# Moonmath manual

TechnoBob and the Least Scruples crew

September 6, 2021

Lorem **ipsum** dolor sit amet, consectetur adipiscing elit. Pellentesque semper viverra dictum. Fusce interdum venenatis leo varius vehicula. Etiam ac massa dolor. Quisque vel massa faucibus, facilisis nulla nec, egestas lectus. Sed orci dui, egestas non felis vel, fringilla pretium odio. *Aliquam* vel consectetur felis. Suspendisse justo massa, maximus eget nisi a, maximus gravida mi.

Here is a citation for demonstration: **?**

# 1 Introduction

This is dump from other papers as inspiration for the intro:

Zero-knowledge proofs (ZKPs) are an important privacy-enhancing tool from cryptography. Theyallow proving the veracity of a statement, related to confidential data, without revealing any in-formation beyond the validity of the statement. ZKPs were initially developed by the academiccommunity in the 1980s, and have seen tremendous improvements since then. They are now ofpractical feasibility in multiple domains of interest to the industry, and to a large community ofdevelopers and researchers. ZKPs can have a positive impact in industries, agencies, and for per-sonal use, by allowing privacy-preserving applications where designated private data can be madeuseful to third parties, despite not being disclosed to them.

ZKP systems involve at least two parties: a prover and a verifier. The goal of the prover is toconvince the verifier that a statement is true, without revealing any additional information. Forexample, suppose the prover holds a birth certificate digitally signed by an authority. In orderto access some service, the prover may have to prove being at least 18 years old, that is, thatthere exists a birth certificate, tied to the identify of the prover and digitally signed by a trustedcertification authority, stating a birthdate consistent with the age claim. A ZKP allows this, withoutthe prover having to reveal the birthdate.

## 1.1 Target audience

This book is accessible for both beginners and experienced developers alike. Concepts are gradually introduced in a logical and steady pace. Nonetheless, the chapters lend themselves rather well to being read in a different order. More experienced developers might get the most benefit by jumping to the chapters that interest them most. If you like to learn by example, then you should go straight to the chapter on Using Clarinet.

It is assumed that you have a basic understanding of programming and the underlying logical concepts. The first chapter covers the general syntax of Clarity but it does not delve into what programming itself is all about. If this is what you are looking for, then you might have a more difficult time working through this book unless you have an (undiscovered) natural affinity for such topics. Do not let that dissuade you though, find an introductory programming book and press on! The straightforward design of Clarity makes it a great first language to pick up.

# 2 The Zoo of Zero-Knowledge Proofs

First, a list of zero-knowledge proof systems:

1. Pinocchio (2013): Paper
   - Notes: trusted setup
2. BCGTV (2013): Paper
   - Notes: trusted setup, implementation
3. BCTV (2013): Paper
   - Notes: trusted setup, implementation
4. Groth16 (2016): Paper
   - Notes: trusted setup
   - Other resources: Talk in 2019 by Georgios Konstantopoulos
5. GM17 (207): Paper
   - Notes: trusted setup
   - Other resources: later Simulation extractability in ROM, 2018
6. Bulletproofs (2017): Paper
   - Notes: no trusted setup
   - Other resources: Polynomial Commitment Scheme on DL, 2016 and KZG10, Polynomial Commitment Scheme on Pairings, 2010
7. Ligero (2017): Paper
   - Notes: no trusted setup
   - Other resources:
8. Hyrax (2017): Paper
   - Notes: no trusted setup
   - Other resources:
9. STARKs (2018): Paper
   - Notes: no trusted setup
   - Other resources:
10. Aurora (2018): Paper
    - Notes: transparent SNARK
    - Other resources:

11. Sonic (2019): Paper
    – Notes: SNORK - SNARK with universal and updateable trusted setup, PCS-based
    – Other resources: Blog post by Mary Maller from 2019 and work on updateable and universal setup from 2018
12. Libra (2019): Paper
    – Notes: trusted setup
    – Other resources:
13. Spartan (2019): Paper
    – Notes: transparent SNARK
    – Other resources:
14. PLONK (2019): Paper
    – Notes: SNORK, PCS-based
    – Other resources: Discussion on Plonk systems and Awesome Plonk list
15. Halo (2019): Paper
    – Notes: no trusted setup, PCS-based, recursive
    – Other resources:
16. Marlin (2019): Paper
    – Notes: SNORK, PCS-based
    – Other resources: Rust Github
17. Fractal (2019): Paper
    – Notes: Recursive, transparent SNARK
    – Other resources:
18. SuperSonic (2019): Paper
    – Notes: transparent SNARK, PCS-based
    – Other resources: Attack on DARK compiler in 2021
19. Redshift (2019): Paper
    – Notes: SNORK, PCS-based
    – Other resources:

**Other resources on the zoo:** Awesome ZKP list on Github, ZKP community with the reference document

## To Do List

- Make table for prover time, verifier time, and proof size
- Think of categories - *Achieved Goals*: Trusted setup or not, Post-quantum or not, ...
- Think of categories - *Mathematical background*: Polynomial commitment scheme, ...
- ... while we discuss the points above, we should also discuss a common notation/language for all these things. (E.g. transparent SNARK/no trusted setup/STARK)

## Points to cover while writing

- Make a historical overview over the "discovery" of the different ZKP systems
- Make reader understand what paper is build on what result etc. - the tree of publications!
- Make reader understand the different terminology, e.g. SNARK/SNORK/STARK, PCS, R1CS, updateable, universal, ...
- Make reader understand the mathematical assumptions - and what this means for the zoo.
- Where will the development/evolution go? What are bottlenecks?

#### Other topics I fell into while compiling this list

- Vector commitments: `https://eprint.iacr.org/2020/527.pdf`
- Snarkl: `http://ace.cs.ohio.edu/~gstewart/papers/snaarkl.pdf`
- Virgo?: `https://people.eecs.berkeley.edu/~kubitron/courses/cs262a-F19/projects/reports/project5_report_ver2.pdf`

# 3 Preliminaries

## 3.1 Purpose of the book

The first version of this book is written by security auditors at Least Authority where we audited quite a few snark based systems. Its included "what we have learned" destilate of the time we spend on various audits.

We intend to let illus- trative examples drive the discussion and present the key concepts of pairing computation with as little machinery as possible. For those that are fresh to pairing-based cryptography, it is our hope that this chapter might be particu- larly useful as a first read and prelude to more complete or advanced expositions (e.g. the related chapters in [Gal12]).

On the other hand, we also hope our beginner-friendly intentions do not leave any sophisticated readers dissatisfied by a lack of formality or generality, so in cases where our discussion does sacrifice completeness, we will at least endeavour to point to where a more thorough exposition can be found.

One advantage of writing a survey on pairing computation in 2012 is that, after more than a decade of intense and fast-paced research by mathematicians and cryptographers around the globe, the field is now racing towards full matu- rity. Therefore, an understanding of this text will equip the reader with most of what they need to know in order to tackle any of the vast literature in this remarkable field, at least for a while yet.

Since we are aiming the discussion at active readers, we have matched every example with a corresponding snippet of (hyperlinked) Magma [BCP97] code 1 , where we take inspiration from the helpful Magma pairing tutorial by Dominguez Perez et al. [DKS09].

Early in the book we will develop examples that we then later extend with most of the things we learn in each chapter. This way we incrementally build a few real world snarks but over full fledged cryptographic systems that are nevertheless simple enough to be computed by pen and paper to illustrate all steps in grwat detail.

## 3.2 How to read this book

Books and papers to read: XXXXXXXXXXXX

Software to try: XXXXXXXXXXXXXXXXXXXX

Correctly prescribing the best reading route for a beginner naturally requires individual diagnosis that depends on their prior knowledge and technical preparation.

## 3.3 Cryptological Systems

The science of information security is referred to as *cryptology*. In the broadest sense, it deals with encryption and decryption processes, with digital signatures, identification protocols, cryptographic hash functions, secrets sharing, electronic voting procedures and electronic money. EXPAND

## 3.4 SNARKS

## 3.5 complexity theory

Before we deal with the mathematics behind zero knowledge proof systems, we must first clarify what is meant by the runtime of an algorithm or the time complexity of an entire mathematical problem. This is particularly important for us when we analyze the various snark systems...

For the reader who is interested in complexity theory, we recommend, or example or , as well as the references contained therein.

### 3.5.1 Runtime complexity

The runtime complexity of an algorithm describes, roughly speaking, the amount of elementary computation steps that this algorithm requires in order to solve a problem, depending on the size of the input data.

Of course, the exact amount of arithmetic operations required depends on many factors such as the implementation, the operating system used, the CPU and many more. However, such accuracy is seldom required and is mostly meaningful to consider only the asymptotic computational effort.

In computer science, the runtime of an algorithm is therefore not specified in individual calculation steps, but instead looks for an upper limit which approximates the runtime as soon as the input quantity becomes very large. This can be done using the so-called *Landau notation* (also called big -$\mathcal{O}$-notation) A precise definition would, however, go beyond the scope of this work and we therefore refer the reader to .

For us, only a rough understanding of transit times is important in order to be able to talk about the security of crypographic systems. For example, $\mathcal{O}(n)$ means that the running time of the algorithm to be considered is linearly dependent on the size of the input set $n$, $\mathcal{O}(n^k)$ means that the running time is polynomial and $\mathcal{O}(2^n)$ stands for an exponential running time (chapter 2.4).

An algorithm which has a running time that is greater than a polynomial is often simply referred to as *slow*.

A generalization of the runtime complexity of an algorithm is the so-called *time complexity of a mathematical problem*, which is defined as the runtime of the fastest possible algorithm that can still solve this problem ( chapter 3.1).

Since the time complexity of a mathematical problem is concerned with the runtime analysis of all possible (and thus possibly still undiscovered) algorithms, this is often a very difficult and deep-seated question .

For us, the time complexity of the so-called discrete logarithm problem will be important. This is a problem for which we only know slow algorithms on classical computers at the moment, but for which at the same time we cannot rule out that faster algorithms also exist.

## 3.6 Hash functions

We assume that $H : \{0,1\}^* \to 0,1^k$ is a **hash function** that maps binary strings of arbitrary length onto strings of length $k$. In addition we define a hash function to be $l$-bounded if it is only able to map from binary strings of length $l$ to binary strings of length $k$.

STUFF ON CRYPTOGRAPHIC HASH FUNCTIOND

**p&p-hash** In this example we define a 16-bounded pen&paper hash function that is simple enough to be computed without a computer. We call it the PaP-Hash and will use it throughout the book as a basic example whenever hashing is involved in other example.

The PaP-Hash $\mathscr{H}_{PaP} : \{0,1\}^{16} \to \{0,1\}^4$ is defined in the following way:

- Decompose the 16-bit preimage $S = (s_0, s_1, \ldots, s_{15})$ into 4 chunks $S_i = (s_{4i+0}, \ldots, s_{4i+3})$ for $i \in \{0,1,2,3\}$.
- For each chunk $S_i$ do a circular bitshift $s_j \to s_{j+1} \bmod 4$ for all $s_j \in S_i$
- Xor all four chunks together $S = S_1\, XOR\, S_2\, XOR\, S_3\, XOR\, S_3$
- Compute the result $\mathscr{H}_{PaP}(S) = S\, XOR\, (1001)$

**Example 1.** *Lets compute our PaP-Hash on a concrete example string* $S = (1110011101110011)$. *Then the decomposition is* $S_0 = (1110)$, $S_1 = (0111)$, $S_2 = (0111)$ *and* $S_3 = (0011)$ *and after a circular bitshift we get* $S'_0 = (0111)$, $S'_1 = (1011)$, $S'_2 = (1011)$ *and* $S'_3 = (1001)$. *Xoring everything together we get* $S = (0111)\, XOR\, (1011)\, XOR\, (1011)\, XOR\, (1001) = (1100)\, XOR\, (0010) = (1110)$. *So we get* $\mathscr{H}_{PaP}(1010011101100011) = (1110)$.

## 3.7 Software Used in This Book

### 3.7.1 Sagemath

It order to provide an interactive learning experience, and to allow getting hands-on with the concepts described in this book, we give examples for how to program them in the Sage programming language. Sage is a dialect of the learning-friendly programming language Python, which was extended and optimized for computing with, in and over algebraic objects. Therefore, we recommend installing Sage before diving into the following chapters.

The installation steps for various system configurations are described on the sage websit [1]. Note however that we use Sage version 9, so if you are using Linux and your package manager only contains version 8, you may need to choose a different installation path, such as using prebuilt binaries.

We recommend the interested reader, who is not familiar with sagemath to read on the many tutorial before starting this book. For example

[1] `https://doc.sagemath.org/html/en/installation/index.html`

# 4 Arithmetics

How much mathematics is needed to understand zero knowledge proofs? The answer, of course, depends on many things like the level of detail the reader want to understand them. For example it is possible to describe those proofs not using mathematics at all. However to read a foundational paper like [GROTH16], enough mathematics is needed to at least understand the basic concepts.

Otherwise any student who is interested in learning the concepts, but who has never seen or played with, say, a finite field, or an elliptic curve, may quickly become overwhelmed. This is not so much due to the complexity of the mathematics needs but perhaps more because of the vast amount of technical jargon, of unknown terms, obscure symbols that quickly makes a text ubreadable, desipte the concepts being actually not that hard. As a result the reader might either loose interest, or gain some dangerous smattering that in a worst case scenario materialize in inmature code.

In this chapter we therefore derive the mathematical concepts needed to understand the basic concepts underlying snark development and we encourage the reader who is not familiar with basic number theory and lliptic curves to take the time and read this chapter until they are able to at least solve most of the simple exercises.

If on the other hand the reader is already skilled in elliptic curve cryptography they might skip this section and only come back for reference and comparision. Maybe the most interesing parts are XXX.

We start at a very basic level and only really requie fundamntal concepts like integer arithmetics. At the same time we'll have a focus on teaching the reader how to think mathematically and to understand that there are numbers and methatical structures out there that appear to be very different from the stuff we learned in school and yet on a deeper level they are in deed very similar.

We want to stress however, that our introduction is informal, incomplete and optimized to enable the reader to understand zero knowledge concepts as efficient as possible. Our focus and design choices are so that we give as little theory as necessary but accompanied by a wealth of numerical examples. We found this on the believe, that such an informal, example- driven approach to learning mathematics may ease the beginner's digestion in the initial stages.

For instance, a beginner would be likely to find it more beneficial to first compute a simple toy snark in a pen and paper style all the way through all steps before they dig deeper and actually devop real world production ready systems. Also having already a few simple examples in you head, its likely easier to only then read the actual academic papers.

However in order to be able to derive those toy example, some mathematical groundwork is needed. This chapter therefore will help the reader to focus on what is important, while at the same time serve as first exercises the reader is encouraged to recompute themself. Every section usually then ends with a list of additional exercises in increasing difficulty order, to help the reader memorising and applying the concepts given.

Overall the goal of this chapter is to provide a reader who is starting with nothing more than basic high school algebra, to be able to solve basic tasks in elliptic curve cryptography without the need of a computer.

We start with a brief recapitulation of basic integer aithmetics like long division, the greatest common divisior and Euklids algorithm. 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.

After this practical warm up, we have to introduce some basic algebraic terms like groups and fields, because those terms are all over the place when reading academic papers in the context of zero knowledge proof. The beginner is good adviced to memorize those terms and think about them. We define these terms in the general abstract way of mathematics, hoping that the non mathematical trained reader will gradually learn to become comfortable with this style. We then give basic examples and do basic computations with these examples to get familiar with the concepts.

## 4.1 Integer Arithmetics

In a sense, integer arithmetics is at the hart of the foundation of large parts of modern cryptography as it provides the most basic tools to do computations in those systems. Fortunately, most readers will probably remember integer arithmetics from school. It is however important for the rest of the book to be able to apply those concepts to understand and execute computations in the various pen and paper examples that are the main contribution of the moon math manual. We will therefore recapitulate those concepts filling up some knowledge gaps.

In what follows we applay standard mathematical notations and use the symbol $\mathbb{Z}$ for the set of all integers, that is we write

$$\mathbb{Z} := \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\} \tag{4.1}$$

So whenever you see the symbol $\mathbb{Z}$, think of the set of all integers. If $a \in \mathbb{Z}$ is an integer, we write $|a|$ for the *absolute value* of $a$, that is the the non-negative value of $a$ without regard to its sign. In addition we will use the symbol $\mathbb{N}$ for the set of all counting numbers, that is we write

$$\mathbb{N} := \{0, 1, 2, 3, \ldots\} \tag{4.2}$$

including the number 0. So whenever you see the symbol $\mathbb{N}$, think of the set of all non negative integers.

To make it easier to memorize new concepts and symbols, we might frequently link to definitions (See 4.2 for a definition of $\mathbb{Z}$) in the begining, but as to many links render a text unreadable, we will assume the reader will become familiar with definitions as the text proceeds at which point we will not link them anymore.

Both sets $\mathbb{N}$ and $\mathbb{Z}$ have a notion of addition as well as multiplication dedined on them and also most of us are probably able to do many integer computations in their head, we will frequently invoke the sagemath system (3.7.1) for more complicated computations. One way to invoke the integer-type in sage is:

```
sage: ZZ # A sage notation for the integer type
Integer Ring
sage: NN # A sage notation for the counting number type
Non negative integer semiring
sage: ZZ(5) # Get an element from the Ring of integers
5
```

```
sage: ZZ(5) + ZZ(3)
8
sage: ZZ(5) * NN(3)
15
sage: ZZ.random_element(10**50)
92039041092082299416745072692640051541444470853017
sage: ZZ(27713).str(2) # Binary string representation
110110001000001
sage: NN(27713).str(2) # Binary string representation
110110001000001
sage: ZZ(27713).str(16) # Hexadecimal string representation
6c41
```

Of particular interest for us are the so called *prime numbers*, which are counting numbers $p \in \mathbb{N}$ with $p \geq 2$, which are divisible by themself and by 1 only. Prime numbers are called *odd* if they are not the number 2. We write $\mathbb{P}$ for the set of all prime numbers and $\mathbb{P}_{\geq 3}$ for the set of all odd prime numbers. $\mathbb{P}$ is infinite and can be ordered according to size, so that we can writem them as

$$2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,\ldots \tag{4.3}$$

which is sequence $A000040$ in OEIS. In particular, we can talk about small and large prime numbers.

As the *fundamental theorem of arithmetics* 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 $n \in \mathbb{N}_{\geq 2}$ be any natural number. Then there are always prime numbers $p_1, p_2, \ldots, p_k \in \mathbb{P}$, such that

$$n = p_1 \cdot p_2 \cdot \ldots \cdot p_k \, . \tag{4.4}$$

This representation is unique, except for permutations in the factors and is called the **prime factorization** of $n$.

**Example 2** (Prime Factorization)**.** *To see what we mean by prime factorization of a number, lets look at the number* $19214758032624000 \in \mathbb{N}$. *To get its prime factors, we can sucessively divide it by all prime numbers in ascending order starting with* 2*. We get*

$$19214758032624000 = 2\cdot2\cdot2\cdot2\cdot2\cdot2\cdot2\cdot3\cdot3\cdot3\cdot5\cdot5\cdot5\cdot7\cdot11\cdot17\cdot17\cdot23\cdot43\cdot43\cdot47$$

*We can double check our findings invoking sage, which provides an algorith to factor counting numbers:*

```
sage: n = NN(19214758032624000)
sage: factor(n)
2^7 * 3^3 * 5^3 * 7 * 11 * 17^2 * 23 * 43^2 * 47
```

Having done the computation from the previous example, reveals an important observation: Computing the factorization was computationally expensive, while on the other hand, giving a string of prime numbers, computing their product is fast.

From this an important question arises: How fast we can compute the prime factorization of a natural number? This is the famous *factorization problem* and as far as we know, there is no method on a classical Turing machine that is able to compute this representation in polynomial time. The fastest algorithms known today run sub-exponentially, with $\mathcal{O}((1+\varepsilon)^n)$ and some $\varepsilon > 0$.

It follows that number factorization $\Leftrightarrow$ prime number multiplication is an example of, what is called a one-way function. Something that is easy to compute in one direction, but hard to compute in the other direction. Existence of one way functions like this are basic crytographic assumptions, that the security of many crypto systems is based on.

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

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

**Exercise 2.** *Compute the factorization of* $6469693230$ *and double check your results using sage.*

**Exercise 3.** *Consider the following equation* $4 \cdot x + 21 = 5$*. Compute the set of all solutions x under the following assumptions: 1. The equation is defined over the type of natural numbers. 2. The equation is defined over the type 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 type of natural numbers. 2. The equation is defined over the type of integers. 2. The equation is defined over the type* $\mathbb{Q}$ *of fractions.*

**Euklidean Division** In general there is no division defined in the usual sense for integers, as for example 7 divided by 3 will not be an integer again. However it is possible to devide any two integers with a remainder. So for example 7 divided by 3 is equal to 2 with a remainder of 1, since $7 = 2 \cdot 3 + 1$.

Doing integer division like this is probably something many of us remember from school. It is usually called *Euclidean division*, or division with remainder and it is an important technique, that every reader must become familiar with to understand 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 counting number $r \in \mathbb{N}$, with $0 \leq r < |b|$ such that

$$a = m \cdot b + r \tag{4.5}$$

This decomposition of $a$ given $b$ is called *Euklidean division*, where $a$ is called the *divident*, $b$ is called the *divisor*, $m$ is called the *quotient* and $r$ is called the *remainder*.

**Notation and Symbols 1.** *Suppose that the numbers* $a, b, m$ *and* $r$ *satisfy equation (4.5). Then we often write*

$$a \; div \; b := m, \qquad a \; mod \; b := r \tag{4.6}$$

*to describe the quotient and the remainder of the Euklidean division. We also say, that an integer a is divisible by another integer b if* $a \; mod \; b = 0$ *holds. In this case we also write* $b|a$*.*

So in a Nutshell Euclidean division is a 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. It can be shown, that both the quotient and the remainder always exist and are unique, as long as the divident is different from 0.

A special situation occures, is the remainder is zero, because in this special case the divident *is* divisible by the divisor. Our notation $b|a$ refelcts that.

**Example 3.** *Applying Euklidean division and our previously defined notation 4.25 to the divisor* $-17$ *and the divident* $4$*, we get*

$$-17\ div\ 4 = -5, \qquad -17\ mod\ 4 = 3$$

*because* $-17 = -5 \cdot 4 + 3$ *is the Euklidean division of* $-17$ *and* $4$ *(Since the remainder is by definition a non-negative number). In this case* $4$ *does not divide* $-17$ *as the reminder is not zero. Writing* $4|-17$ *therefore has no meaning. On the other hand we can write* $4|12$*, since* $4$ *divides* $12$*, as* $12\ mod\ 4 = 0$*. We can invoke sagemath to do the computation for us. We get*

```
sage: ZZ(-17) // ZZ(4) # Integer quotient
-5
sage: ZZ(-17) % ZZ(4) # remainder
3
sage: ZZ(4).divides(ZZ(-17)) # self divides other
False
sage: ZZ(4).divides(ZZ(12))
True
```

Methods to compute Euklidean division for integers are called *integer division algorithms*. Probably the best known algorithm is the so called *long division*, that most of us might have learned in school. It should be noted however that there are faster methods like *Newton–Raphson division*.

As long division is the standard method used for pen-&-paper division of multi-digit numbers expressed in decimal notation, the reader should become familiar with it as we use it all over this book when we do simple pen-and-paper computations. However instead of defining the algorithm formally, we rather give some examples, that hopelly will make the process clear

**Example 4** (Integer Long Division)**.** *To give an example of integer long division algorithm, lets divide the integer* $a = 143785$ *by the number* $b = 17$*. Our goal is therfore to find solutions to equation 4.5, that is we need to find the quotient* $m \in \mathbb{Z}$ *and the reminder* $r \in \mathbb{N}$ *such that* $143785 = m \cdot 17 + r$*. Using a notation that is mostly used in Commonwealth countries, we compute*

$$\begin{array}{r} 8457 \\ 17\overline{)143785} \\ \underline{136\phantom{000}} \\ 77\phantom{00} \\ \underline{68\phantom{00}} \\ 98\phantom{0} \\ \underline{85\phantom{0}} \\ 135 \\ \underline{119} \\ 16 \end{array} \tag{4.7}$$

*We therefore get* $m = 8457$ *as well as* $r = 16$ *and indeed we have* $143785 = 8457 \cdot 17 + 16$*, which we can double check invoking sage:*

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

```
True
```

*In a nutshell, the algorithm loops through the digits of the divident 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 divident.*

**Exercise 5** (Integer Long Division)**.** *Find an $m \in \mathbb{Z}$ as well as an $r \in \mathbb{N}$ such that $a = m \cdot b + r$ holds for the folling pairs $(a,b) = (27,5)$, $(a,b) = (27,-5)$, $(a,b) = (127,0)$, $(a,b) = (-1687,11)$ and . In which cases are your solutions unique?*

$(a,b) = (0,7)$

**Exercise 6** (Long Division Algorithm)**.** *Write an algorithm in pseudocode that computes integer long division, handling all edge cases properly.*

**The Extended Euklidean Algorithm** One of the most critical parts in this book is modular arithmetics XXX and its application in the computations in so called finite fields, as we explain in XXX. In modular arithmetics it is sometimes possible to define actual division and multiplicative inverses of numbers, that is very different from inverses as we know them from other systems like factional numbers.

