2183 whenever the power m_j divides the power m_{j+1} :

$$\mathbb{F}_p \subset \mathbb{F}_{p^{m_1}} \subset \cdots \subset \mathbb{F}_{p^{m_k}} \quad (4.41)$$

2184 To get a more intuitive picture of this, we construct an extension field of the prime field \mathbb{F}_3
2185 in the following example, and we can see how \mathbb{F}_3 sits inside that extension field.

Example 66 (The Extension field \mathbb{F}_{32}). In exercise 48 we have constructed the prime field \mathbb{F}_3 .
In this example, we apply the definition of a field extension 4.38 to construct the extension field
 \mathbb{F}_{32} . We start by choosing an irreducible polynomial of degree 2 with coefficients in \mathbb{F}_3 . We try

$P(t) = t^2 + 1$. Possibly the fastest way to show that P is indeed irreducible is to just insert all elements from \mathbb{F}_3 to see if the result is ever zero. We compute as follows:

$$P(0) = 0^2 + 1 = 1$$

$$P(1) = 1^2 + 1 = 2$$

$$P(2) = 2^2 + 1 = 1 + 1 = 2$$

This implies that P is irreducible, since all factors must be of the form $(t - a)$ for a being a root of P . The set \mathbb{F}_{3^2} contains all polynomials of degrees lower than two, with coefficients in \mathbb{F}_3 , which are precisely as listed below:

$$\mathbb{F}_{3^2} = \{0, 1, 2, t, t+1, t+2, 2t, 2t+1, 2t+2\}$$

2186 As expected, our extension field contains 9 elements. Addition is defined as addition of poly-
2187 nomials; for example $(t+2) + (2t+2) = (1+2)t + (2+2) = 1$. Doing this computation for all
2188 elements gives the following addition table

+	0	1	2	t	t+1	t+2	2t	2t+1	2t+2
0	0	1	2	t	t+1	t+2	2t	2t+1	2t+2
1	1	2	0	t+1	t+2	t	2t+1	2t+2	2t
2	2	0	1	t+2	t	t+1	2t+2	2t	2t+1
t	t	t+1	t+2	2t	2t+1	2t+2	0	1	2
t+1	t+1	t+2	t	2t+1	2t+2	2t	1	2	0
t+2	t+2	t	t+1	2t+2	2t	2t+1	2	0	1
2t	2t	2t+1	2t+2	0	1	2	t	t+1	t+2
2t+1	2t+1	2t+2	2t	1	2	0	t+1	t+2	t
2t+2	2t+2	2t	2t+1	2	0	1	t+2	t	t+1

2190 As we can see, the group $(\mathbb{F}_3, +)$ is a subgroup of the group $(\mathbb{F}_{3^2}, +)$, obtained by only consid-
2191 ering the first three rows and columns of this table.

2192 We can use the addition table to deduce the additive inverse (the negative) of any element
2193 from \mathbb{F}_{3^2} . For example, in \mathbb{F}_{3^2} we have $-(2t+1) = t+2$, since $(2t+1) + (t+2) = 0$

Multiplication needs a bit more computation, as we first have to multiply the polynomials, and whenever the result has a degree ≥ 2 , we have to apply a polynomial division algorithm like 3 to divide the product by the polynomial P and keep the remainder. To see how this works, let us compute the product of $t+2$ and $2t+2$ in \mathbb{F}_{3^2} :

$$\begin{aligned} (t+2) \cdot (2t+2) &= (2t^2 + 2t + t + 1) \bmod (t^2 + 1) \\ &= (2t^2 + 1) \bmod (t^2 + 1) & \# 2t^2 + 1 : t^2 + 1 &= 2 + \frac{2}{t^2 + 1} \\ &= 2 \end{aligned}$$

2194 This means that the product of $t+2$ and $2t+2$ in \mathbb{F}_{3^2} is 2. Performing this computation for all
2195 elements gives the following multiplication table:

·	0	1	2	t	t+1	t+2	2t	2t+1	2t+2
0	0	0	0	0	0	0	0	0	0
1	0	1	2	t	t+1	t+2	2t	2t+1	2t+2
2	0	2	1	2t	2t+2	2t+1	t	t+2	t+1
t	0	t	2t	2	t+2	2t+2	1	t+1	2t+1
t+1	0	t+1	2t+2	t+2	2t	1	2t+1	2	t
t+2	0	t+2	2t+1	2t+2	1	t	t+1	2t	2
2t	0	2t	t	1	2t+1	t+1	2	2t+2	t+2
2t+1	0	2t+1	t+2	t+1	2	2t	2t+2	t	1
2t+2	0	2t+2	t+1	2t+1	t	2	t+2	1	2t

2196

2197 As it was the case in previous examples, we can use the table to deduce the multiplicative
 2198 inverse of any non-zero element from \mathbb{F}_{3^2} . For example, in \mathbb{F}_{3^2} we have $(2t+1)^{-1} = 2t+2$,
 2199 since $(2t+1) \cdot (2t+2) = 1$.