However, to actually compute those inverses we have to get familar with the so-called *extended Euclidean algorithm*. To recapitulate jargon first, the *greatest common divisor* (GCD) of two nonzero integers $a$ and $b$ is the greatest non-zero counting number $d$ such that $d$ divides both $a$ and $b$; that is $d|a$ as well as $d|b$. We write $gcd(a,b) := d$ for this number. In addition two counting numbers are called **relative prime** or **coprime**, if their greates common divisor is 1.

The extended Euclidean algorithm is then a method to calculate the greatest common divisor of two counting numbers $a$ and $b \in \mathbb{N}$, as well as two additional integers $s, t \in \mathbb{Z}$, such that the equation

$$gcd(a,b) = s \cdot a + t \cdot b \tag{4.8}$$

holds. The following pseudocode shows in detail how to calculate these numbers with the extended Euclidean algorithm:

The algorithm is simple enough to be done effectively in pen-&-paper examples, where it is common to write it as a table where the rows represent the while-loop and the colums represent the values of the the array $r$, $s$ and $t$ with index $k$. The following example provides a simple execution:

**Example 5.** *To illustrate the algorithm, lets apply it to the numbers $a = 12$ and $b = 5$. Since $12, 5 \in \mathbb{N}$ as well as $12 \geq 5$ all requirements are meat and we compute*

| $k$ | $r_k$ | $s_k$ | $t_k = (r_k - s_k \cdot a)\ div\ b$ |
|---|---|---|---|
| 0 | 12 | 1 | 0 |
| 1 | 5 | 0 | 1 |
| 2 | 2 | 1 | -2 |
| 3 | 1 | -2 | 5 |

*From this we can see that 12 and 5 are relatively prime (coprime), since their greatest common divisor 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)
```

**Algorithm 1** Extended Euklidean Algorithm

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

**procedure** EXT-EUCLID$(a,b)$

$r_0 \leftarrow a$

$r_1 \leftarrow b$

$s_0 \leftarrow 1$

$s_1 \leftarrow 0$

$k \leftarrow 1$

**while** $r_k \neq 0$ **do**

$q_k \leftarrow r_{k-1}$ div $r_k$

$r_{k+1} \leftarrow r_{k-1} - q_k \cdot r_k$

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

$k \leftarrow k+1$

**end while**

**return** $gcd(a,b) \leftarrow r_{k-1}$, $s \leftarrow s_{k-1}$ and $t := (r_{k-1} - s_{k-1} \cdot a)$ div $b$

**end procedure**

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

**Exercise 7** (Extended Euklidean Algorithm)**.** *Find integers* $s, t \in \mathbb{Z}$ *such that* $gcd(a,b) = s \cdot a + t \cdot b$ *holds for the folling pairs* $(a,b) = (45,10)$, $(a,b) = (13,11)$, $(a,b) = (13,12)$. *What pairs* $(a,b)$ *are coprime?*

**Exercise 8** (Towards Prime fields)**.** *Let* $n \in \mathbb{N}$ *be a counting number and* $p$ *a prime number, such that* $n < p$. *What is the greatest common divisor* $gcd(p,n)$*?*

**Exercise 9.** *Find all numbers* $k \in \mathbb{N}$ *with* $0 \leq k \leq 100$ *such that* $gcd(100,k) = 5$.

**Exercise 10.** *Show that* $gcd(n,m) = gcd(n+m,m)$ *for all* $n,m \in \mathbb{N}$.

## 4.2 Modular arithmetic

In mathematics, so called *modular arithmetic* is a system of arithmetic for integers, where numbers "wrap around" when reaching a certain value, much like calculations on a clock wrap around whenever the value exceeds the number 12, 24 or 60, depending on your clock. 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 letter of which has no meaning on a normal clock that shows hours.

The number at which the wrap occures is called the *modulus*. Modular arithmetics generalizes the clock example to arbitrary moduli and studies equations and phenomena that arizes in this new kind of arithmetics. It is of central importance for understanding most modern crypto systems, in large parts because the exponentiation function has an inverse with respect to certain moduli, that is hard to compute. In addition we will see that it provides the foundation of what is called finite fields (See XXX)

Also it will turn out that modular arithmetic appears very different from ordinary integer arithmetic that we are all familiar with, we encouurage the interested reader to work through the example and to discover that, once they accept that this is a new kind of calculations, its actually not that hard.

**Congruency** In what follows, let $n \in \mathbb{N}$ with $n \geq 2$ be a fixed counting number, that we will call the *modulus* of our modular arithmetics system. With such an $n$ given, we can then group integers into classes, by saying that two integers are in the same class, whenever their Euklidean division 4.1 by $n$ will give the same remainder. We then say that two numbers are *congruent* whenever they are in the same class.

**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 Euklidean divide them by $12$. In the picture of an analog $12$-hour clock, starting at $5$ o'clock, when we add $12$ hours we are again at $5$ o'clock, representing the number $17$. On the other hand when we subtract $12$ hours, we are at $5$ o'clock again, representing the number $-7$.*

We can formulize this intuition of what congruency should be into a proper definition utilizing Euklidean division as explained previously 4.1: Let $a, b \in \mathbb{Z}$ be two integers and $n \in \mathbb{N}$ a natural number. Then $a$ and $b$ are said to be **congruent with respect to the modulus** $n$, if and only if the equation

$$a \bmod n = b \bmod n \tag{4.9}$$

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

A *congruency* is then nothing but an equation "up to congruency", which means that the equation only needs to hold if we take the modulus on both sides. In which case we write

$$a \equiv b \pmod{n} \tag{4.10}$$

**Exercise 11.** *Which of the following pairs of numbers are congruent with respect to the modulus $13$: $(5,19)$, $(13,0)$, $(-4,9)$, $(0,0)$.*

**Exercise 12.** *Find all integers $x$, such that the congruency $x \equiv 4 \pmod{6}$ is satisfied.*

**Modular Arithmetics** On particulary nice thing about congruencies is, that we can do calculations (aithmetics), much like we can with integer equations. That is we can add or multiply numbers on both sides. The main difference is probably that the congruency $a \equiv b \pmod{n}$ is only equivalent to the congruency $k \cdot a \equiv k \cdot b \pmod{n}$ for some non zero integer $k \in \mathbb{Z}$, whenever $k$ and the modulus $n$ are coprime. The following list gives a set of useful rules:

Suppose that the congurencies $a_1 \equiv b_1 \pmod{n}$ as well as $a_2 \equiv b_2 \pmod{n}$ are satisfied for integers $a_1, a_2, b_1, b_2 \in \mathbb{Z}$ and that $k \in \mathbb{Z}$ is another integer. Then:

- $a_1 + k \equiv b_1 + k \pmod{n}$ (compatibility with translation)
- $k \cdot a_1 \equiv k \cdot b_1 \pmod{n}$ (compatibility with scaling)
- $a_1 + a_2 \equiv b_1 + b_2 \pmod{n}$ (compatibility with addition)
- $a_1 \cdot a_2 \equiv b_1 \cdot b_2 \pmod{n}$ (compatibility with multiplication)

Other rules like compatibility with subtraction and exponentiation follow from this rules, as for example compatibility with subtraction is compatibility with scaling by $k = -1$ and compatibility with addition.

Note that the previous rules are implications not equivalences, which means that you can not necessarily reverse those rules. The following rules makes this precise:

- If $a_1 + k \equiv b_1 + k \pmod{n}$, then $a_1 \equiv b_1 \pmod{n}$

- If $k \cdot a_1 \equiv k \cdot b_1 \pmod{n}$ and $k$ is coprime with $n$, then $a_1 \equiv b_1 \pmod{n}$
- If $k \cdot a_1 \equiv k \cdot b_1 \pmod{k \cdot n}$, then $a_1 \equiv b_1 \pmod{n}$

Another property of congruencies, not known in the traditional arithmetics of integers is the so called *Fermat's Little Theorem*. In simple words, it says that in modular arithmetics 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:

$$k^p \equiv k \pmod{p}, \tag{4.11}$$

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

$$k^{p-1} \equiv 1 \pmod{p} \tag{4.12}$$

We can invoke sage, to compute examples for both $k$ being coprime and not coprime to $p$:

```
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
```

Now, since this was a lot to digest for a reader who has never encountered modular arithmetics before, lets compute an example that contains most of the stuff we just described:

**Example 7.** *Assume that we choose the modulus 6 and that our task is to solve the following congruency for $x \in \mathbb{Z}$*

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

*As many rules for congruencies are more or less same as for integers, we can proceed in a way similar, as we would if we had an equation to solve. The first thing we notice, is that $7 \cdot (2x+21) + 11 = 14x + 158$, since both sides of a congruency contain ordinary integers. We can therefore rewrite the congruency into the equivalent form*

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

*In a next step we want to shift all encounters of $x$ to left and every other term to the right. So we applay the "compatibility with translation" rules two times. In a first step we choose $k = -x$ and in a second step we choose $k = -158$. Since "compatibility with translation" transforms a congruency into an equivalent form, the solution set will not change and we get*

$$\begin{aligned}
14x + 158 \equiv x - 102 \pmod{6} &\Leftrightarrow \\
14x - x + 158 - 158 \equiv x - x - 102 - 158 \pmod{6} &\Leftrightarrow \\
13x \equiv -260 \pmod{6}
\end{aligned}$$

*If our congruency would just be a normal integer equation, we would divide both sides by* $13$ *to get* $x = -20$ *as our solution. However in case of a congruency we need to make sure that the modulus and the number we want to divide by are coprime first. Only then will we get an equivalent expression. So we need to the greatest common divisor* $gcd(13,6)$ *and since* $13$ *is prime and* $6$ *is not a multiple of* $13$*, we know* $gcd(13,6) = 1$*, so both numbers are indeed coprime. We therefore compute*

$$13x \equiv -260 \quad (\ mod\ 6\ ) \Leftrightarrow x \equiv -20 \quad (\ mod\ 6\ )$$

*Our task is now to find all integers x, such that x is congruent to* $-20$ *with respect to the modulus* $6$*. So we have to find all x such*

$$x\ mod\ 6 = -20\ mod\ 6$$

*Since* $-4\cdot 6+4 = -20$ *we know* $-20\ mod\ 6 = 4$ *and hence we know that* $x = 4$ *is a solution. However* $22$ *is another solution since* $22\ mod\ 6 = 4$ *as well and so is* $-20$*. In fact there are infinite many solutions given by the set*

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

*Putting all this together we have shown that the every x from the set* $\{x = 4+k\cdot 6 \mid k \in \mathbb{Z}\}$ *is a solution to the congruency* $7\cdot(2x+21)+11 \equiv x-102 \quad (\ mod\ 6\ )$*. We double ckeck for, say,* $x = 4$ *as well as* $x = 14+12\cdot 6 = 86$ *using sage:*

```
sage: (ZZ(7)* (ZZ(2)*ZZ(4) + ZZ(21)) + ZZ(11)) % ZZ(6) == (ZZ
   (4) - ZZ(102)) % ZZ(6)
True
sage: (ZZ(7)* (ZZ(2)*ZZ(76) + ZZ(21)) + ZZ(11)) % ZZ(6) == (
   ZZ(76) - ZZ(102)) % ZZ(6)
True
```

The discouraged reader, who at this point thinks that modular aithmetics is to complicated, might consider two thinks: First, computing congruencies in modular arithmetics is not really more complicated then computations in more familiar number systems like fractional numbers. Its just a matter of getting used to it. Second, the theory of prime fields (and more general residue class rings) takes a different view on modular rithmetics with the attempt to simplify thinks. In other words, once we understand prime field arithmetics, thinks become conceptually cleaner and more easy to compute.

**Exercise 13.** *Choose the modulus* $13$ *and find all solutions* $x \in \mathbb{Z}$ *to the following congruency* $5x+4 \equiv 28+2x \quad (\ mod\ 13\ )$

**Exercise 14.** *Choose the modulus* $23$ *and find all solutions* $x \in \mathbb{Z}$ *to the following congruency* $69x \equiv 5 \quad (\ mod\ 23\ )$

**Exercise 15.** *Choose the modulus* $23$ *and find all solutions* $x \in \mathbb{Z}$ *to the following congruency* $69x \equiv 46 \quad (\ mod\ 23\ )$

**The Chinese Remainder Theorem** We have seen in the previous paragraph how to solve congruencies in modular arithmetic. However one question that remains is, how to solve systems of congruencies, whith different moduli? The answer is given by the so called *Chinese raimainder theorem*, which tells us, that for any $k \in \mathbb{N}$ and coprime natural numbers

$n_1, \ldots n_k \in \mathbb{N}$ as well as integers $a_1, \ldots a_k \in \mathbb{Z}$, the so-called *simultaneous congruency*

$$\begin{aligned} x &\equiv a_1 \quad (\bmod n_1) \\ x &\equiv a_2 \quad (\bmod n_2) \\ &\cdots \\ x &\equiv a_k \quad (\bmod n_k) \end{aligned} \tag{4.13}$$

has a solution and all possible solutions of this congruence system are congruent modulo the product $N = n_1 \cdot \ldots \cdot n_k$. In fact, the following algorithm computes the solution set:

**Algorithm 2** Chinese Reminder Theorem

**Require:** $n_0, \ldots, n_{k-1} \in \mathbb{N}$ coprime

**procedure** CONGRUENCY-SYSTEMS-SOLVER$(k, a_{0,\ldots,k-1}, n_{0,\ldots,k-1})$

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

**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 4.13.

This 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 $n_1, \ldots, n_k$ but we don't need that extension in the book.

**Example 8.** *To illustrate how to solve simultaneous congruences using the Chinese remainder theorem, let's look at the following system of congruencies:*

$$\begin{aligned} x &\equiv 4 \quad (\mathit{mod}\ 7) \\ x &\equiv 1 \quad (\mathit{mod}\ 3) \\ x &\equiv 3 \quad (\mathit{mod}\ 5) \\ x &\equiv 0 \quad (\mathit{mod}\ 11) \end{aligned}$$

*Clearly all moduli are coprime and we have* $N = 7 \cdot 3 \cdot 5 \cdot 11 = 1155$, *as well as* $N_1 = 165$, $N_2 = 385$, $N_3 = 231$ *and* $N_4 = 105$. *From this we calculate with the extended Euclidean algorithm*

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

*so we have* $x = 4 \cdot 2 \cdot 165 + 1 \cdot 1 \cdot 385 + 3 \cdot 1 \cdot 231 + 0 \cdot 2 \cdot 105 = 2398$ *as one solution. Because* $2398 \ \mathit{mod}\ 1155 = 88$ *the set of all solutions is* $\{\ldots, -2222, -1067, 88, 1243, 2398, \ldots\}$. *In particular, there are infinitely many different solutions. We can invoke sage's computation of the Chinese Remainder Theorem (CRT) to double check our findings:*

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

As we have seen in various examples before, computing congruencies can be cumersome and solution sets are huge in general. It is therefore advantagous to find some kind of simplification for modular arithmetic.

Fortunately this is possible and relatively straight forward once we consider all integers that have the same remainder with respect to a given modulus $n$ in Euclidean division to be equivalent. Then we can go a step further and identify each set of numbers with equal remainder with that remainder and call it a *remainder class* or *residue class* in modulo $n$ arithmetics.

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 thoses classes.

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

**Example 9** (Arithmetics modulo 6)**.** *Choosing the modulus $n = 6$ we have six equivalence classes of integers which are congruent modulo 6 (which have the same remainder when divided by 6) and when we identify those remainder classes, with the remainder, we get the following identification:*

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

*Now to compute the addition of those equivalence classes, say $2+5$, one chooses arbitrary elements from both sets say 14 and $-1$, adds those numbers in the usual way and then looks in which equivalence class the result will be.*

*So we have $14+(-1) = 13$ and 13 is in the equivalence class (of) 1. Hence we find that $2+5 = 1$ in modular 6 aithmetics, which is a more readable way to write the congruency $2+5 \equiv 1 \quad (\bmod 6)$.*

*Applying the same reasoning to all equivalence classes, addition and multiplication can be transferred to the equivalence classes and the results are summarized in the following addition and multiplication tables for modulus 6 aithmetics:*

| + | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 2 | 3 | 4 | 5 | 0 |
| 2 | 2 | 3 | 4 | 5 | 0 | 1 |
| 3 | 3 | 4 | 5 | 0 | 1 | 2 |
| 4 | 4 | 5 | 0 | 1 | 2 | 3 |
| 5 | 5 | 0 | 1 | 2 | 3 | 4 |

| · | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| 2 | 0 | 2 | 4 | 0 | 2 | 4 |
| 3 | 0 | 3 | 0 | 3 | 0 | 3 |
| 4 | 0 | 4 | 2 | 0 | 4 | 2 |
| 5 | 0 | 5 | 2 | 3 | 2 | 1 |

*This way we have define a new arithmetic system, that contains just 6 numbers and that comes with its own definition of addition and multiplication. Lets symbolize it by $\mathbb{Z}_6$ and call it modular 6 arithmetics.*

*To see why such an identification of a congruency class with its remainder is useful and actually simplifies congruency computations a lot, lets go back to the congruency*

$$7 \cdot (2x+21)+11 \equiv x - 102 \quad (\bmod 6) \tag{4.14}$$

*from example 7 again. As shown in that example, arithmetics of congruencies can deviate from ordinary arithmetics as for example division needs to check for coprimeness of the modulus and the divident and solutions are in general not unique.*

*The point here is, that we can rewrite this congruency into an equation over our new arithmetic type $\mathbb{Z}_6$ by "projecting onto the remainder classes". In particular, since $7 \bmod 6 = 1$, $21 \bmod 6 = 3$, $11 \bmod 6 = 5$ and $102 \bmod 6 = 0$ we have*

$$7 \cdot (2x + 21) + 11 \equiv x - 102 \quad (\bmod 6) \text{ over } \mathbb{Z}$$
$$\Leftrightarrow \; 1 \cdot (2x + 3) + 5 = x \text{ over } \mathbb{Z}_6$$

*and we can use the multiplication and addition tables to solves the equation on the right, like we would solve normal integer equations:*

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

*So we see that, despite the somewhat unfimilar rules of addition and multiplication, solving congruencies this way is very similar to solving normal equations. And indeed the solution set is identical to the solution set of the original congruency, 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 arithmetics type. This is particulary 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
```

**Jargon 1** ($k$-bit modulus)**.** *In cryptographic papers, we can 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. For example, the number 6 has the binary representation 110 and henxe example describes a 3-bit modulus arithmetics system.*

**Exercise 16.** *Let $a, b, k$ be integers, such that $a \equiv b \quad (\bmod n)$ holds. Show $a^k \equiv b^k \quad (\bmod n)$.*

**Exercise 17.** *Let $a, n$ be integers, such that a and n are not coprime. For which $b \in \mathbb{Z}$ does the congruency $a \cdot x \equiv b \quad (\bmod n)$ have a solution x and how does the solution set look in that case?*

**Modular Inverses** As we know integers can be added, subtracted and multiplied, but not divided in general, as for example $3/2$ is not an integer anymore. To see why this is, from a more theoretical perspective, lets 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 neutral 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:

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 by

$$a \cdot a^{-1} = 1 \tag{4.15}$$

So roughly speaking a multiplicative inverse is defined in such a way, that it cancels the original element to give 1, whenever they are multiplied.

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 $a^{-1}$ exist, then we define *division* by $a$ simply as multiplication by the inverse, i.e

$$\frac{b}{a} := b \cdot a^{-1} \tag{4.16}$$

**Example 10.** *Consider the set of rational numbers $\mathbb{Q}$, that is the set of all fractions. Then the neural 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.*

*Then 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 with respect to multiplication the neutral element is the number* 1 *and we notice, that no integer $a \neq 1$ has a multiplicative inverse, since the equation $a \cdot x = 1$ has no integer solutions for $a \neq 1$.*

*The definition of multiplicative inverse works verbtim for addition, too. 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 then always exist and is given by the negative number $-a$, since $a + (-1) = 0$.*

**Example 12.** *Looking at the set $\mathbb{Z}_6$ of residuel classes modulo* 6 *from example XXX, we can use the multiplication table to find multiplicative inverses. To see that we look at the row of the element and then find the entry equal to* 1*. If such an entry exist, the element of that column is the multiplicative inverse. If on the other hand the row has no entry equal to* 1*, we know that the element has no multiplicative inverse.*

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

*Now since* 5 *has a multiplicative inverse modulo* 6*, it makes sense to "divide by* 5 *in $\mathbb{Z}_6$. For example*

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

From the last example we can make the interesting observation, that while 5 has no multiplicative inverse as an integer, it has a multiplicative inverse in modular 6 arithmetics.

So the question remains, to understand, what elements have multiplicative inverses in modular arithmetics. The answer is, that in modular $n$ arithmetics, a residue class $r$ has a multiplicative inverse, if and only if $n$ and $r$ are coprime. Since $ggt(n,r) = 1$ in that case, we know from the extended Euklidean algorithm, that there are numbers $s$ and $t$, such that

$$1 = s \cdot n + t \cdot r \tag{4.17}$$

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

**Example 13** (Multiplicative inverses in $\mathbb{Z}_6$)**.** *In the previous example we have looked up multiplicative inverses in $\mathbb{Z}_6$ from lookup-table XXX. In real world examples, it is of course usually impossible to write down those lookup tables as the modulus is way to large and the sets occasionally contain more elements, then there are atoms in the observable universe.*

*No to see that $2 \in \mathbb{Z}_6$ has no multiplicative inverse in $\mathbb{Z}_6$ without using the lookup table, we immediately observe that $2$ and $6$ are not coprime since their greatest common divisor is $2$. If follows that equation 4.17 has no solutions $s$ and $t$ and hence $2$ has no multiplicative inverse.*

*The same reasoning works for $3$ and $4$, too as both are not coprime with $6$ and the only case that is different is $5$, since $ggt(6,5) = 1$. To compute the multiplicative inverse of $5$ we use the extended Euklidean algorithm and compute*

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

*So we get $s = 1$ as well as $t = -1$ and have $1 = 1 \cdot 6 - 1 \cdot 5$. From this follows that $-1$ mod $6 = 5$ is the multiplicative inverse of $5$ in modular $6$ arithmetics. We can double check using sage:*

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

At this point the attentive reader might notice, that the situation, where the modulus is a prime number is of particular interest, since we know from exercise XXX, that in this cases all remainder classes must have modular inverses, since $ggt(r,n) = 1$ for prime $n$ and $r < n$. In fact Fermat's little theorem then gives a way to compute multiplicative inverses in this situation, since in case of a prime modulus $p$ and $r < p$, we have

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

which tells us, that the multiplicative inverse of a residue class $r$ in modular $p$ arithmetic is precisely $r^{p-2}$.

**Example 14** (Modular 5 arithmetics)**.** *To see the unique properties of mudular arithmetics whenever the modulus is prime numbers, we will parallel our findings from example XXX, but this time for the prime modulus $5$. For $n = 5$ we have five equivalence classes of integers which are congruent modulo $5$. We write*

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

*Addition and multiplication can be transferred to the equivalence classes, in a way exactly similar to example XXX. 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 |

*Calling the set of reminder classes in modular 5 arithmetics with this addition and multiplication $\mathbb{F}_5$ (for reasons we explain in more detail in XXX), we see some subtle but important differences to the situation in $\mathbb{Z}_6$. In particular we see that in the multiplication table every remainer $r \neq 0$ has the entry 1 in its row and therefore has a multiplicative inverse. In addition there are no non zero elements, such that their product is zero.*

*To use Fermat's little theorem in $\mathbb{F}_5$ for computing multiplicative inverses (instead of using the multiplcation table), lets consider $3 \in \mathbb{F}_3$. We know that the multiplicative inverse is then given 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 $3 \cdot 2 = 1$ in $\mathbb{F}_5$.*

*We can invoke sage to do computations in our modular 5 arithmetics type. This is particulary useful to double-check our computations:*

```
sage: Z5 = Integers(5)
sage: Z5(3)**(5-2)
2
sage: Z5(3)**(-1)
2
sage: Z5(3)**(5-2) == Z5(3)**(-1)
True
```

**Example 15.** *To understand one of the principle difference in prime number modular arithmetics vs. other number modular arithmetics, consider the linear equation $a \cdot x + b = 0$ defined over both types $\mathbb{F}_5$ and $\mathbb{Z}_6$. Since in $\mathbb{F}_5$ every non zero element has a multiplicative inverse, we can always solves equations like this, which is not true in $\mathbb{Z}_6$. To see that consider the equation $3x + 3 = 0$. In $\mathbb{F}_5$ we have*

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

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

$$\begin{aligned}
3x+3 &= 0 && \text{\# add 3 and on both sides}\\
3x+3+3 &= 3 && \text{\# addition-table: } 3+3=0\\
3x &= 3 && \text{\# no multiplicative inverse of 3 exists}
\end{aligned}$$

*So in this case, we can not solve the equation for $x$, by dividing by 3. And indeed we use the multiplication table of $\mathbb{Z}_6$, we find that there are three solutions $x \in \{1,3,5\}$, such that $3x+3=0$ holds true for all of them.*

**Exercise 18.** *Consider the modulus $n = 24$. Which of the integers 7, 1, 0, 805, $-4255$ have multiplicative inverses in modular 24 arithmetics? Compute the inverses, in case they exist.*

**Exercise 19.** *Find the set of all solutions to the congruency $17(2x+5)-4 \equiv 2x+4 \quad (\bmod 5)$. Then project the congruency into $\mathbb{F}_5$ and solve the resulting equation in $\mathbb{F}_5$. Compare the results.*

**Exercise 20.** *Find the set of all solutions to the congruency $17(2x+5)-4 \equiv 2x+4 \quad (\bmod 6)$. Then project the congruency into $\mathbb{Z}_6$ and try to solve the resulting equation in $\mathbb{Z}_6$.*

## 4.3 Polynomial Arithmetics

A polynomial is an expression consisting of variables (also called indeterminates) and coefficients, that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponentiation of variables. All coefficients of a polynomial must have the same type, e.g. being integers or fractions etc. To be more precise a *univariate polynomial* is an expression

$$P(x) := \sum_{j=0}^{m} a_j t^j = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x + a_0 \ , \tag{4.18}$$

where $x$ is called the *indeterminate*, each $a_j$ is called a *coefficient*. If $R$ is the type of the coefficients then the set of all **univariate polynomials with coefficients in** $R$ is written as $R[x]$. We often simply *polynimial* instead of univariate polynimial, write $P(x) \in R[x]$ for a polynomial and denote the constant term as $P(0)$.

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

If an univariate polynomial $P(x) = \sum_{j=0}^{m} a_j x^j$ is given, that is not the zero polynomial, we call

$$deg(P) := m \tag{4.19}$$

the *degree* of $P$ and define the degree of the zero polynomial to be $-\infty$, where $-\infty$ (negative infinity) is a symbol with the property that $-\infty + m = -\infty$ for all counting numbers $m \in \mathbb{N}$. In addition we write

$$Lc(P) := a_m \tag{4.20}$$

and call it the *leading coefficient* of the polynomial $P$. We can restrict the set $R[x]$ of *all* polynomials with coefficients in $R$, 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 $R_{\leq m}[x]$ for the set of all poynomials with a degree less or equal to $m$.

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

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

*In particular 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 polynomial of degree* 3 *and leading coefficient* 1*. The following expressions are not integer polynomial*

$$\begin{aligned}
Q_1(x) &= 2x^2 + 4 + 3x^{-2}\\
Q_2(x) &= 0.5x^4 - 2x\\
Q_3(x) &= 1/x
\end{aligned}$$

*We can invoke sage to do computations with polynomials. To do so we have to specify the symbol for the indertimate and the type for the coefficients. Note however that sage defines the degree of the zero polynomial to be $-1$.*

```
sage: Zx = ZZ['x'] # integer polynomials with indeterminate x
sage: Zt.<t> = ZZ[] # integer polynomials with indeterminate 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
2*x^2 - 4*x + 17
sage: p1.degree()
2
sage: p1.leading_coefficient()
2
sage: p2 = Zt(t^23)
sage: p2
t^23
sage: p6 = Zx([0])
sage: p6.degree()
-1
```

**Example 17** (Polynomials over $\mathbb{Z}_6$)**.** *Recall our definition of the residue classes $\mathbb{Z}_6$ and their arithmetics as defined in* **??**. *The set of all polynomials with indeterminant $x$ and coefficients in $\mathbb{Z}_6$ is symbolized as $\mathbb{Z}_6[x]$. Example of polynomials from $\mathbb{Z}_6$ are:*

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

*As in the previous example $P_7$ is a polynomial. However since we are working with coefficients from $\mathbb{Z}_6$ now the expension of $P_7$ is computed differently, as we have to invoke addition and multiplication in $\mathbb{Z}_6$ as defined in XXX. We get:*

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

*We can invoke sage to do computations with polynomials, that have their coefficients in $\mathbb{Z}_6$. To do so we have to specify the symbol for the indertimate and the type for the coefficients:*

```
sage: Z6 = Integers(6)
sage: Z6x = Z6['x']
sage: Z6x
Univariate Polynomial Ring in x over Ring of integers modulo 6
sage: p1 = Z6x([5,-4,2])
sage: p1
2*x^2 + 2*x + 5
sage: p1 = Z6x([17,-4,2])
sage: p1
2*x^2 + 2*x + 5
sage: Z6x(x-2)*Z6x(x+3)*Z6x(x-5) == Z6x(x^3 + 2*x^2 + x)
True
```

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 occurence of the indertimante $x$ in the polynomial expression.

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

$$P(a) = \sum_{j=0}^{m} a_j b^j \tag{4.21}$$

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

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

*Note however that is not possible to evaluate any of those polynomial on values of different type. It is for example strictly speaking wrong to write $P_1(0.5)$, since $0.5$ is not an integer. We can verify our computations using sage:*

```
sage: Zx = ZZ['x']
sage: p1 = Zx([17,-4,2])
sage: p7 = Zx(x-2)*Zx(x+3)*Zx(x-5)
sage: p1(ZZ(2))
17
sage: p7(ZZ(-6)) == ZZ(-264)
True
```

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

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

```
sage: Z6 = Integers(6)
sage: Z6x = Z6['x']
sage: p1 = Z6x([5,-4,2])
sage: p1(Z6(2)) == Z6(5)
True
```

**Exercise 21.** *Compare both expansions of $P_7$ from $\mathbb{Z}[x]$ and from $\mathbb{Z}_6[x]$ in example XXX and example XXX and consider the definition of $\mathbb{Z}_6$ as given in example XXX. 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 $\mathbb{Z}_6$?*

**Polynomial Aithmetics** Polynomials behave like integers in many ways. In particular they can be added, subtracted and multiplied. In addition they have their own notion of Euklidean division. Roughly speaking two polynomials are added by simply adding the coefficients of the same index and they are multiplied by applying the distributive property, that is by multiplying every term of the left factor with every term of the right factor and add the results together.

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

$$\sum_{n=0}^{m_1} a_n x^n + \sum_{n=0}^{m_2} b_n x^n = \sum_{n=0}^{max(\{m_1,m_2\})} (a_n + b_n)x^n \tag{4.22}$$

$$\left(\sum_{n=0}^{m_1} a_n x^n\right)\cdot\left(\sum_{n=0}^{m_2} b_n x^n\right) = \sum_{n=0}^{m_1+m_2}\sum_{i=0}^{n} a_i b_{n-i} x^n \tag{4.23}$$

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

Regarding ower 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 factors, since we defined $-\infty\cdot m = \infty$ for every integer $m \in \mathbb{Z}$. Using sage's definition of degree, this would not hold, as the zero polynomials degree is $-1$ is sage, which would violate this rule.

**Example 20.** *To given an example of how polynomial aithmetics work, 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*

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

*The product of thess two polynomials is computed by multiplication of each term in the first factor with each term in the second factor. We get*

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

```
sage: Zx = ZZ['x']
sage: P = Zx([2,-4,5])
sage: Q = Zx([5,0,-2,1])
sage: P+Q == Zx(x^3 +3*x^2 -4*x +7)
True
sage: P*Q == Zx(5*x^5 -14*x^4 +10*x^3+21*x^2-20*x +10)
True
```

**Example 21.** *Lets consider the polynomials of the previous example but interpreted in modular 6 aithmetics. 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*

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

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

```
sage: Z6x = Integers(6)['x']
sage: P = Z6x([2,-4,5])
sage: Q = Z6x([5,0,-2,1])
sage: P+Q == Z6x(x^3 +3*x^2 +2*x +1)
True
sage: P*Q == Z6x(5*x^5 +4*x^4 +4*x^3+3*x^2+4*x +4)
True
```

**Exercise 22.** *Compare the sum $P+Q$ and the product $P\cdot Q$ from the previous two example XXX and XXX and consider the definition of $\mathbb{Z}_6$ as given in example XXX. How can we derive the computations in $\mathbb{Z}_6[x]$ from the computations in $Z[x]$?*

**Euklidean Division** The ring of polynomials shares a lot of properties with the integers. In particular there is also the concept of Euclidean division and the algorithm of long division defined for polynomials. Recalling from Euklidean division of integers XXX, we know, that given two integers $a$ and $b \neq 0$ there is always another integer $m$ and a counting number $r$ with $r < |b|$, such that $a = m \cdot b + r$ holds.

We can generalize this to polynomials, whenver the leading coefficient of the divident polynomial has a notion of multiplicative inverse. In fact given two polynomials $A$ and $B \neq 0$ from $R[x]$, such that $Lc(B)^{-1}$ exists in $R$, there exist two polynomials $M$ (the quotient) and $R$ (the remainder), such that

$$A = M \cdot B + R \tag{4.24}$$

and $deg(R) < deg(B)$. Similar to integer Euklidean division both $M$ and $R$ are uniquely defined by these relations.

**Notation and Symbols 2.** *Suppose that the polynomials $A, B, M$ and $R$ satisfy equation XX. Then we often write*

$$A\ div\ B := M, \qquad A\ mod\ B := R \tag{4.25}$$

*to describe the quotient and the remainder polynomials of the Euklidean division. We also say, that a polynomial A is divisible by another polynomial B if $A\ mod\ B = 0$ holds. In this case we also write $B|A$ and call B a factor of A.*

Analog to integers, methods to compute Euklidean division for polynomials are called *polynomial division algorithms*. Probably the best known algorithm is the so called *polynomial long division*.

**Algorithm 3** Polynomial Euklidean Algorithm

**Require:** $A, B \in R[x]$ with $B \neq 0$, such that $Lc(B)^{-1}$ exists in $R$
  **procedure** POLY-LONG-DIVISION$(A, B)$
    $M \leftarrow 0$
    $R \leftarrow A$
    $d \leftarrow deg(B)$
    $c \leftarrow Lc(B)$
    **while** $deg(R) \geq d$ **do**
      $S := Lc(R) \cdot c^{-1} \cdot x^{deg(R)-d}$
      $M \leftarrow M + S$
      $R \leftarrow R - S \cdot B$
    **end while**
    **return** $(Q, R)$
  **end procedure**
**Ensure:** $A = M \cdot B + R$

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, lets 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 applay algorithm XXX. Our goal is to find solutions to equation XXX, that is we need to find the quotient polynomial $M \in \mathbb{Z}[x]$ and the reminder* *polynomial $R \in \mathbb{Z}[x]$ such that $x^5 + 2x^3 - 9 = M(x) \cdot (x^2 + 4x - 1) + R$. Using a notation that is mostly used in Commonwealth countries, we compute*

$$
\begin{array}{rrrrrrr}
 & & & X^3 & -4X^2 & +19X & -80 \\ \cline{2-7}
X^2+4X-1 \big) & X^5 & & +2X^3 & & & -9 \\
 & -X^5 & -4X^4 & +X^3 & & & \\ \cline{2-4}
 & & -4X^4 & +3X^3 & & & \\
 & & 4X^4 & +16X^3 & -4X^2 & & \\ \cline{3-5}
 & & & 19X^3 & -4X^2 & & \\
 & & & -19X^3 & -76X^2 & +19X & \\ \cline{4-6}
 & & & & -80X^2 & +19X & -9 \\
 & & & & 80X^2 & +320X & -80 \\ \cline{5-7}
 & & & & & 339X & -89
\end{array}
\tag{4.26}
$$

*We therefore get $M(x) = x^3 - 4x^2 + 19x - 80$ as well as $R(x) = 339x - 89$ and indeed we have $x^5 + 2x^3 - 9 = (x^3 - 4x^2 + 19x - 80) \cdot (x^2 + 4x - 1) + (339x - 89)$, which we can double check invoking sage:*

```
sage: Zx = ZZ['x']
sage: A = Zx([-9,0,0,2,0,1])
sage: B = Zx([-1,4,1])
sage: M = Zx([-80,19,-4,1])
sage: R = Zx([-89,339])
sage: A == M*B + R
True
```

**Example 23.** *In the previous example polynomial division gave a non trivial (non vanishing, i.e non-zero) remainder. Of special interest are divisions that don't give a remainer. Such divisiors are called factors of the divident.*

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

**Exercise 23.** *Consider the polynomial expressions $P(x) := -3x^4 + 4x^3 + 2x^2 + 4$ and $Q(x) = x^2 - 4x + 2$. Compute the Euklidean division of $P$ by $Q$ in the following types*

1. $P, Q \in \mathbb{Z}[x]$
2. $P, Q \in \mathbb{Z}_6[x]$
3. $P, Q \in \mathbb{Z}_5[x]$

*Then consider the result in $\mathbb{Z}[x]$ and in $\mathbb{Z}_6[x]$. How can compute the result in $\mathbb{Z}_6[x]$ from the result in $\mathbb{Z}[x]$?*

**Exercise 24.** *Show that the polynomial $P(x) = 2x^4 - 3x + 4 \in \mathbb{Z}_5[x]$ is a factor of the polynomial $Q(x) = x^7 + 4x^6 + 4x^5 + x^3 + 2x^2 + 2x + 3 \in \mathbb{Z}_5[x]$, thatvis show $P|Q$. What is $Q$ div $P$?*

**Prime Factors** Recall that the fundamental theorem of arithmetics XXX tells us, that every number is the product of prime numbers. Something similar holds for polynomials, too.

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 Euklidean division. Irreducible polynomials are for polynomials what prime numbers are for integers. They are the basic building blocks from which all other polynomials can be constructed. To be more precise, let $P \in R[x]$ be any polynomial. Then there are always irreducible polynomials $F_1, F_2, \ldots, F_k \in R[x]$, such that

$$P = F_1 \cdot F_2 \cdot \ldots \cdot F_k \,. \tag{4.27}$$

This representation is unique, except for permutations in the factors and is called the **prime factorization** 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 then $1 \cdot (x^2 - 3)$, 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.roots()
[]
sage: p.factor()
x^2 - 3
```

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

Finding prime factors of a polynomial is hard. As we have seen in example XXX, points where a polynomial evaluates to zero, i.e points $x_0 \in R$ with $P(x_0) = 0$ are of special interest, since it can be shown the polynomial $F(x) = (x - x_0)$ is always a factor of $P$. The converse however is not necessarily true, because a polynomial can have irreducible prime factors.

Points where a polynomial evaluates to zero are called the **roots** of the polynomial. To be more precise, let $P \in R[x]$ be a polynomial. Then the set of all roots of $P$ is defined as

$$R_0(P) := \{x_0 \in R \mid P(x_0) = 0\} \tag{4.28}$$

Finding the roots of a polynomial is sometimes called solving the polynomial. It is a hard problem and has been the subject of much research throughout history. In fact it is well known that for polynomials of degree 5 or higher there is, in general, no closed expression, from which the roots can be deduced.

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

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

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

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