2200 From the multiplication table, we can also see that the only quadratic residues in \mathbb{F}_{3^2} are
 2201 from the set $\{0, 1, 2, t, 2t\}$, with $\sqrt{0} = \{0\}$, $\sqrt{1} = \{1, 2\}$, $\sqrt{2} = \{t, 2t\}$, $\sqrt{t} = \{t+2, 2t+1\}$ and
 2202 $\sqrt{2t} = \{t+1, 2t+2\}$.

Since \mathbb{F}_{3^2} is a field, we can solve equations as we would for other fields, (such as rational numbers). To see that, let us find all $x \in \mathbb{F}_{3^2}$ that solve the quadratic equation $(t+1)(x^2 + (2t+2)) = 2$. We compute as follows:

$$\begin{aligned}
 (t+1)(x^2 + (2t+2)) &= 2 && \# 2 \text{ distributive law} \\
 (t+1)x^2 + (t+1)(2t+2) &= 2 \\
 (t+1)x^2 + (t) &= 2 && \# 2 \text{ add the additive inverse of } t \\
 (t+1)x^2 + (t) + (2t) &= (2) + (2t) \\
 (t+1)x^2 &= 2t+2 && \# \text{ multiply with the multiplicative invers of } t+1 \\
 (t+2)(t+1)x^2 &= (t+2)(2t+2) && \# \text{ multiply with the multiplicative invers of } t+1 \\
 x^2 &= 2 && \# 2 \text{ is quadratic residue. Take the roots.} \\
 x &\in \{t, 2t\}
 \end{aligned}$$

2203 Computations in extension fields are arguably on the edge of what can reasonably be done with
 2204 pen and paper. Fortunately, Sage provides us with a simple way to do the computations.

```

2205 sage: Z3 = GF(3) # prime field 225
2206 sage: Z3t.<t> = Z3[] # polynomials over Z3 226
2207 sage: P = Z3t(t^2+1) 227
2208 sage: P.is_irreducible() 228
2209 True 229
2210 sage: F3_2.<t> = GF(3^2, name='t', modulus=P) # Extension 230
2211 field F_3^2
2212 sage: F3_2 231
2213 Finite Field in t of size 3^2 232
2214 sage: F3_2(t+2)*F3_2(2*t+2) == F3_2(2) 233
2215 True 234
2216 sage: F3_2(2*t+2)^(-1) # multiplicative inverse 235
2217 2*t + 1 236
2218 sage: # verify our solution to (t+1)(x^2 + (2t+2)) = 2 237
2219 sage: F3_2(t+1)*(F3_2(t)**2 + F3_2(2*t+2)) == F3_2(2) 238

```

2220 **True** 239
 2221 **sage:** `F3_2(t+1)*(F3_2(2*t)**2 + F3_2(2*t+2)) == F3_2(2)` 240
 2222 **True** 241

2223 *Exercise 52.* Consider the extension field \mathbb{F}_{3^2} from the previous example and find all pairs of
 2224 elements $(x, y) \in \mathbb{F}_{3^2}$, for which the following equation holds:

$$y^2 = x^3 + 4 \quad (4.42)$$

2225 *Exercise 53.* Show that the polynomial $Q = x^2 + x + 2$ from $\mathbb{F}_3[x]$ is irreducible. Construct the
 2226 multiplication table of \mathbb{F}_{3^2} with respect to Q and compare it to the multiplication table of \mathbb{F}_{3^2}
 2227 from example 66.

2228 *Exercise 54.* Show that the polynomial $P = t^3 + t + 1$ from $\mathbb{F}_5[t]$ is irreducible. Then consider the
 2229 extension field \mathbb{F}_{5^3} defined relative to P . Compute the multiplicative inverse of $(2t^2 + 4) \in \mathbb{F}_{5^3}$
 2230 using the extended Euclidean algorithm. Then find all $x \in \mathbb{F}_{5^3}$ that solve the following equation:
 2231

$$(2t^2 + 4)(x - (t^2 + 4t + 2)) = (2t + 3) \quad (4.43)$$

2232 *Exercise 55.* Consider the prime field \mathbb{F}_5 . Show that the polynomial $P = x^2 + 2$ from $\mathbb{F}_5[x]$ is
 2233 irreducible. Implement the finite field \mathbb{F}_{5^2} in sage.

2234 4.4 Projective Planes

2235 Projective planes are particular geometric constructs defined over a given field. In a sense,
 2236 projective planes extend the concept of the ordinary Euclidean plane by including “points at
 2237 infinity.”