```
sage: Zx = ZZ['x']
sage: p = Zx(x^7 + 3*x^6 + 3*x^5 + x^4 - x^3 - 3*x^2 - 3*x -
   1)
sage: p.roots()
[(1, 1), (-1, 4)]
sage: p.factor()
(x - 1) * (x + 1)^4 * (x^2 + 1)
```

*We see that P has the root* $1$ *and that the associated prime factor* $(x-1)$ *occures once in P and that it moreover has the root* $-1$*, where the associated prime factor* $(x+1)$ *occures* $4$ *times in P. This gives the prime factorization*

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

**Lange interpolation** One particulary nice property of polynomials is that a polynomial of degree $m$ is completely determined on $m+1$ evaluation points. Seeing this from a different angle, we can (sometimes) uniquely derive a polynomial of degree $m$ from a set

$$S = \{(x_0,y_0),(x_1,y_1),\ldots,(x_m,y_m) \mid x_i \neq x_j \text{ for all indices i and j}\} \tag{4.29}$$

This "few to many" property of polynomials is used in many places, like for example in erasure codes. It is also of importance in snarks and we therefore need to understand a method to actually compute a polynomial from a set of points.

If the coefficients of the polynomial we want to find have a notion of multiplicative inverse, it is always possible to find such a polynomial and one method is called *Lagrange interpolation*. It works as follows: Give a set like 4.29, a polynomial $P$ of degree $m+1$ with $P(x_i) = y_i$ for all pairs $(x_i, y_i)$ from $S$ is given by the following algorithm:

**Algorithm 4** Lagrange Interpolation

**Require:** $R$ must have multiplicative inverses
**Require:** $S = \{(x_0,y_0),(x_1,y_1),\ldots,(x_m,y_m) \mid x_i, y_i \in R, x_i \neq x_j \text{ for all indices i and j}\}$
  **procedure** LAGRANGE-INTERPOLATION($S$)
    **for** $j \in (0\ldots m)$ **do**
      $l_j(x) \leftarrow \Pi_{i=0;i\neq j}^{m} \frac{x-x_i}{x_j-x_i} = \frac{(x-x_0)}{(x_j-x_0)} \cdots \frac{(x-x_{j-1})}{(x_j-x_{j-1})} \frac{(x-x_{j+1})}{(x_j-x_{j+1})} \cdots \frac{(x-x_m)}{(x_j-x_m)}$
    **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 $(x_j, y_j) \in S$

**Example 27.** *Lets consider the set* $S = \{(0,4),(-2,1),(2,3)\}$ *and our task is to compute a polynomial of degree* $2$ *in* $\mathbb{Q}[x]$ *with fractional number coefficients. Since* $\mathbb{Q}$ *has multiplicative inverses, we can use the Lagrange interpolation algorithm from XXX, to compute the polyno-*

*mial. We get*

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

*And indeed evaluation of $P$ on the x-values of $S$ gives the correct points, since $P(0) = 4$, $P(-2) = 1$ and $P(2) = 3$.*

**Example 28.** *To give another example, more relevant to the topics of this book, lets consider the same set $S = \{(0,4),(-2,1),(2,3)\}$ as in the pevious example. But this times the task is to compute a polynomial $P \in \mathbb{F}_5[x]$ from this data. Since we know that multiplicative inverses exist in $\mathbb{Z}_5$, algorithm XXX applies and we can compute a unique polynomial of degree 2 in $\mathbb{Z}_5[x]$ from S. We can use the lookup tables XXX for computation in $\mathbb{Z}_5$ and get*

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

*And indeed evaluation of $P$ on the x-values of $S$ gives the correct points, since $P(0) = 4$, $P(-2) = 1$ and $P(2) = 3$.*

**Exercise 25.** *Consider example XXX and example XXX again. Why is it not possible to applay algorithm XXX if we consider S as a set of integers, nor as a set in $\mathbb{Z}_6$?*

# 5 Algebra

Todo: Def Subgroup, Fundamental theorem of cyclic groups.

We gave an introduction to the basic computational skills needed for a pen & paper approach to SNARKS in the previous chapter. In this chapter we get a bit more abstract and clarify a lot of mathematical terminology and jargon.

When you read papers about cryptography or mathematical papers in general, you will frequently stumble across algebraic terms like *groups*, *fields*,*rings* and similar. To understand what is going on, it is necessary to get at least some understanding of these terms. In this chapter we therefore with a short introduction to those terms.

In a nutshell, algebraic types like groups or fields define sets that are analog to numbers to various extend, in the sense that you can add, subtract, multiply or divide on thoses sets.

We know many example of sets that fall under those categories, like the natural numbers, the integers, the ratinal or the real numbers. they are in some sense already the most fundamental examples.

## 5.1 Groups

Groups are abstractions that capture the essence of mathematical phenomena, like addition and subtraction, multiplication and division, permutations, or symmetries.

To understand groups, remember back in school when we learned about addition and subtraction of integers (Forgetting about integer multiplication for a moment). We learned that we can always add two integers and that the result is guranteed to be an integer again. We also learned how to deal wih brackets, that nothing happens, when we add zero to any integer, that it doesn't matter in which order we add a given set of integers and that for every integer there is always another integer (the negative), such that when we add both together we get zero.

These conditions are the defining properties of a group and mathematicians have recognozed that the exact same set of rules can be found in very different mathematical structures. It therefore makes sense to give a formulation of what a group should be, detached from any concrete example. This allows one to 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 independend list of properties and making them abstract we arrive at the definition of a group:

A **group** $(\mathbb{G},\cdot)$ is a set $\mathbb{G}$, together with a map $\cdot : \mathbb{G}\times\mathbb{G}\to\mathbb{G}$, called the group law, such that the following properties hold:

- (Existence of a neutral element) There is a $e\in\mathbb{G}$ for all $g\in\mathbb{G}$, such that $e\cdot g = g$ as well as $g\cdot e = g$.
- (Existence of an inverse) For every $g\in\mathbb{G}$ there is a $g^{-1}\in\mathbb{G}$, such that $g\cdot g^{-1} = e$ as well as $g^{-1}\cdot g = e$.
- (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.

Rephrasing the abstract definition in more laymans terms, a group is something, where we can do computations that resembles the behaviour of addition of integers. Therefore when the reader reads the term group they are adviced to think of something where can combine some element with another element into a new element in a way that is reversable and where the order of combining many elements doesn't matter.

**Notation and Symbols 3.** *Let $(\mathbb{G}\cdot)$ be a finite group. If there is no risk of ambigously we frquently drop the symbol $\cdot$ and simply write $\mathbb{G}$ as a notation for the group keeping the group law implicit.*

As we will see in what follows, groups are all over the place in cryptography and in SNARKS. In particular we will see in XXX, that the set of points on an elliptic curve define a group, which is the most important example in this book. To give some more familiar examples first:

**Example 29** (Integer Addition and Subtraction)**.** *The set $(\mathbb{Z},+)$ of integers together with integer addition is the archetypical example of a group, where the group law is traditionally written as $+$ (instead of $\cdot$). To compare integer addition against the abstract axioms of a group, we first see that the neutral element $e$ is the number $0$, since $a+0=a$ for all integers $a \in$ and that the inverse of a number is the negative, since $a+(-a)=0$, for all $a \in \mathbb{Z}$. In addition we know that $(a+b)+c=a+(b+c)$, so integers with addition are indeed a group in the abstract sense.*

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

**Commutative Groups** When we look at the general definition of a group we see that it is somewhat different from what we know from integers. For integers we know, that it doesnt matter in which order we add two integers, as for example $4+2$ is the same as $2+4$. However we also know from example XXX, that this is not always the case in groups.

To capture the special case of a group where the order in which the group law is executed doesn't matter, the concept of so called a **commutative group** is introduced. To be more precise a group is called commutative if $g_1 \cdot g_2 = g_2 \cdot g_1$ holds for all $g_1, g_2 \in \mathbb{G}$.

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

**Example 31.** *Consider the group of integers with integer addition again. Since $a+b=b+a$ for all integers, this group is the archetypical example of a commutative group. Since there are infinite many integers, $(\mathbb{Z},+)$ is not a finite group.*

**Example 32.** *Consider our definition of modulo $6$ residue classes $(\mathbb{Z}_6,+)$ as defined in the addition table from example XXX. As we see the residue class $0$ is the neutral element in modulo $6$ arithmetics and the inverse of a residue class $r$ is given by $6-r$, since $r+(6-r)=6$, which is congruent to $0$, since $6 \bmod 6 = 0$. Moreover $(r_1+r_2)+r_3 = r_1+(r_2+r_3)$ is inherited from integer arithmetic.*

*We therefore see that $(\mathbb{Z}_6,+)$ is a group and since addition table XX is symmetric, we see $r_1+r_2=r_2+r_1$ which shows that $(\mathbb{Z}_6,+)$ is commutative.*

The previous example provided us with an important example of commuative groups that are important in this book. Abstracting from this example and considering residue classes $(\mathbb{Z}_n,+)$ for arbitrary moduli $n$, it can be shown that $(\mathbb{Z},+)$ is a commutative group with neutral element

0 and additive inverse $n-r$ for any element $r \in \mathbb{Z}_n$. We call such a group the *reminder class groups* of modulus $n$.

Of particular importance for pairing based cryptography in general and snarks in particular are so called *pairing maps* on commutative groups. To be more precise let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_3$ be three commutative groups. For historical reasons, we write the group law on $\mathbb{G}_1$ and $\mathbb{G}_2$ in additive notation and the group law on $\mathbb{G}_3$ in multiplicative notation. Then a **pairing map** is a function

$$e(\cdot,\cdot) : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3 \tag{5.1}$$

that takes pairs $(g_1, g_2)$ (products) of elements from $\mathbb{G}_1$ and $\mathbb{G}_2$ and maps them somehow to elements from $\mathbb{G}_3$, such that the *bilinearity* property holds: For all $g_1, g_1' \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ we have $e(g_1 + g_1', g_2) = e(g_1, g_2) \cdot e(g_1', g_2)$ and for all $g_1 \in \mathbb{G}_1$ and $g_2, g_2' \in \mathbb{G}_2$ we have $e(g_1, g_2 + g_2') = e(g_1, g_2) \cdot e(g_1, g_2')$.

A pairing map is called *non-degenerated*, if whenever the result of the pairing is the neutral element in $\mathbb{G}_3$, one of the input values must be 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 $g_2 = e_{\mathbb{G}_2}$.

So roughly speaking bilinearity means, that it doesn't matter if we first execute the group law on any side and then apply the bilinear map of if we first applay the bilinear map and then apply the group law. Moreover non-degeneray means that the result of the pairing is zero, only if at least one of the input values is zero.

**Example 33.** *Maybe the most basic example of a non-degenerate pairing is optained, if we take $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_3$ all to be the group of integers with addition $(\mathbb{Z}, +)$. Then the following map*

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

*defines aa non-degenerate pairing. To see that observe, that bilinearity follows from the distriutive law of integers, since for $a, b, c \in \mathbb{Z}$, we have $e(a+b, c) = (a+b) \cdot c = a \cdot c + b \cdot c = e(a,c) + e(b,c)$ and the same reasoning is true for the second argument.*

*To the that $e(\cdot,\cdot)$ is non degenrate, assume that $e(a,b) = 0$. Then $a \cdot b = 0$ and this implies that $a$ or $b$ must be zero.*

**Exercise 26.** *Consider example XXX again and let $\mathbb{F}_5^*$ be the set of all remainder classes from $\mathbb{F}_5$ without the class 0. Then $\mathbb{F}_5^* = \{1, 2, 3, 4\}$. Show that $(\mathbb{F}_5^*, \cdot)$ is a commutative group.*

**Exercise 27.** *Generalizing the previous exercise, consider general moduli $n$ and let $\mathbb{Z}_n^*$ be the set of all remainder classes from $\mathbb{Z}_n$ without the class 0. Then $\mathbb{Z}_n^* = \{1, 2, \ldots, n-1\}$. Give 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 proof the commutative group axioms.*

**Exercise 28.** *Consider the remainder class groups $(\mathbb{Z}_n, +)$ for some modulus $n$. Show that the map*

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

*is bilinear. Why is it not a pairing in general and what condition must be imposed on $n$, such that the map is a pairing?*

**Finite groups** As we have seen in the previous examples, groups can either contain infinite many elements (as the 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. Then we frquently write ord$(\mathbb{G})$ or $|\mathbb{G}|$ for the order of $\mathbb{G}$.*

**Example 34.** *Consider the remainder class groups $(\mathbb{Z}_6,+)$ and $(\mathbb{F}_5,+)$ from example XXX and example XXX and the group $(\mathbb{F}_5^*,\cdot)$ from exercise XX. We can easily see that the order of $(\mathbb{Z}_6,+)$ is $6$, the order of $(\mathbb{F}_5,+)$ is five and the order of $(\mathbb{F}_5^*,\cdot)$ is $4$.*

*To be more general, considering arbitrary moduli n, then we know from Euklidean division, that the order of the remainder class group $(\mathbb{Z}_n,+)$ is n.*

**Exercise 29.** *The RSA crypto system is based on a modulus n that is typically the product of two prime numbers of size 2048-bits. What is (approximately) the order of the rainder class group $(\mathbb{Z}_n,+)$ in this case?*

**Generators** Of special interest, when working with groups are sets of elements that can generate the entire group, by applying the group law repeadly to those elements or their inverses only.

Of course every group $\mathbb{G}$ has trivially a set of generators, when we just consider every element of the group to be in the generator set. So the more interesting question is to find the smallest set of generators. Of particular interest in this regard are groups that have a single generator, that is there exist an element $g \in \mathbb{G}$, such that every other element from $\mathbb{G}$ can be computed by repeated combination of $g$ and its inverse $g^{-1}$ only. Those groups are called **cyclic groups**.

**Example 35.** *The most basic example of a cyclic group are the integers $(\mathbb{Z},+)$ with integer addition. To see that observe that $1$ is a generator of $\mathbb{Z}$, since every integer can be obtained by repeadly add either $1$ or its inverse $-1$ to itself. For example $-4$ is generated by $-1$, since $-4 = -1+(-1)+(-1)+(-1)$.*

**Example 36.** *Consider a modulus n and the remainder class groups $(\mathbb{Z}_n,+)$ from example XXX. These groups are cyclic, with generator $1$, since every other element of that group can be constructed by repeadly 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.*

**Example 37.** *Let $p \in ¶$ be prime number and $(\mathbb{F}_p^*,\cdot)$ the finite group from exercise XXX. Then $(\mathbb{F}_p^*,\cdot)$ is cyclic and every element $g \in \mathbb{F}_q^*$ is a generator.*

**The discrete Logarithm problem** In cryptography in general and in snark development in particular, we often do computations "in the exponent" of a generator. To see what this means, 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 is a map, called the **exponential map** with respect to the generator $g$

$$g^{(\cdot)} : \mathbb{Z}_n \to \mathbb{G}\; x \mapsto g^x \tag{5.2}$$

where $g^x$ means "multiply $g$ $x$-times by itself and $g^0 = e_{\mathbb{G}}$. This map has the remarkable property maps the additive group law of the remainder class group $(\mathbb{Z}_n,+)$ in a one-to-one correspondence to the group law of $\mathbb{G}$.

To see that 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 $\mathbb{G}$ and since $g^{x+y} = g^x \cdot g^y$, the map respects the group laws.

Since 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. This is what is usually meant by saying we do our computations "in the exponent".

**Example 38.** *Consider the multiplicative group $(\mathbb{F}_5^*, \cdot)$ from example XXX. We know that $\mathbb{F}_5^*$ is a cyclic group of order* 4 *and that every element is a generator. Choose* $3 \in \mathbb{F}_5^*$*, we then know that the map*

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

*respects the group law of addition in $\mathbb{Z}_4$ and the group law of multiplication in $\mathbb{F}_5^*$. And indeed doing a computation like*

$$\begin{aligned} 3^{2+3-2} &= 3^3 \\ &= 2 \end{aligned}$$

*in the exponent gives the same result as doing the same computation in $\mathbb{F}*_5$, that is*

$$\begin{aligned} 3^{2+3-2} &= 3^2 \cdot 3^3 \cdot 3^{-2} \\ &= 4 \cdot 2 \cdot (-3)^2 \\ &= 3 \cdot 2^2 \\ &= 3 \cdot 4 \\ &= 2 \end{aligned}$$

Since the exponential map is a one-to-one correspondence, that respects the group law, it can be shown that this map has an inverse

$$log_g(\cdot) : \mathbb{G} \to \mathbb{Z}_n \; x \mapsto log_g(x) \tag{5.3}$$

which is called the **discrete logarithm** map with respect to the base $g$. Discrete logarithms are highly importsnt in cryptography as there are groups, such that the exponential map and its inverse the discrete logarithm, are believed to be one way functions, that is while it is possible to compute the exponential map in polynomial time, computing the discrete log takes (sub)-exponential time.

Now consider a finite cyclic group $\mathbb{G}$ of order $n$ and a generator $g$ of $\mathbb{G}$. The **discrete logarithm problem** is then the task, to find a solution $x \in \mathbb{Z}_n$, to the equation

$$h = g^x \tag{5.4}$$

for some given $h \in \mathbb{G}$. In groups where the expontial map and the discrete logarithm map are believed to be examples of one way functions, it is computationally hard to find solutions to this equation.

**Cofactor Clearing** TODO: (theorem: every factor of order defines a subgroup...)

## 5.2 Commutative Rings

Thinking of integers again, we know, that there are actually two operations addition and multiplication and as we know addition defines a group structure on the set of integers. However multiplication does not define a group structure as we know that integers in general don't have multiplicative inverses.

Combinations like this are captured by the concept of a so called *commutative ring with unit*. To be more precise, a commutative ring with unit $(R, +, \cdot, 1)$ is a set $R$, provided with two maps $+ : R \cdot R \to R$ and $\cdot : R \cdot R \to R$, called *addition* and *multiplication*, such that the following conditions hold:

- $(R,+)$ is a commutative group, where the neutral element is denoted with 0.
- (Commuativity of the multiplication) We have $r_1 \cdot r_2 = r_2 \cdot r_1$ for all $r_1, r_2 \in R$.
- (Existence of a unit) There is an element $1 \in R$, such that $1 \cdot g$ holds 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 laws $g_1 \cdot (g_2 + g_3) = g_1 \cdot g_2 + g_1 \cdot g_3$ holds.

**Example 39** (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.

**Example 40** (Underlying commutative group of a ring)**.** *Every commutative ring with unit $(R,+,\cdot,1)$ gives rise to group, if we just forget about the multiplication*

The following example is more interesting. The motivated reader is encuraged to think through this example, not so much because we need this in what follows, but more so as it helps to detach the reader from familiar styles of computation.

**Example 41.** *Let $S := \{\bullet, \star, \odot, \otimes\}$ be a set that contains four elements and let adiition 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 $(S,\cup,\circ)$ 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 number when it comes to bracketing and computation rules. The only differences are the symbols and the actual way to add and multiply. With this we solve the equation for x in the "usual way"*

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

*So even despite this equation looked really alien on the surface, computation was basically exactly the way "normal" equation like for fractional numbers are done.*

*Note however that in a ring, things can be very different, then most are used to, whenever a multiplicative inverse would be needed to solve an equation in the usual way. For example the equation*

$$\odot \circ x = \otimes$$

*can not be solved for x in the usual way, since there is no multiplicative inverse for $\odot$ in our ring. And in fact looking at the multiplication table we see that no such x exits. On another example the equation*

$$\odot \circ x = \odot$$

*can has not a single solution but two $x \in \{\star, \otimes\}$. Having no or two solutions is certainly not something to expect from types like $\mathbb{Q}$.*

**Example 42.** *Considering polynomials again, we note from their definition, that what we have called the type R of the coefficients, must in fact be a commutative ring with unit, since we need addition, multiplication, commutativity and the existence of a unit for $R[x]$ to have the properties we expect.*

*Now considering R to be a ring, addition and multiplication of polynomials as defined in XXX, actually makes $R[x]$ into a commutative ring with unit, too, where the polynomial $1$ is the multiplicative unit.*

**Example 43.** *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 XXX. It can be shown that $(\mathbb{Z}_n, +, \cdot)$ is a commutative ring with unit $1$.*

Considering the exponential map from XXX again, let $\mathbb{G}$ be a finite cyclic group of order $n$ with generator $g \in \mathbb{G}$. Then the ring structure of $(\mathbb{Z}_n, +, \cdot)$ is mapped onto the group structure of $\mathbb{G}$ in the following way:

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

This of particular interest in cryptographic and snarks, as it allows for the evaluation of polynomials with coefficients in $\mathbb{Z}_n$ to be evaluated "in the exponent". To be more precise let $p \in \mathbb{Z}_n[x]$ be a polyninomial with $p(x) = a_m \cdot x^m + a_{m-1}x^{m-1} + \ldots + a_1 x + a_0$. Then the previously defined exponential laws XXX imply that

$$\begin{aligned} g^{p(x)} &= g^{a_m \cdot x^m + a_{m-1}x^{m-1} + \ldots + a_1 x + a_0} \\ &= \left(g^{x^m}\right)^{a_m} \cdot \left(g^{x^{m-1}}\right)^{a_{m-1}} \cdot \ldots \cdot (g^x)^{a_1} \cdot g^{a_0} \end{aligned}$$

and hence to evaluate $p$ at some point $s$ in the exponent, we can insert $s$ into the right hand side of the last equation and evaluate the product.

As we will see this is a key insight to understand many snark protocols like e.g. Groth16 or XXX.

**Example 44.** *To give an example for the evaluation of a polynomial in the exponent of a finite cyclic group, xonsider the exponential map*

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

*from example XXX. Choosing the polynomial $p(x) = 2x^2 + 3x + 1$ from $\mathbb{Z}_4[x]$, we can evaluate the polynomial at say $x = 2$ in the exponent of 3 in two different ways. On the one hand side we can evaluate p at 2 and then write the result into the expinent, which gives*

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

*and on the other hand we can use the right hand side of equation to evaluate p at 2 in the exponent of 3, which gives:*

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

## 5.3 Fields

In this chapter we started with the definition of a group, which we the expended into the definition of a commutative ring with unit. Those rings generalize the behaviour of integers. In this section we will look at the special case of commutative rings, where every element, other then the neutral element of addition, has a multiplicative inverse. Those structures behave very much like the rational numbers $\mathbb{Q}$, which are in a sense an extension of the ring of integers, that is constructed by just including newly defined multiplicative inverses (the fractions) to the integers.

Now considering the definition of a ring XXX again, we define a **field** $(\mathbb{F}, +, \cdot)$ to be a set $\mathbb{F}$, together with two maps $+ : \mathbb{F} \cdot \mathbb{F} \to \mathbb{F}$ and $\cdot : \mathbb{F} \cdot \mathbb{F} \to \mathbb{F}$, called *addition* and *multiplication*, such that the following conditions holds

- $(\mathbb{F}, +)$ 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) For all $g_1, g_2, g_3 \in \mathbb{F}$ the distributive law $g_1 \cdot (g_2 + g_3) = g_1 \cdot g_2 + g_1 \cdot g_3$ holds.

If a field is iven and the definition of its addition and multiplication is not ambiguous, we will often simple write $\mathbb{F}$ instead of $(\mathbb{F}, +, \cdot)$ to describe it. We moreover write $\mathbb{F}^*$ to describe the multiplicative group of the field, that is the set of elements, except the neutral element of addition, with the multiplication as group law.

The **characteristic** $char(\mathbb{F})$ of a field $\mathbb{F}$ is the smallest natural number $n \geq 1$, for which the $n$ -fold sum of 1 equals zero, i.e. for which $\sum_{i=1}^{n} 1 = 0$. If such a $n > 0$ exists, the field is also called to have a *finite characteristic*. If, on the other hand, every finite sum of 1 is not equal to zero, then the field is defined to have characteristic 0.

**Example 45** (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 counting number $n \in \mathbb{N}$, such that $\sum_{j=0}^{n} 1 = 0$ in the rational numers, the characteristic $char(\mathbb{Q})$ of the field $\mathbb{Q}$ is zero. In sage rational numbers are called like this*

```
sage: QQ
Rational Field
sage: QQ(1/5) # Get an element from the field of rational numbers
1/5
sage: QQ(1/5) / QQ(3) # Division
1/15
```

**Example 46** (Field with two elements)**.** *It can be shown that in any field, the neutral element $0$ of addition must be different from the neutral element $1$ of multiplication, that is we always have $0 \neq 1$ in a field. From this follows that the smallest field must contain at least two elements and 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:*

$$\begin{array}{c|cc} + & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array} \qquad \begin{array}{c|cc} \cdot & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array}$$

*Since $1+1=0$ in the field $\mathbb{F}_2$, we know that the characteristic of $\mathbb{F}_2$ is there, that is we have $char(\mathbb{F}_2)=0$.*

*For reasons we will understand better in XXX, sage defines this field as a so called Galois field with 2 elements. It is called like this:*

```
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
```

**Example 47.** *Both the real numbers $\mathbb{R}$ as well as the complex numbers $\mathbb{C}$ are well known examples of fields.*

**Exercise 30.** *Consider our remainder class ring $(\mathbb{F}_5,+,\cdot)$ and show that it is a field. What is the characteristic of $\mathbb{F}_5$?*

**Prime fields** As we have seen in the variou examples of the previous sections, modular arithmetics behaves in many ways similar to ordinary arithmetics of integers, which is due to the fact that remainder class sets $\mathbb{Z}_n$ are commutative rings with units.

However at the same time we have seen in XXX, that, whenever the modulus is a prime number, every remainder class other then the zero class, has a modular multiplicative inverse.

This is an important observation, since it immediately implies, that in case of 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 finite characteristic $n$

To distinguish this important case from arbitrary reminder class rings, we write $(\mathbb{F}_p, +, \cdot)$ for the field of all remainder classes for a prime number modulus $p \in \mathbb{P}$ 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 many desireable properties. One of them is, that since these sets are finite and a prime field of characteristic $p$ can be represented on a computer in roughly $log_2(p)$ amount of space, no precision problems occure, that are for example unavoidable for computer representations of rational numbers or even the integers, because those sets are infinite.

Since prime fields are special cases of remainder class rings, all computations remain the same. Addition and multiplication can be computed by first doing normal integer addition and multiplication and then take the remainder modulus $p$. Subtraction and division can be computed by addition or multiplication with the additive or the multiplicative inverse, respectively. The additive inverse $-x$ of a field element $x \in \mathbb{F}_p$ is given by $p - x$ and the multiplicative inverse of $x \neq 0$ is given by $x^{p-2}$, or can be computed using the extended Euclidean algorithm.

Note however that these computations might not be the fastest to implement on a computer. They are however useful in this book as they are easy to compute for small prime numbers.

**Example 48.** *The smallest field is the field $\mathbb{F}_2$ of characteristic $2$ as we have seen it in example XXX. It is the prime field of the prime number $2$.*

**Example 49.** *To summarize the basic aspects of computation in prime fields, lets 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 like bracketing (distributivity), summing ect. are identical to the rules we learned in school when we where dealing with rational, real or complex numbers. We get*

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

*In this computation we computed the multiplicative inverse of $3$ using the identity $x^{-1} = x^{p-2}$ in a prime field. This impractical for large prime numbers. Recall that another way of computing the multiplicative inverse is the Extended Euclidean algorithm. To see that again, the task is to compute $x^{-1} \cdot 3 + t \cdot 5 = 1$, but $t$ is actulally irrelevant. We get*

| $k$ | $r_k$ | $x_k^{-1}$ | $t_k = (r_k - s_k \cdot a)\ div\ b$ |
|---|---|---|---|
| 0 | 3 | 1 | $\cdot$ |
| 1 | 5 | 0 | $\cdot$ |
| 2 | 3 | 1 | $\cdot$ |
| 3 | 2 | -1 | $\cdot$ |
| 4 | 1 | 2 | $\cdot$ |

*So the multiplicative inverse of* 3 *in* $\mathbb{Z}_5$ *is* 2 *and indeed if compute* $3 \cdot 2$ *we get* 1 *in* $\mathbb{F}_5$.

**Square Roots** In this part we deal with square numbers also called *quadratic residues* and *square roots* in prime fields. This is of particular importance in our studies on elliptic curves as only square numbers can actually be points on an elliptic curve.

To make the intuition of quadratic risidues and roots precise, let $p \in \mathbb{P}$ be a prime number and $\mathbb{F}_p$ its associate 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 equation

$$x^2 = y \tag{5.5}$$

In this case $y$ is called a **quadratic residue**. On the other hand, if $y$ is given and the quadratic equation has no $x$ solution, we call $y$ as **quadratic non-residue**. For any $y \in \mathbb{F}_p$ we write

$$\sqrt{y} := \{x \in \mathbb{F}_p \mid x^2 = y\} \tag{5.6}$$

for the set of all square roots of $y$ in the prime field $\mathbb{F}_n$. (If $y$ is a quadratic non-residue, then $\sqrt{y} = \emptyset$ and if $y = 0$, then $\sqrt{y} = \{0\}$)

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

It can be shown that in any prime field every non zero element has either no square root or two of them. 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**. This makes sense, as the larger one can always be computed as the modulus minus the smaller one, which is the definition of the negative in prime fields.

**Example 50** (Quadratic (Non)-Residues and roots in $\mathbb{F}_5$)**.** *Let us consider our example prime field* $\mathbb{F}_5$ *again. All square numbers can be found on the main diagonal of the multiplication table XXX. As you can see, in* $\mathbb{Z}_5$ *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, why we will recapitulate 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 so-called *Legendre symbol* of $y$ is defined as follows:

$$\left(\frac{y}{p}\right) := \begin{cases} 1 & \text{if } y \text{ has square roots} \\ -1 & \text{if } y \text{ has no square roots} \\ 0 & \text{if } y = 0 \end{cases} \tag{5.7}$$

**Example 51.** *Look at the quadratic residues and non residues in $\mathbb{F}_5$ from example XXX again, we can deduce the following Legendre symbols, from example XXX.*

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

The legendre symbol gives a criterion to decide wheather or not an element from a prime field has a quadratic root or not. This however is not just of theoretic use, as the following so called *Euler criterion* gives a compact way to actually compute the Legendre symbol. To see that, 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 as

$$\left(\frac{y}{p}\right) = y^{\frac{p-1}{2}}\ . \tag{5.8}$$

**Example 52.** *Look at the quadratic residues and non residues in $\mathbb{F}_5$ from example XXX again, we can compute the following Legendre symbols using the Euler criterium:*

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

**Exercise 31.** *Consider the prime field $\mathbb{F}_{13}$. Find the set of all pairs $(x,y) \in \mathbb{F}_{13} \times \mathbb{F}_{13}$ that satisfy the equation*

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

**Exponentiation** TO APPEAR...

**Extension Fields** We defined prime fields in the previous section. They are the basic building blocks for cryptography in general and snarks in particular.

However as we will see in XX so called *pairing based* snark systems are crucially dependend on group pairings XXX defined over the group of rational points of elliptic curves. For those pairings to be non-trivial the elliptic curve must not only be defined over a prime field but over a so called *extension field* of a given prime field.

We therefore have to understand field extensions. To understand them first observe the field $\mathbb{F}'$ is called an *extension* of a firld $\mathbb{F}$, if $\mathbb{F}$ is a subfield of $\mathbb{F}'$, that is $\mathbb{F}$ is a subset of $\mathbb{F}'$ and restricting the addition and multiplication laws of $\mathbb{F}'$ to the subset $\mathbb{F}$ recovers the appropriate laws of $\mathbb{F}$.

Now it can be shown, that whenever $p \in \mathbb{P}$ is a prime and $m \in \mathbb{N}$ a natural number, then there is a field $\mathbb{F}_{p^m}$ with characteristic $p$ and $p^m$ elements, such that $\mathbb{F}_{p^m}$ is an extension field of the prime field $\mathbb{F}_p$.

Similar to how prime fields $\mathbb{F}_p$ are generated by starting with the ring of integers and then divide by a prime number $p$ and keep the remainder, prime field extensions $\mathbb{F}_{p^m}$ are generated by starting with the ring $\mathbb{F}_p[x]$ of polynomials and then divide them by an irreducible polynomial of degree $m$ and keep the remainder.

To be more precise let $P \in F_p[x]$ be an irreducible polynomial of degree $m$ with coefficients from the given prime field $\mathbb{F}_p$. Then the underlying set $\mathbb{F}_{p^m}$ of the extension field is given by the set of all polynomials with a degree less then $m$:

$$\mathbb{F}_{p^m} := \{a_{m-1}x^{m-1} + a_{k-2}x^{k-2} + \ldots + a_1x + a_0 \mid a_i \in \mathbb{F}_p\} \tag{5.9}$$

which can be shown to be the set of all remainders when dividing any polynomial $Q \in \mathbb{F}_p[x]$ by $P$. So elements of the extension field are polynomials of degree less then $m$. This is analog to how $\mathbb{F}_p$ is the set of all remainders, when dividing integers by $p$.

Addition in then inheritec from $\mathbb{F}_p[x]$, which means that addition on $\mathbb{F}_{p^m}$ is defined as normal addition of polynomials. To be more precise, we have

$$+ : \mathbb{F}_{p^m} \times \mathbb{F}_{p^m} \to \mathbb{F}_{p^m} \ , \ (\sum_{j=0}^{m} a_j x^j, \sum_{j=0}^{m} b_j x^j) \mapsto \sum_{j=0}^{m} (a_j + b_j) x^j \tag{5.10}$$

and we can see that the neutral element is (the polynomial) 0 and that the additive inverse is given by the polynomial with all negative coefficients.

Multiplication in inheritec from $\mathbb{F}_p[x]$, too, but we have to divide the result by our modulus polynomial $P$, whenever the degree of the resulting polynomial is equal or greater to $m$. To be more precise, we have

$$\cdot \ \mathbb{F}_{p^m} \times \mathbb{F}_{p^m} \to \mathbb{F}_{p^m} \ , \ (\sum_{j=0}^{m} a_j x^j, \sum_{j=0}^{m} b_j x^j) \mapsto \left( \sum_{n=0}^{2m} \sum_{i=0}^{n} a_i b_{n-i} x^n \right) \bmod P \tag{5.11}$$

and we can see that the neutral element is (the polynomial) 1. It is however not obvious from this definition how the multiplicative inverse looks.

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

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

It can however be shown, that the fields for different choices of $P$ are isomorphic, which means that there is a one to one identification between all of them and hence from an abstract point of view they are the same thing. From an implementations point of view however some choices are better, because they allow for faster computations.

**Example 53** (The Extension field $\mathbb{F}_{3^2}$)**.** *In (XXX) we have constructed the prime field $\mathbb{F}_3$. In this example we apply the definition (XXX) of a field extension to construct $\mathbb{F}_{3^2}$. We start by choosing an irreducibe polynomial of degree 2 with coefficients in $\mathbb{F}_3$. We try $P(t) = t^2 + 1$. Maybe 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 never zero. WE compute*

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

*This implies, that P is irreducible. The set $\mathbb{F}_{3^2}$ then contains all poynomials of degrees lower then two with coefficients in $\mathbb{F}_3$, which is precisely*

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

*So our extension field contains 9 elements as expected. Addition is defined as addition of polynomials. For example $(t+2)+(2t+2) = (1+2)t+(2+2) = 1$. Doing this computation for all elements give 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 | r+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 |

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

*As it was the case in previozs examples, we can use the table to deduce the negative of any element 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 $\geq 2$, we have to divide it by P and keep the remainder. To see how this works compute the product of $t+2$ and $2t+2$ in $\mathbb{F}_{3^2}$*

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

*So the product of $t+2$ and $2t+2$ in $\mathbb{F}_{3^2}$ is 2. Doing this computation for all elements give 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 |

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

*From the multiplication table we can also see, that the only quadratic residues in $\mathbb{F}_{3^2}$ are 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 $\sqrt{2t}=\{t+1,2t+2\}$.*

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

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

*Computations in extension fields are arguably on the edge of what can reasonbly be done with pen and paper. Fortunately sage provides us with a simple way to do the computations.*

```
sage: Z3 = GF(3) # prime field
sage: Z3t.<t> = Z3[] # polynomials over Z3
sage: P = Z3t(t^2+1)
sage: P.is_irreducible()
True
sage: F3_2.<t> = GF(3^2, name='t', modulus=P)
sage: F3_2
Finite Field in t of size 3^2
sage: F3_2(t+2)*F3_2(2*t+2) == F3_2(2)
True
sage: F3_2(2*t+2)^(-1) # multiplicative inverse
2*t + 1
sage: # verify our solution to (t+1)(x^2 + (2t+2)) = 2
sage: F3_2(t+1)*(F3_2(t)**2 + F3_2(2*t+2)) == F3_2(2)
True
sage: F3_2(t+1)*(F3_2(2*t)**2 + F3_2(2*t+2)) == F3_2(2)
True
```

**Exercise 32.** *Consider the extension field $\mathbb{F}_{3^2}$ from the previous example and find all pairs of elements $(x,y)\in\mathbb{F}_{3^2}$, such that*

$$y^2=x^3+4$$

**Exercise 33.** *Show that the polynomial $P=x^3+x+1$ from $\mathbb{F}_5[x]$ is irreducible. Then consider the extension field $\mathbb{F}_{5^3}$ defined relative to $P$. Compute the multiplicative inverse of $(2t^2+4)\in\mathbb{F}_{5^3}$ using the extended Euklidean algorithm. Then find all $x\in\mathbb{F}_{5^3}$ that solve the equation*

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

## 5.4 Projective Planes

Projective planes are a certain type of geometry defined over some given field, that in a sense extend the concept of the ordinary Euklidean plane by including "points at infinity".

Such an inclusion of infinity points makes them particulary useful in the description of elliptic curves, as the description of such a curve in an ordinary plane needs an additional symbol "the point at infinity" to give the set of points on the curve the structute of a group. Translating the curve into projective geometry, then includes this "point at infinity" more naturally into the set of all points on a projective plane.

To understand the idea for the construction of projective planes, note that in an ordinary Euclidean plane, two lines either intersect in a single point, or are parallel. In the latter case both lines are either the same, that is they intersect in all points, or do not intersect at all. A projective plane can then be thought of as an ordinary plane, but equipped with additional "points at infinity" such that two different lines always intersect in a single point. Parallel lines intersect "at infinity".

To be more precise, let $\mathbb{F}$ be a field, $\mathbb{F}^3 := \mathbb{F} \times \mathbb{F} \times F$ the set of all three tupels over $\mathbb{F}$ and $x \in \mathbb{F}^3$ with $x = (x_1, x_2, x_3)$. Then there is exactly one *line* in $\mathbb{F}^3$ that intersects both $(0,0,0)$ and $x$. This line is given by

$$L_x := \{(k \cdot x_1, k \cdot x_2, k \cdot x_3) \mid k \in \mathbb{F}\} \tag{5.12}$$

A *point* in the **projective plane** over $\mathbb{F}$ is then defined as such a *line* and the projective plane is the set of all such points, that is

$$\mathbb{F}\mathbb{P}^2 := \{L_x \mid x \in \mathbb{F}^3 \text{ with } x \neq (0,0,0)\} \tag{5.13}$$

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

To understand why $L_x$ is called a line, consider the situation, where the underlying field $\mathbb{F}$ are the real numbers $\mathbb{R}$. Then $\mathbb{R}^3$ can be seen as the three dimensional space and $L_{(x,y,z)}$ is then an ordinary line in this 3-dimensional space that intersects zero and the point with coordinates $x$, $y$ and $z$.

The key observation here is, that points in the projective plane, are lines in the 3-dimensional space $\mathbb{F}^3$, also for finite fields, the terms space and line share very little visual similarity with their counterparts over the real numbers.

It follows from this that points $L_x \in \mathbb{F}\mathbb{P}^2$ are not simply described by fixed coordinates $(x_1, x_2, x_3)$, but by *sets of coordinates* rather, where two different coordinates $(x_1, x_2, x_3)$ and $(x'_1, x'_2, x'_3)$, with $x_j \neq 0$ and $x'_j \neq 0$ describe the same point, if and only if there is some field element $k$, such that $(x_1, x_2, x_3) = (k \cdot x'_1, k \cdot x'_2, k \cdot x'_3)$. Descriptions like that are called **projective coordinates**.

**Notation and Symbols 6** (Projective coordinates)**.** *To distingush the (equivalence class) of projective coordinates of a line $L_x$ from the coordinates of $x$, we write $[x_1 : x_2 : x_3]$ for the set of all projective coordinates of $L_x$. Coordinates of the form $[x_1 : x_2 : 1]$ are descriptions of so called **affine points** and coordinates of the form $[x_1 : x_2 : 0]$ are descriptions of so called **points at infinity**. In particular the projective coordinate $[1 : 0 : 0]$ describes the so called **line at infinity**.*

**Example 54.** *Consider the field $\mathbb{F}_3$ from example XXX. As this field only contains, three elements it takes not to much effort to construct its associated projective plane $\mathbb{F}_3\mathbb{P}^2$, as we knwo that it only contain $13$ elemts.*

*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 $(0,0,0)$. Since those lines are parameterized by tupels $(x_1,x_2,x_3)$. We compute:*

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

*Those lines define the* 13 *points in the projective plane $\mathbb{F}_3\mathbb{P}$ as follows*

$$
\begin{aligned}
\mathbb{F}_3\mathbb{P} = \{&L_{(0,0,1)}, L_{(0,1,0)}, L_{(0,1,1)}, L_{(1,0,0)}, L_{(1,0,1)},\\
&L_{(1,0,2)}, L_{(1,1,0)}, L_{(1,1,1)}, L_{(1,1,2)}, L_{(1,2,0)}, L_{(1,2,1)}, L_{(1,2,2)}\}
\end{aligned}
$$

*To understand the abiguity in projective coordinates a bit better, lets consider the point $L_{(1,2,2)}$. As this point in the projective plane is a line in $\mathbb{F}_3^3$, it has the projective coordinates $(1,2,2)$ as well as $(2,1,1)$, since the former coordinate give the latter, when multiplied in $\mathbb{F}_3$ by the factor* 2*. In addition note, that for the same reasons the points $L_{(1,2,2)}$ and $L_{(2,1,1)}$ are the same, since their underlying sets are equal.*

**Exercise 34.** *Construct the so called Fano plane, that is the projective plane over the finite field $\mathbb{F}_2$.*

# 6 Elliptic Curves

Generally speaking, elliptic curves are "curves" defined in geometric planes like the Euklidean or the projective plane over some given field. One of the key features of elliptic curves over finite fields from the point of view of cryptography is their set of points has a group law, such that the resulting group is finite and cyclic and it is believed that the discrete logarithm problem on these groups is hard.

A special class of elliptic curves are so called *pairing friendly curve*, which have a notation of a group pairing as defined in XXX. This pairing has cryptographicall nice prperties. Those curve are useful in the development of SNAKS, since they allow to compute so called R1CS-satisfiability "in the exponent" (THIS HAS TO BE REWRITTEN WITH WAY MORE DETAIL)

In this chapter we introduce epileptic curves as they are used in pairing based approaches to the construction of snarks. The eliptic curves we consider are all defined over prime fields or prime field extensions and the reader should be familiar with the contend of the previous section on those fields.

In its most generality elliptic curves are defined as a smooth projective curve of genus 1 defined over some field $\mathbb{F}$ with a distinguished $\mathbb{F}$-rational point, but this definition is not very useful for the introductary character of this book. We will therefore look at 3 more practical definitions in the following sections, by introducing Weierstraß, Montgommery and Edwards curves. All of them are useful in cryptography and nesessary to understand for the contnuation of the book.

## 6.1 Elliptic Curve Arithmetics

### 6.1.1 Short Weierstraß Curves

In this section we introduce the so called short Weierstraß curves, which are the most general types of curves over finite fields of characteristic greater then 3.

We start with their reprrsention in affine space. This reprresentation has the advantage that affine points are just pairs of numbers which is more convinient to work with for the beginner. However it has the disadvantage that a special "point at infinity" that is not a point on the curve, is necessary to describe the group structure. We introduce the elliptic curve group law and describe elliptic curve scalar multiplication, which is nothing but an instatation of the exponential map from general cyclic groups.

Then we look at the projective representation of short Weierstrass curves. It has the advantage that no special sybol is necessary to represent the point at infinity but comew with the drawback that projective points are classes of numbers, which might be a bit unusual for a beginner.

We finish this section with an explicit equivalence that transforms affine representations into projective once and vice versa.

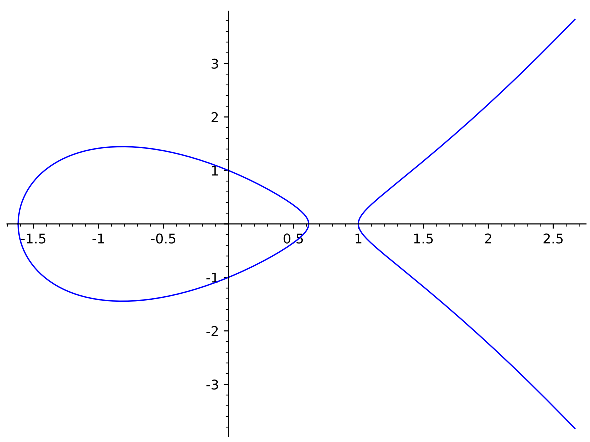
**Affine short Weierstraß form** Probably the least abstract and most straight forward way to introduce elliptic curves for non-mathematicians and beginners is the so called affine representation of a short Weierstraß curve. To see what this is, let $\mathbb{F}$ be a finite field of order $q$ and $a,b \in \mathbb{F}$ two field elements such that $4a^3 + 27b^2 \bmod q \neq 0$. Then a **short Weierstrass elliptic curve** $E(\mathbb{F})$ over $\mathbb{F}$ in its affine representation is the set

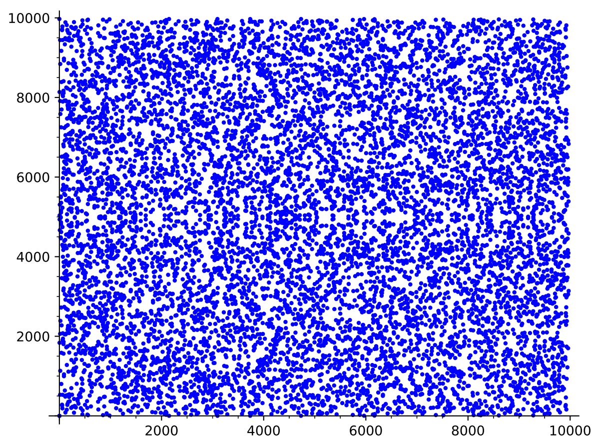
$$E(\mathbb{F}) = \{(x,y) \in \mathbb{F} \times \mathbb{F} \mid y^2 = x^3 + a \cdot x + b\} \bigcup \{\mathcal{O}\} \tag{6.1}$$

of all pairs of field elements $(x,y) \in \mathbb{F} \times \mathbb{F}$, that satisfy the short Weierstrass cubic equation $y^2 = x^3 + a \cdot x + b$, together with a distingushid symbol $\mathcal{O}$, called the **point at infinity**.

**Notation and Symbols 7.** *In the literature, the set $E(\mathbb{F})$, which includs the sybol $\mathcal{O}$ is often called the set of rational points of the elliptic curve, in which case the curve itself is usually written as $E/\mathbb{F}$. However in what follows we will frequently identify an elliptic curve with its set of rational points and therefore use the symbol $E(\mathbb{F})$ instead. This is possible in our case, since we only really care about the group structure of the curve in consideration.*

The term "curve" appears, because in the ordinary 2 dimensional plane $\mathbb{R}^2$, the set of all points $(x,y)$ that satisfy $y^2 = x^3 + a \cdot x + b$ looks like a curve. We should note however, that visualizing elliptic curves over finite fields as "curves" has its limitations and we will therefore not stress the geometric picture too much, but focus on the compuational properties instead. To understand the visual difference, consider the following two elliptic curves:




Both elliptic curves are defined by the same short Weierstraß equation $y^2 = x^3 - 2x + 1$, but the first curve is define in the real affine plane $\mathbb{R}^2$, while the second one is defined in the affine plane $\mathbb{F}_{9973}^2$. Every blue dot represents a pair $(x,y)$ that is solution to $y^2 = x^3 - 2x + 1$ and as we can see the second curve hardly looks like a geometric structure one would naturally call a curve. So the geometric intuitions from $\mathbb{R}^2$ is kind of obfuscated in curves over finite fields.

The identity $6\cdot(4a^3+27b^2) \bmod q \neq 0$ ensures that the curve is non-singular, which basically means that the curve has no cusps or self-intersections.

When dealing with elliptic curves compuations can quickly become cumbersome and tedious. So on the one hand side the reader is adviced to do as many compuations in a pen and paper style as possible. This helps a lot to get a deeper understanding for the details. On the other hand side however, compuations are sometimes simply to large to be done by hand and one might get lost in the details. Fortunately sage is very helpful in dealing with elliptic curves. One we to define elliptic curves and work is them goes like this:

```
sage: F5 = GF(5) # define the base field
sage: a = F5(2) # parameter a
sage: b = F5(4) # parameter b
sage: F5(6)*(F5(4)*F5(2)^3+F5(27)*F5(4)^2) != F5(0)
True
sage: # short Weierstrass curve
sage: E = EllipticCurve(F5,[a,b]) # y^2 == x^3 + ax +b
sage: P = E(0,2) # 2^2 == 0^3 + 2*0 + 4
sage: P.xy() # affine coordinates
(0, 2)
sage: INF = E(0) # point at infinity
sage: try:  # point at infinity has no affine coordinates
....:     INF.xy()
....: except ZeroDivisionError:
....:     pass
sage: P = E.plot() # create a plotted version
```

The following three examples will give a more practical understanding of what an elliptic curve is and how we can compute them. The reader is adviced to read them carefully and ideally to parallel the computation themselfs. We will repeatedly build on these example in this chapter and use the second example at various places in this book.

**Example 55.** *To provide the reader with a small example of an elliptic curve, where all computation can e done in a pen and paper style, consider the prime field $\mathbb{F}_5$ from example (XXX). The reader who had worked through the examples and exercises in the previous section knows this prime field well.*

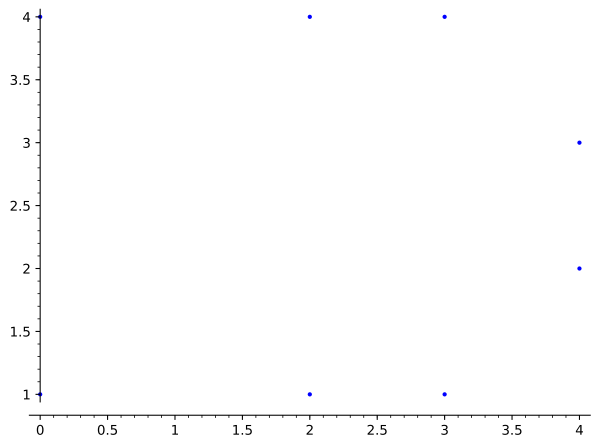
*To define an elliptic curve over $\mathbb{F}_5$, we have to choose to numbers a and b from that field. Assuming we choose $a=1$ and $b=1$ then $4a^3+27b^2 \equiv 1 \pmod 5$ from which follows that the corresponding elliptic curve $E_1(\mathbb{F}_5)$ is given by the set of all pairs $(x,y)$ from $\mathbb{F}_5$ that satisfy the equation $y^2=x^3+x+1$, together with the special symbol $\mathcal{O}$, which represents the "point at infinity".*

*To get a better understand of that curve, obersever that if we choose arbitrarily the pair $(x,y)=(1,1)$, we see that $1^2 \neq 1^3+1+1$ and hence $(1,1)$ is not an element of the curve $E_1(\mathbb{F}_5)$. On the other hand chosing for exmple $(x,y)=(2,1)$ gives $1^2=2^3+2+1$ and hence the pair $(2,1)$ is an element of $E_1(\mathbb{F}_5)$ (Remember that all computations are done in modulo 5 arithmetics).*

*Now since the set $\mathbb{F}_5\times\mathbb{F}_5$ of all pairs $(x,y)$ from $\mathbb{F}_5$ contains only $5\cdot 5=25$ pairs, we can compute the curve, by just inserting every possible pair $(x,y)$ into the short Weierstraß equation $y^2=x^3+x+1$. If the equation holds, the pair is a curve point, if not that means that the point is not on the curve. Combining the result of this computation with the point at infinity gives the* *curve as:*

$$E_1(\mathbb{F}_5) = \{\mathcal{O}, (0,1), (2,1), (3,1), (4,2), (4,3), (0,4), (2,4), (3,4)\}$$

*So our elliptic curve is a set of* $9$ *elements.* $8$ *of which are pairs of numbers and one special symbol* $\mathcal{O}$*. Visualizing* $E1$ *gives:*



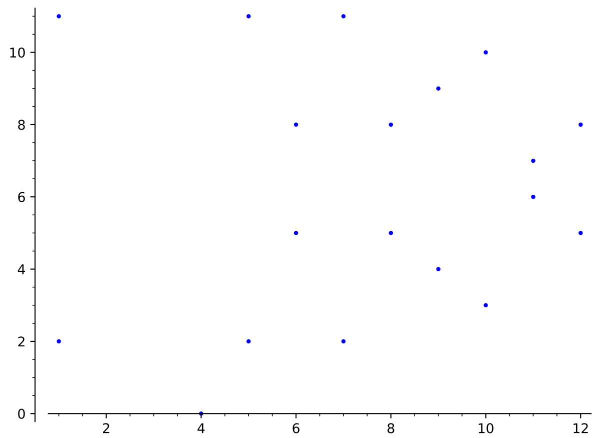

In the development of SNARKS it is sometimes necessary to do elliptic curve cryptograph "in a circuit", which basically means that the elliptic curves needs to be implemented in a certain SNARK-friedly way. We will look at what this means in XXX. To be able to do this efficiently it is desireable to have curves with special properties. The following example is a pen and paper version of such a curve, that parallels the definition of a cryptographically secure curve called *Baby-JubJub* which is extensively used in real world snarks. The interested reader is adviced to read this example carefully as we will use it and build on it in various places throughout the book.

**Example 56** (Pen-JubJub)**.** *Consider the prime field* $\mathbb{F}_{13}$ *from exercise XXX. If we choose* $a = 8$ *and* $b = 8$ *then* $4a^3 + 27b^2 \equiv 6 \pmod{13}$ *and the corresponding elliptic curve is given by all pairs* $(x,y)$ *from* $\mathbb{F}_1 3$ *such that* $y^2 = x^3 + 8x + 8$ *holds. We write PJJ_13 for this curve and call it the Pen-JubJub curve.*

*Now since the set* $\mathbb{F}_{13} \times \mathbb{F}_{13}$ *of all pairs* $(x,y)$ *from* $\mathbb{F}_{13}$ *contains only* $13 \cdot 13 = 169$ *pairs, we can compute the curve, by just inserting every possible pair* $(x,y)$ *into the short Weierstraß equation* $y^2 = x^3 + 8x + 8$*. We get*

$$\begin{aligned} PJJ\_13 = \{&\mathcal{O}, (1,2), (1,11), (4,0), (5,2), (5,11), (6,5), (6,8), (7,2), (7,11), \\ &(8,5), (8,8), (9,4), (9,9), (10,3), (10,10), (11,6), (11,7), (12,5), (12,8)\} \end{aligned}$$

*As we can see the curve consist of* $20$ *points.* $19$ *points from the affine plane and the point at infinity. To get a visual impression of the PJJ_13 curve, we might plot all of its points (except the point at infinity) in the* $\mathbb{F}_{13} \times \mathbb{F}_{13}$ *affine plane. We get:*




*As we will see in what follows this curve is kind of special as it is possible to represent it in two alternitive forms, called the Montgomery and the twisted Edwards form (See xxx and XXX).*

Now that we have seen two pen and paper friendly elliptic curves, lets look at a curve that is used in actual cryptography. Cryptographically secure elliptic curve are not qualitatively defferent from the curves we looked at so far. The only difference is that the prime number modulus of the prime field is much larger. Typical examples use prime numbers, which have binary representations in the size of more then double the size of the desired security level. So if for example a security of 128 bit is desired, a prime moduls of binary size $\geq 256$ is choosen. The following example provides such a curve.

**Example 57** (Bitcoin's Secp256k1 curve)**.** *To give an example of a real world, cryptographically secure curve, lets look at curve Secp256k1, which is famous for being used in the public key cryptography of Bitcoin. The prime field $\mathbb{F}_p$ of Secp256k1 if defined by the prime number*

$$p = 115792089237316195423570985008687907853269984665640564039457584007908834671663$$

*which has a binary representation that need 256 bits. This implies that the $\mathbb{F}_p$ approximately contains $2^{256}$ many elements. So the underlying field is large. To get an image of how large the base field is, consider that the number $2^{256}$ is approximately in the same order of magnitute as the estimated number of atomes in the observeable universe.*

*Curve Secp256k1 is then defined by the parameters $a, b \in \mathbb{F}_p$ with $a = 0$ and $b = 7$. Since $4 \cdot 0^3 + 27 \cdot 7^2 \bmod p = 1323$, those parameters indeed define an elliptic curve given by*

$$Secp256k1 = \{(x,y) \in \mathbb{F}_p \times \mathbb{F}_p \mid y^2 = x^3 + 7\}$$

*Clearly Secp256k1 is a curve, to large to do computations by hand. It is therefore very usefull to understand that sage handles those curve efficiently. For example, while the order of Secp256k1 is large, sage is able to compute it efficiently.*