2238 To understand the idea of constructing of projective planes, note that in an ordinary Eu-
 2239 clidean plane, two lines either intersect in a single point or are parallel. In the latter case, both
 2240 lines are either the same, that is, they intersect in all points, or do not intersect at all. A projec-
 2241 tive plane can then be thought of as an ordinary plane, but equipped with additional “point at
 2242 infinity” such that two different lines always intersect in a single point. Parallel lines intersect
 2243 “at infinity”.

2244 Such an inclusion of infinity points makes projective planes particularly useful in the de-
 2245 scription of elliptic curves, as the description of such a curve in an ordinary plane needs an
 2246 additional symbol “the point at infinity” to give the set of points on the curve the structure of a
 2247 group 5.1. Translating the curve into projective geometry includes this “point at infinity” more
 2248 naturally into the set of all points on a projective plane.

2249 To be more precise, let \mathbb{F} be a field, $\mathbb{F}^3 := \mathbb{F} \times \mathbb{F} \times \mathbb{F}$ the set of all tuples of three elements
 2250 over \mathbb{F} and $x \in \mathbb{F}^3$ with $x = (X, Y, Z)$. Then there is exactly one *line* L_x in \mathbb{F}^3 that intersects both
 2251 $(0, 0, 0)$ and x , given by the set $L_x = \{(k \cdot X, k \cdot Y, k \cdot Z) \mid k \in \mathbb{F}\}$. A point in the **projective plane**
 2252 over \mathbb{F} can then be defined as such a **line** if we exclude the intersection of that line with $(0, 0, 0)$.
 2253 This leads to the following definition of a **point** in projective geometry:

$$[X : Y : Z] := \{(k \cdot X, k \cdot Y, k \cdot Z) \mid k \in \mathbb{F}^*\} \quad (4.44)$$

2254 Points in projective geometry are therefore lines in \mathbb{F}^3 where the intersection with $(0, 0, 0)$ is
 2255 excluded. Given a field \mathbb{F} the **projective plane** of that field is then defined as the set of all
 2256 points, excluding the point $[0 : 0 : 0]$:

$$\mathbb{FP}^2 := \{[X : Y : Z] \mid (X, Y, Z) \in \mathbb{F}^3 \text{ with } (X, Y, Z) \neq (0, 0, 0)\} \quad (4.45)$$

It can be shown that a projective plane over a finite field \mathbb{F}_{p^m} contains $p^{2m} + p^m + 1$ number of elements.

To understand why the projective point $[X : Y : Z]$ is also a line, consider the situation where the underlying field \mathbb{F} is the set of rational numbers \mathbb{Q} . In this case, \mathbb{Q}^3 can be seen as the three-dimensional space, and $[X : Y : Z]$ is an ordinary line in this 3-dimensional space that intersects zero and the point with coordinates X, Y and Z such that the intersection with zero is excluded.

The key observation here is that points in the projective plane \mathbb{P}^2 are lines in the 3-dimensional space \mathbb{F}^3 . However it should be kept in mind that, for finite fields, the terms **space** and **line** share very little visual similarity with their counterparts over the set of rational numbers.

It follows from this that points $[X : Y : Z] \in \mathbb{P}^2$ are not simply described by fixed coordinates (X, Y, Z) , but by **sets of coordinates**, where two different coordinates (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) describe the same point if and only if there is some non-zero field element $k \in \mathbb{F}^*$ such that $(X_1, Y_1, Z_1) = (k \cdot X_2, k \cdot Y_2, k \cdot Z_2)$. Points $[X : Y : Z]$ are called **projective coordinates**.

Notation and Symbols 12 (Projective coordinates). Projective coordinates of the form $[X : Y : 1]$ are descriptions of so-called **affine points**. Projective coordinates of the form $[X : Y : 0]$ are descriptions of so-called **points at infinity**. In particular, the projective coordinate $[1 : 0 : 0]$ describes the so-called **line at infinity**.

Example 67. Consider the field \mathbb{F}_3 from exercise 48. As this field only contains three elements, it does not take too much effort to construct its associated projective plane $\mathbb{F}_3\mathbb{P}^2$, as we know that it only contains 13 elements.