```
sage: p =
  11579208923731619542357098500868790785326998466564056403945758400790888

sage: p.is_prime()
True
sage: p.nbits()
256
sage: Fp = GF(p)
```

```
sage: Secp256k1 = EllipticCurve(Fp,[0,7])
sage: r = Secp256k1.order() # number of elements
sage: r.is_prime()
True
sage: r.nbits()
256
```

**Exercise 35.** *Look-up the definition of curve BLS12-381, implement it in sage and computes its order.*

**Affine compressed representation** As we have seen in example XXX, cryptographically secure elliptic curves are definec over large prime fields, where elements of those fields typically need more then 255 bits storage on a computer. Since elliptic curve points consists of pairs of those field element, they need double that amount of storage.

To reduce the amount of space needed to represent a curve point note however, that up to a sign the $y$-coordinate of a curve point can be computed from the $x$-coordinate, by simply inserting $x$ into the Weierstraß equation and then computing the roots of the result. This gives two results and it follows that we can represent a curve point in **compressed form** by simply storing the $x$-coordinate together with a single sign bit only, the latter of which deterministically decides which of the two roots to choose. In case that the $y$-coordinate is zero, both sign bit give the same result.

For example one convention could be to always choose the root closer to 0, when the sign bit is 0 and the root closer to the order of $\mathbb{F}$ when the sign bit is 1.

**Example 58** (Pen-JubJub)**.** *To understand the concept of compressed curve points a bit better consider the PJJ_13 curve from example XXX again. Since this curve is defined over the prime field $\mathbb{F}_{13}$ and numbers between $0$ and $13$ need approximately $4$ bits to be represented, each PJJ_13 -point need $8$-bits of storage in uncompressed form, while it would need only $5$ bits in compressed form. To see how this works, recall that in uncompressed form we have*

$$\begin{aligned}PJJ\_13 = \{&\mathcal{O},(1,2),(1,11),(4,0),(5,2),(5,11),(6,5),(6,8),(7,2),(7,11),\\&(8,5),(8,8),(9,4),(9,9),(10,3),(10,10),(11,6),(11,7),(12,5),(12,8)\}\end{aligned}$$

*Using the technique of point compression, we can replace the y-ccordinate in each $(x,y)$ pair by a sign bit, indicating, wheather or not y is closer to $0$ or to $13$. So y values in the range $[0,\ldots,6]$ having sign bit $0$ and y-values in the range $[7,\ldots,12]$ having sign bit $1$. Applying this to the points in PJJ_13 gives the compressed representation:*

$$\begin{aligned}PJJ\_13 = \{&\mathcal{O},(1,0),(1,1),(4,0),(5,0),(5,1),(6,0),(6,1),(7,0),(7,1),\\&(8,0),(8,1),(9,0),(9,1),(10,0),(10,1),(11,0),(11,1),(12,0),(12,1)\}\end{aligned}$$

*Note that the numbers $7,\ldots,12$ are the negatives (additive inverses) of the numbers $1,\ldots,6$ in modular $13$ arithmetics and that $-0 = 0$. Calling the compression bit a "sign bit" therefore makes sense.*

*To recover the uncompressed point of say $(5,1)$, we insert the x-ccordinate $5$ into the Weierstraß equation and get $y^2 = 5^3 + 8\cdot 5 + 8 = 4$. As expected $4$ is a quadratic residue in $\mathbb{F}_{13}$ with roots $\sqrt{4} = \{2,11\}$. Now since the sign bit of the point is $1$, we have to choose the root closer to the modulus $13$ which is $11$. The uncompressed point is therefore $(5,11)$.*

**Affine group law** One of the key properties of an elliptic curve is that it is possible to define a group law on the set of its rational points, such that the point at infinity serves as the neutral element and inverses are reflections on the $x$-axis.

The origin of this law can be understood in a geometric picture and is known as the *chord-and-tangent rule*. In the affine representation of a short Weierstraß curve, the rule can be described in the following way:

- (Point addition) Let $P, Q \in E(\mathbb{F})\backslash\{\mathcal{O}\}$ with $P \neq Q$ be two distinct points on an elliptic curve, that are both not the point at infinity. Then the sum of $P$ and $Q$ is defined as follows: Consider the line $l$ which intersects the curve in $P$ and $Q$. If $l$ intersects the elliptic curve at a third point $R'$, define the sum $R = P + Q$ of $P$ and $Q$ as the reflection of $R'$ at the $x$-axis. If it does not intersect the curve at a third point define the sum to be the point at infinity $\mathcal{O}$. It can be shown, that no such chord-line will intersect the curve in more then three points, so addition is not ambigious.

- (Point doubling) Let $P \in E(\mathbb{F})\backslash\{\mathcal{O}\}$ be a point on an elliptic curve, that is not the point at infinity. Then the sum of $P$ with itself (the doubling) is defined as follows: Consider the line wich is tangent to the elliptic curve at $P$, if this line intersects the elliptic curve at a second point $R'$. The sum $2P = P + P$ is then the reflection of $R'$ at the $x$-axis. If it does not intersect the curve at a third point define the sum to be the point at infinity $\mathcal{O}$. It can be shown, It can be shown, that no such tangent-line will intersect the curve in more then two points, so addition is not ambigious.

- (Point at infinity) We define the point at infinity $\mathcal{O}$ as the neutral ement of addition, that is we define $P + \mathcal{O} = P$ for all points $P \in E(\mathbb{F})$.

It can be shown that the points of an elliptic curve form a commutative group with respect to the tangent and chord rule, such that $\mathcal{O}$ acts the neutral element and the inverse of any element $P \in E(\mathbb{F})$ is the reflection of $P$ on the $x$-axis.

To translate the geometric description into algebraic equations, first observe that for any two given curve points $(x_1, y_1), (x_2, y_2) \in E(\mathbb{F})$, it can be shown that the identity $x_1 = x_2$ implies $y_2 = \pm y_1$, which shows that the following rules are a complete description of the affine addition law.

- (Neutral element) Point at infinity $\mathcal{O}$ is the neutral element.

- (Additive inverse ) The addivive inverse of $\mathcal{O}$ is $\mathcal{O}$ and for any other curve point $(x, y) \in E(\mathbb{F})\backslash\{\mathcal{O}\}$, the additive inverse is given by $(x, -y)$.

- (Addition rule) For any two curve points $P, Q \in E(\mathbb{F})$ addition is defined by one of the following three cases:
  1. (Adding the neutral element) If $Q = \mathcal{O}$, then the sum is defined as $P + Q = P$.
  2. (Adding inverse elements) If $P = (x, y)$ and $Q = (x, -y)$, the sum is defined as $P + Q = \mathcal{O}$.
  3. (Adding non self-inverse equal points) If $P = (x, y)$ and $Q = (x, y)$ with $y \neq 0$, the sum $2P = (x', y')$ is defined by

$$x' = \left(\frac{3x^2+a}{2y}\right)^2 - 2x \quad , \quad y' = \left(\frac{3x^2+a}{2y}\right)^2 (x - x') - y$$

4. (Adding non inverse differen points) If $P=(x_1,y_1)$ and $Q=(x_2,y_2)$ such that $x_1\neq x_2$, the sum $R=P+Q$ with $R=(x_3,y_3)$ is defined by

$$x_3=\left(\tfrac{y_2-y_1}{x_2-x_1}\right)^2-x_1-x_2 \quad , \quad y_3=\left(\tfrac{y_2-y_1}{x_2-x_1}\right)(x_1-x_3)-y_1$$

Note that short Weierstraß curve points $P$ with $P=(x,0)$ are inverse to themselfs, which implies $2P=\mathcal{O}$ in this case.

As we can see, it is very efficient to compute inverses on elliptic curves. However computing the addition of elliptic curve points in the affine representation needs to consider many cases and involves extensive finite field divisions. As we will see in the next paragraph this can be simplified in projective coordinates.

To get some practical impression of how the group law on an elliptic curve is computed, lets look at some actual cases:

**Example 59.** *Consider the elliptic curve $E_1(\mathbb{F}_5)$ from example XXX again. As we have seen, the set of rational points contains* 9 *elements and is given by*

$$E_1(\mathbb{F}_5)=\{\mathcal{O},(0,1),(2,1),(3,1),(4,2),(4,3),(0,4),(2,4),(3,4)\}$$

*We know that this set defines a group, so we can add any two elements from $E_1(\mathbb{F}_5)$ to get a third element.*

*To give an example consider the elements $(0,1)$ and $(4,2)$. Neither of these elements is the neutral element $\mathcal{O}$ and since the x-coordinate of $(0,1)$ is different from the x-coordinate of $(4,2)$, we know that we have to use the chord rule, that is rule number 4 from XXX to compute the sum $(0,1)\oplus(4,2)$. We get*

$$\begin{aligned}
x_3 &= \left(\frac{y_2-y_1}{x_2-x_1}\right)^2-x_1-x_2 && \text{\# insert points}\\
&= \left(\frac{2-1}{4-0}\right)^2-0-4 && \text{\# simplify in } \mathbb{F}_5\\
&= \left(\frac{1}{4}\right)^2+1=4^2+1=1+1=2
\end{aligned}$$

$$\begin{aligned}
y_3 &= \left(\frac{y_2-y_1}{x_2-x_1}\right)(x_1-x_3)-y_1 && \text{\# insert points}\\
&= \left(\frac{2-1}{4-1}\right)(0-2)-1 && \text{\# simplify in } \mathbb{F}_5\\
&= \left(\frac{1}{4}\right)\cdot 3+4=4\cdot 3+4=2+4=1
\end{aligned}$$

*So in our elliptic curve $E_1(\mathbb{F}_5)$ we get $(0,1)\oplus(4,2)=(2,1)$ and indeed the pair $(2,1)$ is an element of $E_1(\mathbb{F}_5)$ as expected. On the other hand we have $(0,1)\oplus(0,4)=\mathcal{O}$, since both points have equal x-coordinates and inverse y-coordinates rendering them as inverse to each other.*

*Adding the point* $(4,2)$ *to itself, we have to use the tangent rule, that is rule 3 from XXX. We get*

$$\begin{aligned}
x' &= \left(\frac{3x^2+a}{2y}\right)^2 - 2x && \textit{\# insert points}\\
&= \left(\frac{3\cdot 4^2+1}{2\cdot 2}\right)^2 - 2\cdot 4 && \textit{\# simplify in } \mathbb{F}_5\\
&= \left(\frac{3\cdot 1+1}{4}\right)^2 + 3\cdot 4 = \left(\frac{4}{4}\right)^2 + 2 = 1+2 = 3
\end{aligned}$$

$$\begin{aligned}
y' &= \left(\frac{3x^2+a}{2y}\right)^2 (x-x') - y && \textit{\# insert points}\\
&= \left(\frac{3\cdot 4^2+1}{2\cdot 2}\right)^2 (4-3) - 2 && \textit{\# simplify in } \mathbb{F}_5\\
&= 1\cdot 1+3 = 4
\end{aligned}$$

*So in our elliptic curve* $E_1(\mathbb{F}_5)$ *we get the doubling* $2\cdot(4,2)$, *that is* $(4,2)\oplus(4,2) = (3,4)$ *and indeed the pair* $(3,4)$ *is an element of* $E_1(\mathbb{F}_5)$ *as expected. The group* $E_1(\mathbb{F}_5)$ *has no self inverse points, since no point has* $0$ *as its y-coordinate. We can invoke sage to double check the computations.*

```
sage: F5 = GF(5)
sage: E1 = EllipticCurve(F5,[1,1])
sage: INF = E1(0) # point at infinity
sage: P1 = E1(0,1)
sage: P2 = E1(4,2)
sage: P3 = E1(0,4)
sage: R1 = E1(2,1)
sage: R2 = E1(3,4)
sage: R1 == P1+P2
True
sage: INF == P1+P3
True
sage: R2 == P2+P2
True
sage: R2 == 2*P2
True
sage: P3 == P3 + INF
True
```

**Example 60** (Pen-JubJub)**.** *Consider the PJJ_13-curve from example XXX again and recall that its group of rational points is given by*

$$\begin{aligned}
PJJ_13 = \{&\mathcal{O},(1,2),(1,11),(4,0),(5,2),(5,11),(6,5),(6,8),(7,2),(7,11),\\
&(8,5),(8,8),(9,4),(9,9),(10,3),(10,10),(11,6),(11,7),(12,5),(12,8)\}
\end{aligned}$$

*In contrast to the group from the previous example, this group contains a self inverse point, given by* $(4,0)$. *To see what this means, observe that we can not add* $(4,0)$ *to itself using the* *tangent rule 3 from XXX, as the y-coordinate is zero. Instead we have to use rule 2, since* $0 = -0$. *We therefore get* $(4,0) \oplus (4,0) = \mathcal{O}$ *in PJJ_13. The point* $(4,0)$ *is therefore inverse to itself, as adding it to itself gives the neutral element.*

```
sage: F13 = GF(13)
sage: MJJ = EllipticCurve(F13,[8,8])
sage: P = MJJ(4,0)
sage: INF = MJJ(0) # Point at infinity
sage: INF == P+P
True
sage: INF == 2*P
True
```

**Exercise 36.** *Consider the PJJ_13-curve from example XXX.*

1. *Compute the inverse of* $(10,10)$, $\mathcal{O}$, $(4,0)$ *and* $(1,2)$.
2. *Compute the expression* $3*(1,11)-(9,9)$.
3. *Solve the equation* $x+2(9,4) = (5,2)$ *for some* $x \in$ *PJJ_13*
4. *Solve the equation* $x \cdot (7,11) = (8,5)$ *for* $x \in \mathbb{Z}$

**Scalar multiplication** As we have seen in the previous section, elliptic curves $E(\mathbb{F})$ have the structure of a commutative group associated to them. It can moreover be shown, that this group is finite and cyclic, whenever the field is finite.

To understand the elliptic curve scalar multiplication, recall from XXX that every finite cyclic group of order $q$ has a genenerator $g$ and an associated exponential map $g^{(\cdot)} : \mathbb{Z}_q \to \mathbb{G}$, where $g^n$ is the $n$-fold product of $g$ with itself.

Now, elliptic curve scalar multiplication is then nothing but the exponential map, written in additive notation. To be more precise let $\mathbb{F}$ be a finite field, $E(\mathbb{F})$ an elliptic curve of order $r$ and $P$ a generator of $E(\mathbb{F})$. Then the **elliptic curve scalar multiplication** with base $P$ is given by

$$[\cdot]P : \mathbb{Z}_r \to E(\mathbb{F}); m \mapsto [m]P$$

where $[0]P = \mathcal{O}$ and $[m]P = P + P + \ldots + P$ is the $m$-fold sum of $P$ with itself. Elliptic curve scalar multiplication is therefore nothing but an instantiation of the general exponential map, when using additive instead of multiplicative notation. This map is a homomorphis of groups, which means that $[n+m]P = [n]P \oplus [m]P$.

As with all finite, cyclic groups the inverse of the exponential map exist and is usually called the *elliptic curve discrete logarithm map*. However elliptic curve are believed to be XXX-groups, which means that we don't know of any efficient way to actuall compute this map.

Scalar multiplication and its inverse, the elliptic curve discrete logarithm, define the elliptic curve discrete logarithm *problem*, which consists of finding solutions $m \in \mathbb{Z}_r$, such that

$$P = [m]Q \tag{6.2}$$

holds. Any solution $m$ is usually called a *discrete logarithm* relation between $P$ and $Q$. If $Q$ is a generator of the curve, then there is a discrete logarithm relation between $Q$ and any other point, since $Q$ generates the group by repeatedly adding $Q$ to itself. So for generator $Q$ and point $P$, we know some discrete logarithm relation exist. However since elliptic curves are believed to be

XXX-groups, finding actual relations $m$ is computationally hard, with runtimes approximately in the size of the order of the group. It pracis we often need the assumption that a discrete logarithm relation exists, but that at the same time noone knows this relation.

One useful property of the exponential map in regard to the examples in this book, is that it can be used to greatly simplify pen and paper computations. As we have seen in example XXX, computing the elliptic curve addition law takes quit a bit of effort, when done without a computer. However when $g$ is a generator of small pen and paper elliptic curve group of order $r$, we can use the exponential map to write the group as

$$\mathbb{G} = \{[1]g \to [2]g \to [3]g \to \cdots \to [r-1]g \to \mathcal{O}\} \tag{6.3}$$

using cofactor clearing, which implies that $[r]g = \mathcal{O}$. "Logarithmic ordering" like this greatly simplifies complicated elliptic curve addition to the much simpler case of modular $r$ addition. So in order to add two curve points $P$ and $Q$, we only have to look up their discrete log relations with the generator, say $P = [n]g$ and $Q = [m]g$ and compute the sum as $P \oplus Q = [n+m]g$. This is, of course, only possible for small groups which we can organize as in XXX.

In the following example we will look at some implications of the fact that elliptic curves are finite cyclic groups. We will apply the fundamental theorem of finite cyclic groups and look how it reflects on the curves in consideration.

**Example 61.** *Consider the elliptic curve group $E_1(\mathbb{F}_5)$ from example XXX. Since it is a finite cyclic group of order 9 and the prime factorization of 9 is $3 \cdot 3$, we can use the fundamental theorem of finite cyclic groups to reason about all its subgroups. In fact since the only prime factor of 9 is 3, we know that $E_1(\mathbb{F}_5)$ has the following subgrous:*

- $\mathbb{G}_1 = E_1(\mathbb{F}_5)$ *is a subgroup of order 9. By definition any group is a subgroup of itself.*
- $\mathbb{G}_2 = \{(2,1),(2,4),\mathcal{O}\}$ *is a subgroup of order 3. This is the subgroup associated to the prime factor 3.*
- $\mathbb{G}_3 = \{\mathcal{O}\}$ *is a subgroup of order 1. This is the trivial subgroup.*

*Moreover since $E_1(\mathbb{F}_5)$ and all its subgroups are cyclic, we know from XXX, that they must have generators. For example the curve point $(2,1)$ is a generator of the order 3-subgroup $\mathbb{G}_2$, since every element of $\mathbb{G}_2$ can be generated, by repeaedly adding $(2,1)$ to itself:*

$$\begin{aligned}[1](2,1) &= (2,1)\\ [2](2,1) &= (2,4)\\ [3](2,1) &= \mathcal{O}\end{aligned}$$

*Since $(2,1)$ is a generator we know from XXX, that it gives rise to an exponential map from the finite field $\mathbb{F}_3$ onto $\mathbb{G}_2$ defined by scalar multiplication*

$$[\cdot](2,1) : \mathbb{F}_3 \to \mathbb{G}_2 \; : \; x \mapsto [x](2,1)$$

*To give an example of a generator that generates the entire group $E_1(\mathbb{F}_5)$ consider the point $(0,1)$. Applying the tangent rule repeadedly we compute with some efford:*

$$\begin{array}{lcllcl}
[0](0,1) & = & \mathcal{O} & [1](0,1) & = & (0,1)\\
[2](0,1) & = & (4,2) & [3](0,1) & = & (2,1)\\
[4](0,1) & = & (3,4) & [5](0,1) & = & (3,1)\\
[6](0,1) & = & (2,4) & [7](0,1) & = & (4,3)\\
[8](0,1) & = & (0,4) & [9](0,1) & = & \mathcal{O}
\end{array}$$

*Again, since $(2,1)$ is a generator we know from XXX, that it gives rise to an exponential map. However since the group order is not a prime number, the exponential maps, does not map a from any field but from the residue class ring $\mathbb{Z}_9$ only:*

$$[\cdot](0,1) : \mathbb{Z}_9 \to \mathbb{G}_1 \; : \; x \mapsto [x](0,1)$$

*Using the generator $(0,1)$ and its associated exponential map, we can write $E(\mathbb{F}_1)$ i logarithmic order with respect to $(0,1)$ as explained in XXX. We get*

$$E_1(\mathbb{F}_5) = \{(0,1) \to (4,2) \to (2,1) \to (3,4) \to (3,1) \to (2,4) \to (4,3) \to (0,4) \to \mathcal{O}\}$$

*indicating that the first element is a generator and the n-th element is the scalar product of n and the generator. To how this logarithmic orders like this simplify the computations in small elliptic curve groups, consider example XXX again. In that example we use the chord and tangent rule to compute $(0,1) \oplus (4,2)$. Now in the logarithmic order of $E_1(\mathbb{F})$ we can compute that sum much easier, since we can directly see that $(0,1) = [1](0,1)$ and $(4,2) = [2](0,1)$. We can then deduce $(0,1) \oplus (4,2) = (2,1)$ immediately, since $[1](0,1) \oplus [2](0,1) = [3](0,1) = (2,1)$.*

*To give another example, we can immediately see that $(3,4) \oplus (4,3) = (4,2)$, without doing any expensive elliptic curve addition, since we know $(3,4) = [4](0,1)$ as well as $(4,3) = [7](0,1)$ from the logarithmic representation of $E_1(\mathbb{F}_5)$ and since $4+7 = 2$ in $\mathbb{Z}_9$, the result must be $[2](0,1) = (4,2)$.*

*Finally we can use $E_1(\mathbb{F}_5)$ as an example to understand the concept of cofactor clearing from XXX. Since the order of $E_1(\mathbb{F}_5)$ is 9 we only have a single factor, which happen to be the cofactor as well. Cofactor clearing then implies that we can map any element from $E_1(\mathbb{F}_5)$ onto its prime factor group $\mathbb{G}_2$ by scalar multiplication with 3. For example taking the element $(3,4)$ which is not in $\mathbb{G}_2$ and multiplying it with 3, we get $[3](3,4) = (2,1)$, which is an element of $\mathbb{G}_2$ as expected.*

In the following example we will look at the subgroups of our pen-jubjub curve, define generators and compute the logarithmic order for pen and paper computations. Then we have anothef look at the principle of cofactor clearing.

**Example 62.** *Consider the pen-jubjub curve PJJ_13 from example XXX again. Since the order of PJJ_13 is 20 and the prime factorization of 20 is $2^2 \cdot 5$, we know that the PJJ_13 contains a "large" prime order subgroup of size 5 and a small prime oder subgroup of size 2.*

*To compute those groups we can apply the technique of cofactor clearing in a try and repeat loop. We start the loop by arbitrarily choose an element $P \in PJJ\_13$. Then we multiply that element with the cofactor of the group, we want to compute. If the result is $\mathcal{O}$, we try a different element and repeat the process until the result is different from the point at infinity.*

*To compute a generator for the small prime order subgroup $(PJJ\_13)_2$, first observe that the cofactor is 10, since $20 = 2 \cdot 10$. We then arbitrarily choose the curve point $(5,11) \in PJJ\_13$ and compute $[10](5,11) = \mathcal{O}$. Since the result is the point at infinity, we have to try another curve point, say $(9,4)$. We get $[10](9,4) = (4,0)$ and we can deduce that $(4,0)$ is a generator of $(PJJ\_13)_2$. Logarithmic order of then gives*

$$(PJJ\_13)_2 = \{(4,0) \to \mathcal{O}\}$$

*as expected, since we know from example XXX that $(4,0)$ is self inverse, with $(4,0) \oplus (4,0) = \mathcal{O}$. Double checking the computations using sage:*

```
sage: F13 = GF(13)                                          228
```

```
sage: PJJ = EllipticCurve(F13,[8,8])
sage: P = PJJ(5,11)
sage: INF = PJJ(0)
sage: 10*P == INF
True
sage: Q = PJJ(9,4)
sage: R = PJJ(4,0)
sage: 10*Q == R
True
```

*We can apply the same reasoning to the "large" prime order subgroup $(PJJ\_13)_5$, which contains 5 elements. To compute a generator for this group, first observe that the associated cofactor is 4, since $20 = 5 \cdot 4$. We choose the curve point $(9,4) \in PJJ\_13$ again and compute $[4](9,4) = (7,11)$ and we can deduce that $(7,11)$ is a generator of $(PJJ\_13)_5$. Using the generator $(7,11)$, we compute the exponential map $[\cdot](7,11) : \mathbb{F}_5 \to PJJ\_13$ and get*

$$
\begin{aligned}
[0](7,11) &= \mathcal{O} \\
[1](7,11) &= (7,11) \\
[2](7,11) &= (8,5) \\
[3](7,11) &= (8,8) \\
[4](7,11) &= (7,2)
\end{aligned}
$$

*We can use this computation to write the large order prime group $(PJJ\_13)_5$ of the pen-jubjub curve in logarithmic order, which we will use quite frequently in what follows. We get:*

$$(PJJ\_13)_5 = \{(7,11) \to (8,5) \to (8,8) \to (7,2) \to \mathcal{O}\}$$

*From this, we can immediately see that for example $(8,8) \oplus (7,2) = (8,5)$, since $3+4 = 2$ in $\mathbb{F}_5$.*

From the previous two examples, the reader might get the impression, that elliptic curve computation can be largely replaced by modular arithmetics. This however is not true in general, but only an arefact of small groups where it is possible to write the entire group in a logarithmic order. The following example gives some understanding, why this is not possible in cryptographically secure groups

**Example 63.** *SEKTP BICOIN. DISCRET LOG HARDNESS PROHIBITS ADDITION IN THE FIELD...*

**Projective short Weierstraß form** As we can see, we had to add a special point at infinity to the definition of a short Weierstrass curve. Recalling from the definition of projective planes we know, that similar points at infinity are handled as ordinary points in projective geometry. It make therefore sense to look at the definition of a short Weierstraß curve in projective geometry.

To see what a short Weierstraß curve in projective coordinates is, let $\mathbb{F}_q$ be a finite field and $a, b \in \mathbb{F}_q$ two field elements such that $4a^3 + 27b^2 \bmod q \neq 0$. Then a **short Weierstrass elliptic curve** $E/\mathbb{F}_q$ over $\mathbb{F}_q$ in its projective representation is the set

$$E/\mathbb{F}_q\mathbb{P}^2 = \{[x:y:z] \in \mathbb{F}_q\mathbb{P}^2 \mid y^2 \cdot z = x^3 + a \cdot x \cdot z^2 + b \cdot z^3\} \tag{6.4}$$

of all points $[x:y:z] \in \mathbb{F}_q\mathbb{P}^2$ from the projective plane, that satisfy the *homogenous* cubic equation $y^2 \cdot z = x^3 + a \cdot x \cdot z^2 + b \cdot z^3$.

As we have seen in XXX, in projective geometry points at infinity are given by equivalence classes $[x:y:0]$. Inserting representatives $(x_1,x_2,0)\in[x:y:0]$ from those classes into the defining homogenous cubic equations gives

$$\begin{aligned} y^2\cdot 0 &= x^3+a\cdot x\cdot 0^2+b\cdot 0^3 & \Leftrightarrow \\ 0 &= x^3 \end{aligned}$$

which shows that the only projective point at infinity that is a point on the curve is the class $[0,1,0]$. This point is the projective representation of $\mathcal{O}$. The projective representation of a short Weierstraß curve therefore has the advantage to not need a special symbol to represent the special "point at infinity" $\mathcal{O}$ from the affine definition.

As we have seen in XXX, one of the key properties of an elliptic curve is that it comes with a definition of a group law on the set of its rational points, described geometrically by the chord and tangent rule. This rule was kind of intuitive, with the exception of the distinguished point at infinity, which appered whenever the chord did not have a third intersection point with the curve.

One of the key features of projective coordinates is now, that in projective space it is guranteed that any chord will always intersect the curve in three points, so the geometric picture simplifies as we don't need to consider external points and associated cases.

Again, it can be shown that the points of an elliptic curve in projective space form a commutative group with respect to the tangent and chord rule, such that the projective point $[0:1:0]$ is the neutral element and the additive inverse of an element $[x:y:z]$ is given by $[x:-y:z]$. The addition law is then usually described by the following algorithm, that minimizes the number of needed additions and multiplications in the base field.

**Example 64.** *Consider the elliptic curve $E_1(F_5)$ from example (XXX). We know that in its affine representation, the set of rational points is given by*

$$E_1(\mathbb{F}_5)=\{\mathcal{O},(0,1),(2,1),(3,1),(4,2),(4,3),(0,4),(2,4),(3,4)\}$$

*which is defined as the set of all pairs $(x,y)\in\mathbb{F}_5\times\mathbb{F}_5$, such that the affine short Weierstrass equation $y^2=x^3+ax+b$ with $a=1$ and $b=1$ is satisfied.*

*To finde the projective representation of a short Weierstrass curve with the same parameter $a=1$ and b, we have to compute the set of projective points $[X:Y:Z]$ from the projective plane $\mathbb{F}_5\mathrm{P}^2$, that satisfy the homogenous cubic equation*

$$Y^2Z=X^3+1\cdot XZ^2+1\cdot Z^3$$

*We know from XXX, that $\mathbb{F}_5\mathrm{P}^2$ contains $5^2+5+1=31$ elements, so we can take the effort and inseret all elements into equation XXX and see if both sides match. To see how this works, consider the projective point $[0:4:1]$. We know from XXX, that this point in the projective plane represents the line*

$$[0:4:1]=\{(0,0,0),(0,4,1),(0,3,2),(0,2,3),(0,1,4)\}$$

*in the three dimensional space $\mathbb{F}^3$. To check whether or not $[0:4:1]$ satisfies XXX, we can insert any representative, that is we can insert any element from XXX. Each element satisfies the equation if and only if any other satisfies the equation. So we insert $(0,4,1)$ and get*

$$1^2\cdot 1=0^3+1\cdot 0\cdot 1^2+1\cdot 1^3$$

**Algorithm 5** Projective Weierstraß Addition Law

**Require:** $[X_1 : Y_1 : Z_1], [X_2 : Y_2 : Z_2] \in E(\mathbb{FP}^2)$
  **procedure** ADD-RULE($[X_1 : Y_1 : Z_1], [X_2 : Y_2 : Z_2]$)
    **if** $[X_1 : Y_1 : Z_1] == [0 : 1 : 0]$ **then**
      $[X_3 : Y_3 : Z_3] \leftarrow [X_2 : Y_2 : Z_2]$
    **else if** $[X_2 : Y_2 : Z_2] == [0 : 1 : 0]$ **then**
      $[X_3 : Y_3 : Z_3] \leftarrow [X_1 : Y_1 : Z_1]$
    **else**
      $U_1 \leftarrow Y_2 \cdot Z_1$
      $U_2 \leftarrow Y_1 \cdot Z_2$
      $V_1 \leftarrow X_2 \cdot Z_1$
      $V_2 \leftarrow X_1 \cdot Z_2$
      **if** $V_1 == V_2$ **then**
        **if** $U_1 \neq U_2$ **then** $[X_3 : Y_3 : Z_3] \leftarrow [0 : 1 : 0]$
        **else**
          **if** $Y_1 == 0$ **then** $[X_3 : Y_3 : Z_3] \leftarrow [0 : 1 : 0]$
          **else**
            $W \leftarrow a \cdot Z_1^2 + 3 \cdot X_1^2$
            $S \leftarrow Y_1 \cdot Z_1$
            $B \leftarrow X_1 \cdot Y_1 \cdot S$
            $H \leftarrow W^2 - 8 \cdot B$
            $X' \leftarrow 2 \cdot H \cdot S$
            $Y' \leftarrow W \cdot (4 \cdot B - H) - 8 \cdot Y_1^2 \cdot S^2$
            $Z' \leftarrow 8 \cdot S^3$
            $[X_3 : Y_3 : Z_3] \leftarrow [X' : Y' : Z']$
          **end if**
        **end if**
      **else**
        $U = U_1 - U_2$
        $V = V_1 - V_2$
        $W = Z_1 \cdot Z_2$
        $A = U^2 \cdot W - V^3 - 2 \cdot V^2 \cdot V_2$
        $X' = V \cdot A$
        $Y' = U \cdot (V^2 \cdot V_2 - A) - V^3 \cdot U_2$
        $Z' = V^3 \cdot W$
        $[X_3 : Y_3 : Z_3] \leftarrow [X' : Y' : Z']$
      **end if**
    **end if**
    **return** $[X_3 : Y_3 : Z_3]$
  **end procedure**
**Ensure:** $[X_3 : Y_3 : Z_3] == [X_1 : Y_1 : Z_1] \oplus [X_2 : Y_2 : Z_2]$

*which tells us that the affine point $[0:4:1]$ is indeed a solution. And as we can see, would just as well insert any other representative. For example inserting $(0,3,2)$ also satisfies XXX, since*

$$3^2 \cdot 2 = 0^3 + 1 \cdot 0 \cdot 2^2 + 1 \cdot 2^3$$

*To find the projective reprresentation of $E_1$, we first observe that the projective line at infinity $[1:0:0]$ is not a curve point on any projective short Weierstraß curve since it can not satisfy XXX for any parameter a and b. So we can exclude it from our consideration.*

*Moreover a point at infinity $[X:Y:0]$ can only satisfy equation XXX for any a and b, if $X=0$, which implies that the only point at infinity relavant for short Weierstrass elliptic curves is $[0:1:0]$, since $[0:k:0]=[0:1:0]$ for all k from the finite field. So we can exclude all points at infinity except the point $[0:1:0]$.*

*So all points that remain are the affine points $[X:Y:1]$. Inserting all of them into XXX we get the set of all projective curve points as*

$$E_1(\mathbb{F}_5\mathrm{P}^2) = \{[0:1:0],[0:1:1],[2:1:1],[3:1:1],[4:2:1],[4:3:1],[0:4:1],[2:4:1],[3:4:1]\}$$

*If we compare this with the affine representation we see that there is a 1:1 correspondence between the points in the affine representation XXX and the affine points in projective geometry and that the point $[0:1:0]$ represents the additional point $\mathcal{O}$ in the projective representation.*

**Example 65** (Polynomial evaluation on secret points)**.** *Since scalar multiplication is assumed to be a one way function, it can be used to encrypt computations. For example it can be used to proof identities of bounded degree polynomials (with some probability), without actually revealing the polynomials. To see what this means, consider the moon-jubjub curve $MJJ(\mathbb{F}_{13})$ from XXX and the set $\mathbb{F}_{13}[x]_{\leq 2}$ of all polynomials with coefficiants in $\mathbb{F}_{13}$ and maximum degree 2.*

*Now assume that there are two parties A and B such that A choose polynomial $P_A$ and B chooses polynomial $P_B$ from $\mathbb{F}_{13}[x]_{\leq 2}$. The task is to check (with some probability) weather or not $P_A$ equals $P_B$ without actually revealing any information about the polynomials.*

*This task can be solved, by evaluating the polynomials at a secret point in the exponent of a (DFHM-PROPERTY) group and then compare the results.*

*So we assume that there is some trusted third party C that chooses a publically known generator of a large prime order subgroup of $MJJ(\mathbb{F}_{13})$, say a secrete point $s \in \mathbb{F}_{13}$, say $s=2$. C then c*

**Exercise 37.** *Compare that affine addition law for short Weierstraß curves with the projective addition rule. Which branch in the projective rule corresponds to which case in the affine law?*

**Coordinate Transformations** As we have seen in example XXX, there was a close relation between the affine and the projective representation of a short Weierstrass curve. This was no accident. In fact from a mathematical point of view projective and affine short Weierstraß curves describe the same thing as there is a one-to-one correspondence (an isomorphism) between both representations for any given parameters $a$ and $b$.

To specify the isomorphism, let $E/\mathbb{F}$ and $E/\mathbb{F}\mathrm{P}^2$ be an affine and a projective short Weierstraß curve defined for the same parameters $a$ and $b$. Then the map

$$\Phi : E/\mathbb{F}_q \to E/\mathbb{F}_q\mathbb{P}^2 \; : \; \begin{array}{lcl} (x,y) & \mapsto & [x:y:1] \\ \mathcal{O} & \mapsto & [0:1:0] \end{array} \tag{6.5}$$

maps points from a the affine representation to points from the projective representation of a short Weierstraß curve, that is if the pair of points $(x,y)$ saisfies the affine equation $y^2 = x^3 + ax + b$, then all homogeneous coordinates $(x_1,x_2,x_3) \in [x : y : 1]$ satisfy the projective equation $x_2^2 \cdot x_3 = x_1^3 + ax_2 \cdot x_3^2 + b \cdot x_3^3$. The inverse is given by the map

$$\Phi^{-1} : E/\mathbb{F}_q\mathbb{P}^2 \to E/\mathbb{F}_q \ : \ [x : y : z] \mapsto \begin{cases} (\frac{x}{z}, \frac{y}{z}) & \text{if } z \neq 0 \\ \mathcal{O} & \text{if } z = 0 \end{cases} \tag{6.6}$$

Note the only projective point $[x : y : z]$ with $z \neq 0$ that satisfys XXX is given by the class $[0 : 1 : 0]$.

One key feature of $\Phi$ and its inverse is, that it respects the group structure, which means that $\Phi((x_1,y_1) \oplus (x_2,y_2))$ is equl to $\Phi(x_1,y_1) \oplus \Phi(x_2,y_2)$. The same holds true for the inverse map $\Phi^{-1}$.

Maps with these properties are caled *group isomorphisms* and from a mathematical point of view the exisrence of $\Phi$ implies, that both definition are equivalent and implementations can choose freely between both representations.

### 6.1.2 Montgomery Curves

History and use of them (otimized scalar multiplication)

**Affine Montgomery Form** To see what a Montgomery curve in affine coordinates is, let $\mathbb{F}_q$ be a finite field of characteristic $char(\mathbb{F}_q) > 2$ and $A, B \in \mathbb{F}_q$ two field elements such that $B \neq 0$ and $A^2 \neq 4$. Then a **Montomery elliptic curve** $M(\mathbb{F}_q)$ over $\mathbb{F}_q$ in its affine representation is the set

$$M(\mathbb{F}_q) = \{(x,y) \in \mathbb{F}_q \times \mathbb{F}_q \mid B \cdot y^2 = x^3 + A \cdot x^2 + x\} \bigcup \{\mathcal{O}\} \tag{6.7}$$

of all pairs of field elements $(x,y) \in \mathbb{F}_q \times \mathbb{F}_q$, that satisfy the Montgomery cubic equation $B \cdot y^2 = x^3 + A \cdot x^2 + x$, together with a distingushid symbol $\mathcal{O}$, called the **point at infinity**.

Despite the fact that Montgomery curves look different then short Weierstrass curve, they are in fact just a special way to describe certain short Weierstrass curves. In fact every curve in affine Montgomery form can be transformed into an elliptic curve in Weierstrass form. To see that assume that a curve in Montgomery form $By^2 = x^3 + Ax^2 + x$ is given. The associated Weierstrass form is then

$$y^2 = x^3 + \frac{3 - A^2}{3B^2} \cdot x + \frac{2A^3 - 9A}{27B^3}$$

On the other hand, an elliptic curve over base field $\mathbb{F}_q$ in Weierstrass form $E : y^2 = x^3 + ax + b$ can be converted to Montgomery form if and only if the following conditions hold:

- The number of points on $E(F_q)$ is divisible by 4
- The polynomial $z^3 + az + b \in \mathbb{F}_q[z]$ has at least one root $\alpha \in \mathbb{F}_q$
- $3\alpha^2 + a$ is a quadratic residue in $\mathbb{F}_q$.

When these conditions are satisfied, then for $s = (\sqrt{3\alpha^2 + a})^{-1}$ the equivalent Montgomery curve is given by

$$sy^2 = x^3 + (3\alpha s)x^2 + x$$

So we see that Montgomery curves a special cases of short Weierstrass curves. As such they have a group structure defined on the set of their points, which can also be derived from a chord and tangent rule. In accordance with short Weierstrass curves, it can be shown that the identity $x_1 = x_2$ implies $y_2 = \pm y_1$, which shows that the following rules are a complete description of the affine addition law.

- (Neutral element) Point at infinity $\mathcal{O}$ is the neutral element.
- (Additive inverse ) The addivive inverse of $\mathcal{O}$ is $\mathcal{O}$ and for any other curve point $(x,y) \in M(\mathbb{F}_q)\backslash\{\mathcal{O}\}$, the additive inverse is given by $(x,-y)$.
- (Addition rule) For any two curve points $P,Q \in M(\mathbb{F}_q)$ addition is defined by one of the following cases:
  1. (Adding the neutral element) If $Q = \mathcal{O}$, then the sum is defined as $P+Q = P$.
  2. (Adding inverse elements) If $P = (x,y)$ and $Q = (x,-y)$, the sum is defined as $P+Q = \mathcal{O}$.
  3. (Adding non self-inverse equal points) If $P = (x,y)$ and $Q = (x,y)$ with $y \neq 0$, the sum $2P = (x',y')$ is defined by
  $$x' = (\tfrac{3x_1^2+2Ax_1+1}{2By_1})^2 \cdot B - (x_1+x_2) - A \quad , \quad y' = \tfrac{3x_1^2+2Ax_1+1}{2By_1}(x_1 - x') - y_1$$
  4. (Adding non inverse differen points) If $P = (x_1,y_1)$ and $Q = (x_2,y_2)$ such that $x_1 \neq x_2$, the sum $R = P+Q$ with $R = (x_3,y_3)$ is defined by
  $$x' = (\tfrac{y_2-y_1}{x_2-x_1})^2 B - (x_1+x_2) - A \quad , \quad y' = \tfrac{y_2-y_1}{x_2-x_1}(x_1 - x') - y_1$$

### 6.1.3 Twisted Edwards Curves

On of our main goals is do implement example snarks defined over the scalar field of our BLS6 curve. In particular we want to do elliptic curve cryptography inside of circuits. To do so we need an elliptic curve that we can implement as a circuit over the scalar field of BLS6.

As we have seen in XXX Weierstrass curve have somewhat complicated addition and doubling laws as many cases have to be distinguished. Those cases translate to branches in computer programs, which are costly when implemented in circuits/r1cs. It is therefore advantageous to look for curves with a more simple addition/doubling rule.

Edwards curves are particular useful here as they have a compact addition law that works for all point. Implementing that rule therefore needs no branching.

**Twisted Edwards Form** In this section we describe curves, that are defined over finite fields of characteristics $\neq 2$. An affine **twisted Edwards curve** over a finite field $\mathbb{F}$ is a curve, defined by the equation

$$E/\mathbb{F} : a \cdot x^2 + y^2 = 1 + d \cdot x^2 y^2$$

where $x,y \in \mathbb{F}$ and $d \in \mathbb{F}\backslash\{0,1\}$ and $d \in \mathbb{F}\backslash\{1\}$ with $a \neq d$ Assuming that $a$ has a root in $\mathbb{F}$ and $d$ has no root in $\mathbb{F}$. Such a curve is called an Edwards curve (non twisted), if $a = 1$.

The most remarkable fact about twisted Edwards curves is their simple addition law. the sum of any two points $(x_1,y_1)$, $(x_2,y_2)$ on such an Edwards curve $E$ is given by

$$(x_1,y_1) + (x_2,y_2) = \left( \frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2} \right)$$

The point $(0,1)$ is the neutral element of the addition law. The inverse of a point $(x_1,y_1)$ on $E$ is $(-x_1,y1)$. The addition law is very simple to implement and it can also be used for point doubling. It also works for the neutral element, for inverses..

As Edwards curves have such simple non branching addition laws that work for all points, including the neutral element and inverses, they are a interesting for snarks as they can be implemented with only a few constraints in circuits.

(0,1)

The twisted Edwards form is just another wa to describe Montgomery elliptic curve. In fact every curve in twisted Edwards form can be transformed into an elliptic curve in Montgomery form and vice versa. To see that assume that a curve in twisted Edwards form $a\cdot x^2+y^2=1+d\cdot x^2y^2$ is given. The associated Montgomery form is then

$$\frac{4}{a-d}y^2 = x^3+\frac{2(a+d)}{a-d}\cdot x^2+x$$

On the other hand a Montgomery curve $By^2=x^3+Ax^2+x$ (WITH STUFF) can be transformed into a twisted Edwards curve

$$(\frac{A+2}{B})x^2+y^2 = 1+(\frac{A-2}{B})x^2y^2$$

### 6.1.4 STUFF

**Keys** - Keys on elliptic curves - compressed keys

**Example 66.** *Lets consider the curve $E_1/\mathbb{F}_5$ from example XXX again. We have*

$$E_1/\mathbb{F}_5 = \{\mathcal{O},(0,0),(2,0),(3,0)\}$$

*So as always $\mathcal{O}$ is the neutral element. Since all elements have $0$ as their y-coordinate, it follows that all of them are self inverse, that is $-P=P$. To add, say $(2,0)$ and $(3,0)$ we use the addition rule, since their x-coordinates differ, we get*

$$(0,0)+(2,0)=(3,0)$$
$$(0,0)+(3,0)=(2,0)$$
$$(2,0)+(3,0)=(0,0)$$

*As we can see $(0,2)$ is not a generator of the group since $[1](0,2)=(0,2)$, $[2](0,2)=(0,2)+(0,2)=\mathcal{O}$. HMM I GUESS THERE IS SOMETHING WRONG HERE. THREE TWO ELEMENT SUBGROUPS SHOULDN'T EXIST??*

**Example 67.** *Consider our prime field $F_5$ from (XXX). If we choose $a=1$ and $b=1$ then $4a^3+27b^2=1\neq 0$ and the corresponding elliptic curve $E/\mathbb{F}_3$ is given by all pairs $(x,y)$ from $\mathbb{F}_5$ such that $y^2=x^3+x+1$. We can find this set simply by trying all 25 combinations of pairs. We get*

$$E_2/\mathbb{F}_5 = \{\mathcal{O},(0,1),(2,1),(3,1),(4,2),(4,3),(0,4),(2,4),(3,4)\}$$

*So our elliptic curve contains 9 elements and the trace t is therefore $-3$.*

**Cryptographically secure elliptic curves** Not all elliptic curves satisfy the requirements from applied cryptography .... Here is a list of properties a curve should satisfy:

1. TECHNOBOB

### 6.1.5 twists

## 6.2 Pairings

In this section, we discuss *pairings*, which form the basis of several zk-SNARKs and other zero knowledge proof schemes. The SNARKs derived from pairings have the advantage of constant-sized proof sizes, which is crucial to blockchains.

We start out by defining pairings and discussing a simple application which bears some resemblance to the more advanced SNARKs. We then introduce the pairings arising from elliptic curves and describe Miller's algorithm which makes these pairings practical rather than just theoretically interesting.

### 6.2.1 Hashing to Curve

In various cryptographic primitives the ability to generate elliptic curve points from binary strings that have properties similar to those of cryptographically secure hash functions is desireable. We call this hash-to-curve.

Of particular interest are constructions, that not only hash to curve points, but where the images have spacial properties, like being in the pairing groupd $\mathbb{G}_1$ or $\mathbb{G}_2$.

Thinking about hashing to curves and hashing to certain subgroups in particular maybe the most obvious thing that comes to mind, is to simply hash to the scalar field of the eliptic curve and then use a generator of the subgroup taget and multiply the generator with the hash value in that scalar field. The result is a valid curve point that is guranteed to be in the desired subgroup.

To be more precise given an elliptic curve $E/\mathbb{F}$, a subgroup $\mathbb{G}$ of $E$ of order $r$, a generator $g$ of $\mathbb{G}$, a bit-string $s$ and a hash function $H : \{0,1\}^* \to \mathbb{F}_r$, then $[H(s)]g$ is an element of $\mathbb{G}$ and hence $H' : \{0,1\}^* \to \mathbb{G}$ with $H'(s) := [H(s)]g$ is a hash function.

This naive approach however is almost never adequate in cryptographic applications as a discrete log relation is always know between any two given hash value $H'(s)$ and $H'(t)$. In fact for $H(s) \neq 0$, we can define $x = H(t)/H(s)$ and then have $[x]H'(s) = H'(t)$.

So we need a different approach:

**Try and increment hash functions** One of the most straight forward ways to hash a bitstring onto an eliptic curve point, in a secure way, is to use a cryptographic hash function together with one of the methods we described in XXX to hash to the base field of the curve. Ideally the hash function generates an image that is at least one bit longer then the bit representation of the base field modulus.

The image in the base field can then be interpreted as the $x$-coordinate of the curve point and the two possible $y$-coordinates are then derived from the curve equation, while one of the bits that exceeded the modulus determines which of the two $y$-coordinates to choose.

Such an approach would be easy to implement and deterministic and it will conserve the cryptographic properties of the original hash function. However not all $x$-ccordinates genrated in such a way, will result in quadratic residues, when inserted into the defining equation. It follows that not all field elements give rise to actual curve points. In fact on a prime field, only half of the field elements are quadratic residues and hence assuming an even distribution of the hash values in the field, this method would fail to generate a curve point in about half of the attempts.

One way to account for this problem is the so called *try and increment* method. Its basic assumption is, that hashing different values, the result will eventually lead to a valid curve point.

Therefore instead of simply hashing a string $s$ to the field the concatenation of $s$ with additional bytes is hashed to the field instead. The bytes are initially zero and interpreted as an unsigned integer. If the first *try* of hashing to the field does not result in a valid curve point, the counter is *incremented* and hashed again. This is repeated until a valid curve point is found eventually.

This method has the advantegous that is relatively easy to implement in code and that it preserves the cryptographic properties of the original hash function. However it is not guranteed to find a valid curve point, as there is a chance that all possible values in the choosen size of the counter bytes fail to generate a quadratic residue. Fortunately it is possible to make the probability for this arbitrarily small by choosing large enough counters and relying on the (approximate) uniformity of the hash-to-field function.

One might think that another disadvantage of this method in the context of snarks is that it can not be implemented as a circuit effectively. This however is not fully true, as a circuit/r1cs only needs to enforce the correctness of the computation. Hence for the circuit it is enough to check the hash of the string and the correct counter. It does not need to find that counter.

So to be more formal, we can define a try and increment hash-to-curve like this

DEF

Conidering certain subgroups of the elliptic curve, the usefulness of this methods depends highly on the actual situation. For example if a hash to the $n$-torsion subgroup $\mathbb{G}_1$ is desired, there are two possibilities:

First generic try and increment can be used, followed by a cofactor clearing step. This way every hash on the curve is considered valid, but then projected to $\mathbb{G}_1$ afterwards

Second, the try step not only checks if the curve point actually exists, but also if it is a point in $\mathbb{G}_1$. If its not then the increment step is executed until a valid point is found. This is possible in most applications, since $\mathbb{G}_1$ is usually the by fare largest subgroup in $E$ and the probability to find a point in it is large.

The situation for $\mathbb{G}_2$ as defined in XXX is different and the try and increment method usually fails to find hash values in $\mathbb{G}_2$. For once $\mathbb{G}_2$ is defined in $E$ over an extension field, not the prime field itself, so hashing to $\mathbb{F}$ must be extended into a hash to the extension field. This is possible, but not desireable ventually, because even if we find valid curve points in the curve, $\mathbb{G}_2$. We therefore need different approaches for hashing into $\mathbb{G}_2$

### Pederson Hashes

**Definition 6.2.1.1** (Pedersen's Hash function)**.** *Let $\mathbb{G}$ be a cyclic group of order $r$ and $\{g_1, \ldots, g_k\} \subset \mathbb{G}$ a uniform randomly generated set of generators of $\mathbb{G}$. Then **Pedersen's hash function***

$$H : (\mathbb{F}_r)^k \to \mathbb{G}$$

*is defined for any message $M = (M_1, \ldots, M_k) \in \mathbb{F}_r \times \cdots \times \mathbb{F}_r$, by $H(M) = \Pi_{j=1}^{k}[M_j]g_j$.*

SIMPLE WORD DESCRIPTION

**Remark 1.** *Of course Pedersen's hash function can be combined with a hash-to-field function as described in XXX to give a function that maps ordinary bit strings to group points. However for this function to be secure, it is necessary to proof that there is no known discrete log relations between the elements of the generator set.*

*It can be shown that collision resistance is equivalent to the hardness of the discrete log problem XXX.*

**Example 68.**

### 6.2.2 Special Functions

**Theorem 6.2.2.1** (Pseudo-Randomness in cyclic groups)**.** *Let $\mathbb{G}$ be a cyclic group of order r, such that DDT is hard in $\mathbb{G}$, let g be a generator and $\{a_0, a_1, \ldots, a_n\} \subset \mathbb{F}_r^*$ a uniform randomly generated set of invertible field elements, from the scalar field of $\mathbb{G}$. Then the function*

$$H : \{0,1\}^n \to \mathbb{G}$$

*defined for any binary string $s = (s_1, \ldots, s_n)$ by $H(s) = [a_0 \cdot \Pi_{j=1}^n a_j^{s_j}]g$ is a pseudo-random function.*

## 6.3 Constructing Elliptic curves

**Order of an Elliptic Curve** An interesting question is: How many elements does a curve over a finite field contain? The first thing to note is that, since $\Phi$ from XXX is a one-to-one correspondence, affine curves have the same number of points as isomorphic projective curve. It is therefore enough to study the affine case only.

Since an affine short Weierstraß curve consists of pairs of elements from $\mathbb{F}_q$ plus the point at infinity and the field $\mathbb{F}_q$ contains $q$ elements, the curve can contain at most $q^2+1$ many elements. There is however a more precise estimation, usually called the **Hasse bound**. To understand it, let $E/\mathbb{F}_q$ be an affine short Weierstraß curve over a finite field $\mathbb{F}_q$ and let $|E/\mathbb{F}_q|$ be the number of elements in that curve. Then there is an integer $t \in \mathbb{Z}$ called the **trace**, with $|t| \leq 2\sqrt{q}$ and

$$|E/\mathbb{F}_q| = q + 1 - t \tag{6.8}$$

So roughly speaking, the number of elements in an elliptic curve is approximately equal to the size of an underlying field.

### 6.3.1 The Complex Multiplication Method

- Choose a prime number $p \in \mathbb{P}$ and integers $t, D \in \mathbb{Z}$, such that the equation $-Dv^2 = 4p - t^2$ has solutions $\pm v \in \mathbb{Z}$.
- If one of the values $p+1-t$ or $p+1+t$ a prime number, then proceed to the next steps, otherwise we go back to step 1.
- Compute the set $CL(Dv^2) = \{(a,b,c) \mid a,b,c \in \mathbb{Z}, |b| \leq a \leq \sqrt{\frac{Dv^2}{3}}, a \leq c, b^2 - 4ac = -Dv^2, (a,b,c) = 1WHATISTHIS?\}$. If $|b| = a$ or $a = c$, then $b \geq 0$.
- Compute $H_D(x) = \Pi_{(a,b,c)\in CL(D)}(x - j(\frac{-b+\sqrt{-Dv^2}}{2a}))$
- Round the coefficients of $H_D$ to the closed integers.
- Compute $H_{D,p} = H_D \; mod_p$
- Find a root $j$ of $H_{D,p}$
- If $j \neq 0$ or $j \neq 1728$, then choose $c \in \mathbb{F}_p$ and define the elliptic curve $E/\mathbb{F}_p$ defined by $y^2 = x^3 + 3kx + 2k$ for $k = \frac{j}{1728-j}$.

- Compute the order of $E/\mathbb{F}_p$. If it divides either $p+1-u$ or $p+1-u$, then $E/\mathbb{F}_p$ is the result.
- Otherwise choose $c \in \mathbb{F}_p$ with $c \neq 1$ and $c \neq 0$ and define the elliptic curve $E'/\mathbb{F}_p$ defined by $y^2 = x^3 + 3kc^2x + 2kc^3$ for $k = \frac{j}{1728-j}$.

## 6.4 Classes of elliptic curves

In this section we describes ways to describe elliptic curves different from the general Weierstrass form. Alternative descriptions are sometimes useful because of DIFFERENT ways to express the group laws

## 6.5 Pend and Paper example curves

### 6.5.1 BLS6-6 – our pen& paper curve

**Definition 6.5.1.1** (Cofactor Clearing)**.** *Since $BLS6-6(13)$ is a subgroup on our curve, it is not possible to leave the subgroup using the curves algebraic laws like scalar multiplication or addition. However in applications it often happens that random elements of the curve are generated, while what we really want are points in the subgroup. To get those points we can use cofactor clearing.*

In this example we want to use the complex multiplication method, to derive a pairing friendly elliptic curve that has similar properties to curves that are used in actual cryptographic protocols. However we design the curve specifically to be useful in pen&paper examples, which mostly means that the curve should contain only a few points, such that we are able to derive exhaustive addition and pairing tables.

A well understood family of pairing friendly curves are the BLS curves (STUFF ABOUT THE HISTORY AND THE NAMING CONVENTION)

BLS curves are particular useful in our case if the embedding degree $k$ satisfies $k \equiv 6 \pmod 0$. In this case the system of polynomials from section XXX parameterizes these curves.

Of course the smallest embedding degree $k$ that satisfies the congruency, is $k = 6$. We therefore aim for a BLS6 curve as our main pen&paper example.

As explained in XXX, the defining polynomials for any BLS6 curve are given by

$$\begin{aligned}
r(x) &= \Phi_6(x) \\
t(x) &= x+1 \\
q(x) &= \frac{1}{3}(x-1)^2(x^2-x+1)+x
\end{aligned}$$

where $\Phi_6$ is the 6-th cyclotomic polynomial. For any $x \in \mathbb{N}$, where $r(x), t(x), q(x)$ are natural numbers, with $q(x) > 3$ and $r(x) > 3$ those values describe elliptic curves with discriminant $D = 3$, characteristic $p(x)$, prime order subgroup $r(x)$ and Frobenious trace $t(x)$.

We start by looking-up the 6-th cyclotomic polynimial which is $\Phi_6 = x^2 - x + 1$ and then insert small values for $x$ into the defining polynomials $r, t, q$. This gives the following results:

$$\begin{array}{lll}
x=1 & (r(x),t(x),q(x)) & (1,2,1) \\
x=2 & (r(x),t(x),q(x)) & (3,3,3) \\
x=3 & (r(x),t(x),q(x)) & (7,4,\frac{37}{3}) \\
x=4 & (r(x),t(x),q(x)) & (13,5,43)
\end{array}$$

Since $q(1) = 1$ is not a prime number, the first $x$ that gives a proper curve is $x = 2$. However such a curve would be defined over a base field of characteristic 3 and we would rather like to avoid that. We therefore use $x = 4$, which defines a curve of fields of characteristic 43. Since the prime field $\mathbb{F}_{43}$ has 43 elements and 43 has binary representation 101011, which are 6 digits, the name of our pen&paper curve should be $BLS6-6$.

We can check that the embedding degree is indeed 6, since $k = 6$ is the smallest number $k$ such that $r = 13$ divides $43^k - 1$.

Strictly speaking BLS6-6 is not pairing friendly according to the definition in XXX, since indeed $r = 13 > \sqrt{43}$, but the second requirement is not satisfied. This however is irrelevant as the hole point of constructing this curve is to have a "large" prime oder subgroup that is as small as possible.

From the the defining equations of BLS curves, we can immediately deduce that BLS6-6 has a "large" subgroup of prime order 13, which is well suited for our purposes as 13 elements can be easily handled in the associated addition, scalar multiplication and pairing tables.

To see how the rest of the curve will look, we use XX to compute the number of rational points on the curve which is either $q+1-t$ or $q+1+t$. We get $43+1-5 = 39$ or $43+1+5 = 49$. Since our subgroup order $r = 13$ must divide the number of points, it follows that our curve has 39 point and hence a cofactor of 3, which implies that there is a single non trivial "small" order subgroup that contains three elements.

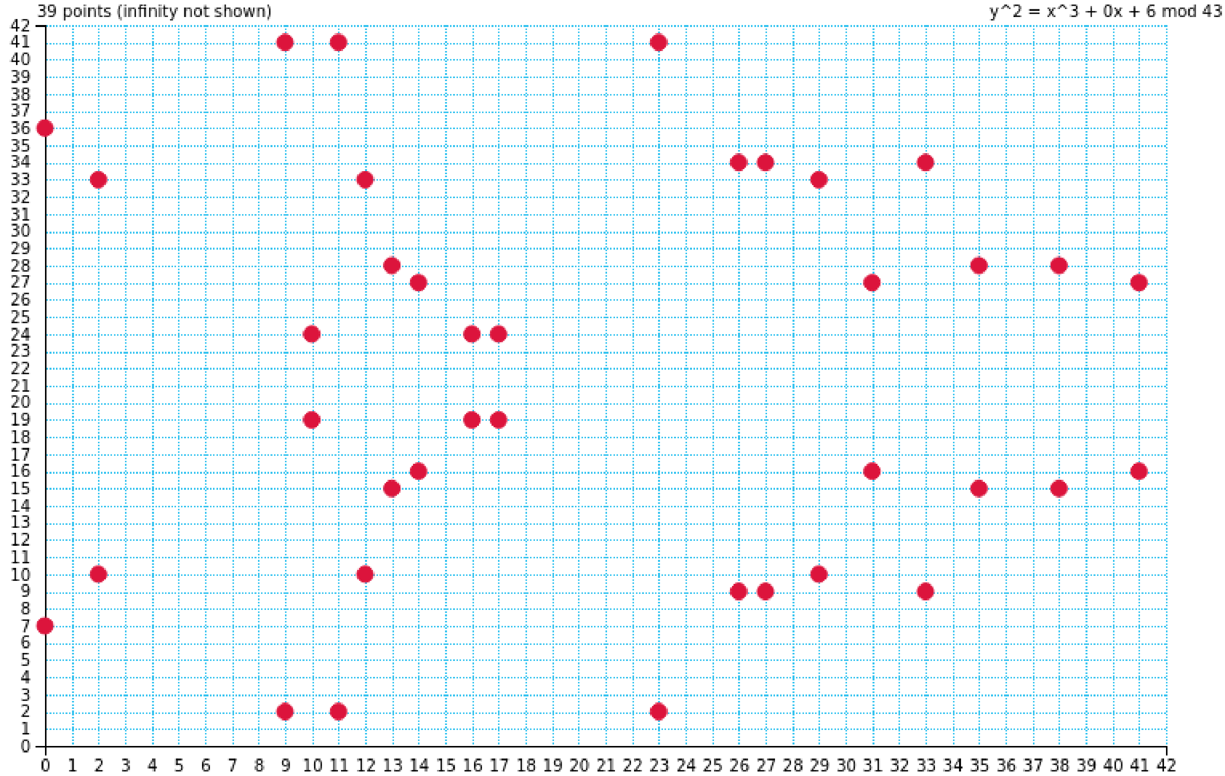
To compute the defining equation $y^2 = x^3 + ax + b$ of BLS6-6, we use the complex multiplication algorithm as described above in XXX. The goal is to find $a, b \in \mathbb{F}_{43}$ representations, that are particulary nice to work with. As shown for example in XXX the discriminant $D$ of all BLS curves is $-3$, which gives them the general form $y^2 = x^3 + b$.

This is because the Hilbert class polynomial $H_3(x) = x$, since $CL(3) = \{[1,1,1]\}$ and in this case $j(\frac{-1+i\sqrt{3}}{1}) = 0$. It follows that the general curve equation is given by $y^2 = x^3 + b$ and it only remains to find $b$, such that the curve has the correct number of points which is 39. Since $b \in \mathbb{F}_{43}$, we can just put values for $b$ into the equation and count points. The smallest value then is $b$ and we get

$$BLS6-6: \quad y^2 = x^3 + 6 \quad \text{for all } x, y \in \mathbb{F}_{43}$$

There are other choice for $b$ like $b = 10$ or $b = 23$, but all these curves are isomorphic and hence represent the same thing really but in different way only.

Since BLS6-6 only contains 39 points it is possible to give a visual impression of the curve:




As we can see our curve is somewhat nice, as it does not contain self inverse points that is points with $y = 0$. It follows that the addition law can be optimized, since the branch for those cases can be eliminated.

Note: Is there a way to printe the entire addition table from https://graui.de/code/elliptic2/ here? Would be nice to have but is a bit large.

Since the order of BLS6-6 is $39 = 3 \cdot 13$, we know that it has a "large" subgroup of order 13 and small subgroup of order 3. We can use XXX to find those groups. We have $BLS6-6(3) = \{\mathcal{O}, (0,7), (0,36)\}$.

In addition we have the generator $g_{BLS6} := (13,15)$ that generates

$$\begin{aligned} BLS6-6(13) = & \\ & \{(13,15) \to (33,34) \to (38,15) \to (35,28) \to (26,34) \to (27,34) \to \\ & (27,9) \to (26,9) \to (35,15) \to (38,28) \to (33,9) \to (13,28) \to \mathcal{O}\} \end{aligned} \tag{6.9}$$

Computations "in the exponent": In cryptography and in particular in snarks a lot HAPPENS IN THE EXPONENT...

To use our example to explain what this means observe that from this representation, we can deduce a map from the scalar field $\mathbb{F}_{13}$ to $BLS6-6(13)$ with respect to our generator. WE have

$$[\cdot]_{(13,15)} : \mathbb{F}_{13} \to BLS6-6(13)\ ;\ x \mapsto [x](13,15)$$

So for example we have $[1]_{(13,15)} = (13,15)$, $[7]_{(13,15)} = (27,9)$ and $[0]_{(13,15)} = \mathcal{O}$. In particular this map is a homomorphism of groups from the additive group $\mathbb{F}_{13}$ to $BLS6-6(13)$. This means in particular, the the additive neutral element from $\mathbb{F}_{13}$ is mapped to $\mathcal{O}$ and negatives are mapped to inverses. For example $[-2]_{(13,15)} = -[2]_{(13,15)}$, since $[-2]_{(13,15)} = [11]_{(13,15)} = (33,9) = (33,-34) = -(33,34) = -[2]_{(13,15)}$

The map also give a visualization of the ECDL problem in $BLS6-6(13)$, which is concerned with finding solutions $x \in \mathbb{F}_{13}$ for the equation $[x]_{(13,15)} = (x,y)$ for any $(x,y) \in BLS6-6(13)$.

Of course ECDL is not hard in $BLS6-6(13)$, since we can deduce the solutions easily from XXX. For example the solution to $[x]_{(13,15)} = (35,15)$ is $x = 9$, since $[9](13,15) = (35,15)$.

Since $[0]_{(13,15)}$ maps the group of cyclic integers modulo 13 onto our group $BLS6-6(13)$, we can use this to write down the group law in the following way:

| $\cdot$ | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) | (35,28) | (26,34) | (27,34) | (27,9) | (26,9) | (35,15) | (38,28) | (33,9) | (13,28) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{O}$ | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) | (35,28) | (26,34) | (27,34) | (27,9) | (26,9) | (35,15) | (38,28) | (33,9) | (13,28) |
| (13,15) | (13,15) | (33,34) | (38,15) | (35,28) | (26,34) | (27,34) | (27,9) | (26,9) | (35,15) | (38,28) | (33,9) | (13,28) | $\mathcal{O}$ |
| (33,34) | (33,34) | (38,15) | (35,28) | (26,34) | (27,34) | (27,9) | (26,9) | (35,15) | (38,28) | (33,9) | (13,28) | $\mathcal{O}$ | (13,15) |
| (38,15) | (38,15) | (35,28) | (26,34) | (27,34) | (27,9) | (26,9) | (35,15) | (38,28) | (33,9) | (13,28) | $\mathcal{O}$ | (13,15) | (33,34) |
| (35,28) | (35,28) | (26,34) | (27,34) | (27,9) | (26,9) | (35,15) | (38,28) | (33,9) | (13,28) | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) |
| (26,34) | (26,34) | (27,34) | (27,9) | (26,9) | (35,15) | (38,28) | (33,9) | (13,28) | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) | (35,28) |
| (27,34) | (27,34) | (27,9) | (26,9) | (35,15) | (38,28) | (33,9) | (13,28) | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) | (35,28) | (26,34) |
| (27,9) | (27,9) | (26,9) | (35,15) | (38,28) | (33,9) | (13,28) | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) | (35,28) | (26,34) | (27,34) |
| (26,9) | (26,9) | (35,15) | (38,28) | (33,9) | (13,28) | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) | (35,28) | (26,34) | (27,34) | (27,9) |
| (35,15) | (35,15) | (38,28) | (33,9) | (13,28) | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) | (35,28) | (26,34) | (27,34) | (27,9) | (26,9) |
| (38,28) | (38,28) | (33,9) | (13,28) | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) | (35,28) | (26,34) | (27,34) | (27,9) | (26,9) | (35,15) |
| (33,9) | (33,9) | (13,28) | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) | (35,28) | (26,34) | (27,34) | (27,9) | (26,9) | (35,15) | (38,28) |
| (13,28) | (13,28) | $\mathcal{O}$ | (13,15) | (33,34) | (38,15) | (35,28) | (26,34) | (27,34) | (27,9) | (26,9) | (35,15) | (38,28) | (33,9) |

Cofactor clearing:

Given an arbitrary point on the curve that is not in any of our two subgroups like $(2,33)$, we can project it on both subgroups $BLS6-6(3)$ and $BLS6-6(13)$ respectively, by *multiplication with the cofactor*. Since $39 = 3 \cdot 13$, we have to multiply $(2,33)$ with 13 to map it onto $BLS6-6(3)$ and we have to multiply $(2,33)$ with 3 to map it onto $BLS6-6(13)$. Indeed we get $[13](2,33) = (0,36)$ which is an element of $BLS6-6(3)$ and $[3](2,33) = (35,15)$ which is an element of $BLS6-6(13)$

In what follows we want to compute type 2 pairings on our BLS6 curve. We therefore need to extract the subgroup $\mathbb{G}_1$ as well as $\mathbb{G}_2$ from the full 13-torsion group. We already know from XXX that $\mathbb{G}_1$ is given by

$$\mathbb{G}_1 = \{(13,15) \to (33,34) \to (38,15) \to (35,28) \to (26,34) \to (27,34) \to \\ (27,9) \to (26,9) \to (35,15) \to (38,28) \to (33,9) \to (13,28) \to \mathcal{O}\}$$

In type 2 pairings, the group $\mathbb{G}_2$ is defined by those elements $P$ of the full 13-torsion group, that are mapped to $43 \cdot P$ under the Frobenius endomorphism XXX. Since $BLS6/\mathbb{F}_{13^6}$ contains 6321251664 elements, we can not simply loop through all elements, to find the full 13-torsion group and extract all elements from $\mathbb{G}_2$. However we can derive the full 13-torsion as the set of all 13-division points and then extract $G_2$ from this