To find $\mathbb{F}_3\mathbb{P}^2$, we have to compute the set of all lines in $\mathbb{F}_3 \times \mathbb{F}_3 \times \mathbb{F}_3$ that intersect $(0, 0, 0)$, excluding their intersection with $(0, 0, 0)$. Since those lines are parameterized by tuples (x_1, x_2, x_3) ,

we compute as follows:

$$\begin{aligned}
[0 : 0 : 1] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(0, 0, 1), (0, 0, 2)\} \\
[0 : 0 : 2] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(0, 0, 2), (0, 0, 1)\} = [0 : 0 : 1] \\
[0 : 1 : 0] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(0, 1, 0), (0, 2, 0)\} \\
[0 : 1 : 1] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(0, 1, 1), (0, 2, 2)\} \\
[0 : 1 : 2] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(0, 1, 2), (0, 2, 1)\} \\
[0 : 2 : 0] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(0, 2, 0), (0, 1, 0)\} = [0 : 1 : 0] \\
[0 : 2 : 1] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(0, 2, 1), (0, 1, 2)\} = [0 : 1 : 2] \\
[0 : 2 : 2] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(0, 2, 2), (0, 1, 1)\} = [0 : 1 : 1] \\
[1 : 0 : 0] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(1, 0, 0), (2, 0, 0)\} \\
[1 : 0 : 1] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(1, 0, 1), (2, 0, 2)\} \\
[1 : 0 : 2] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(1, 0, 2), (2, 0, 1)\} \\
[1 : 1 : 0] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(1, 1, 0), (2, 2, 0)\} \\
[1 : 1 : 1] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(1, 1, 1), (2, 2, 2)\} \\
[1 : 1 : 2] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(1, 1, 2), (2, 2, 1)\} \\
[1 : 2 : 0] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(1, 2, 0), (2, 1, 0)\} \\
[1 : 2 : 1] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(1, 2, 1), (2, 1, 2)\} \\
[1 : 2 : 2] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(1, 2, 2), (2, 1, 1)\} \\
[2 : 0 : 0] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(2, 0, 0), (1, 0, 0)\} = [1 : 0 : 0] \\
[2 : 0 : 1] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(2, 0, 1), (1, 0, 2)\} = [1 : 0 : 2] \\
[2 : 0 : 2] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(2, 0, 2), (1, 0, 1)\} = [1 : 0 : 1] \\
[2 : 1 : 0] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(2, 1, 0), (1, 2, 0)\} = [1 : 2 : 0] \\
[2 : 1 : 1] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(2, 1, 1), (1, 2, 2)\} = [1 : 2 : 2] \\
[2 : 1 : 2] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(2, 1, 2), (1, 2, 1)\} = [1 : 2 : 1] \\
[2 : 2 : 0] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(2, 2, 0), (1, 1, 0)\} = [1 : 1 : 0] \\
[2 : 2 : 1] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(2, 2, 1), (1, 1, 2)\} = [1 : 1 : 2] \\
[2 : 2 : 2] &= \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}_3^*\} = \{(2, 2, 2), (1, 1, 1)\} = [1 : 1 : 1]
\end{aligned}$$

These lines define the 13 points in the projective plane $\mathbb{F}_3\mathbb{P}$:

$$\begin{aligned}
\mathbb{F}_3\mathbb{P} = \{ & [0 : 0 : 1], [0 : 1 : 0], [0 : 1 : 1], [0 : 1 : 2], [1 : 0 : 0], [1 : 0 : 1], \\
& [1 : 0 : 2], [1 : 1 : 0], [1 : 1 : 1], [1 : 1 : 2], [1 : 2 : 0], [1 : 2 : 1], [1 : 2 : 2] \}
\end{aligned}$$

2278 This projective plane contains 9 affine points, three points at infinity and one line at infinity.

2279 To understand the ambiguity in projective coordinates a bit better, let us consider the point
 2280 $[1 : 2 : 2]$. As this point in the projective plane is a line in $\mathbb{F}_3^3 \setminus \{(0, 0, 0)\}$, it has the projective
 2281 coordinates $(1, 2, 2)$ as well as $(2, 1, 1)$, since the former coordinate gives the latter when multi-
 2282 plied in \mathbb{F}_3 by the factor 2. In addition, note that, for the same reasons, the points $[1 : 2 : 2]$ and
 2283 $[2 : 1 : 1]$ are the same, since their underlying sets are equal.

2284 *Exercise 56.* Construct the so-called **Fano plane**, that is, the projective plane over the finite
 2285 field \mathbb{F}_2 .

Bibliography

- Jens Groth. On the size of pairing-based non-interactive arguments. *IACR Cryptol. ePrint Arch.*, 2016:260, 2016. URL <http://eprint.iacr.org/2016/260>.
- P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi: 10.1109/SFCS.1994.365700.
- David Fifield. The equivalence of the computational diffie–hellman and discrete logarithm problems in certain groups, 2012. URL <https://web.stanford.edu/class/cs259c/finalpapers/dlp-cdh.pdf>.
- Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. ISBN 978-3-540-46766-3. URL <https://fmouhart.epheme.re/Crypto-1617/TD08.pdf>.
- Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. *Cryptology ePrint Archive, Report 2016/492*, 2016. <https://ia.cr/2016/492>.