```
sage: F43 = GF(43)
sage: F43t.<t> = F43[]
sage: F43_6.<v> = GF(43^6, name='v', modulus=t^6+6) # t^6+6
   irreducible
sage: BLS6 = EllipticCurve (F43_6,[0 ,6])
sage: INF = BLS6(0) # point at infinity
sage: for P in INF.division_points(13): # PI(P) == [q]P
....:     if P.order() == 13: # exclude point at infinity
....:         PiP = BLS6([a.frobenius() for a in P])
....:         qP = 43*P
....:         if PiP == qP:
....:             print(P.xy())
```

Choose $g_2 = (7v^2, 16v^3)$ as generator of $\mathbb{G}_2$, we get

$$\begin{aligned}\mathbb{G}_2 = \{&(7v^2,16v^3) \to (10v^2,28v^3) \to (42v^2,16v^3) \to (37v^2,27v^3) \to \\ &(16v^2,28v^3) \to (17v^2,28v^3) \to (17v^2,15v^3) \to (16v^2,15v^3) \to \\ &(37v^2,16v^3) \to (42v^2,27v^3) \to (10v^2,15v^3) \to (7v^2,27v^3) \to \mathcal{O}\}\end{aligned}$$

e.g. $[3]g_2 = (42v^2, 16v^3)$.

Having those groups we can do pairings. We choose the Weil pairing and invoke sagemath. For example the Weil pairing between our two generators is

$$e(g_1, g_2) = 5v^5 + 16v^4 + 16v^3 + 15v^2 + 3v + 41$$

```
sage: g1 = BLS6([13,15])
sage: g2 = BLS6([7*v^2, 16*v^3])
sage: g1.weil_pairing(g2,13)
5*v^5 + 16*v^4 + 16*v^3 + 15*v^2 + 3*v + 41
```

As we have seen, $\mathbb{G}_2$ needs quite a bit more storage space then $\mathbb{G}_1$, since elements in $\mathbb{G}_2$ are pairs of polynomials of degree $< 6$ with coefficients in $\mathbb{F}_{43}$, while elements from $\mathbb{G}_1$ are just pairs of elements from $\mathbb{F}_{43}$.

As we know from XXX it is possible to reduce the space needed to store $\mathbb{G}_2$ by using the concept of a twist. In our case *BLS*6 has embedding degree 6 and the curve parameter $a$ in $y^2 = x^3 + ax + b$ is zero. We therefore know from XXX, that BLS6 has three different twist: A quadratic twist, a cubic twist and a sextic twist. We want to compute all of these twist:

The quadratic twisted BLS6-6 curve: Consider our BLS6-6 curve $BLS6-6/\mathbb{F}_{43^6}$. A quadratic twist is then another curve $BLS6-6_{2-twist}$ over $\mathbb{F}_{43^3}$ isomorphic to the original curve. We use XXX. The task is to find an $\omega \in \mathbb{F}_{43^6}$, such that $\omega^2 \in \mathbb{F}_{43^3}$. We choose $\omega = x^4 + 7x^3 + 9x^2 + 11x + 8$. Then we interpret $\delta = \omega^2 = 27x^2 + 17x + 35$ as an element from $\mathbb{F}_{43^3}$. So our twisted curve is $y^2 = x^3 + a\delta^2 x + b\delta^3 = x^3 + 6 \cdot (27t^2 + 17t + 35)$ so we get

$$BLS6-6_{2-twist}/\mathbb{F}_{43^3} : y^2 = x^3 + (10t^2 + 14t + 15)$$

### Baby JubJub

To give an understanding what the Baby-JubJub curve is, we want to parallel its development here to find a Baby-Jubjub like curve for pen and paper.

As with the original large Baby-JubJub curve we apply the method from to define a pen& paper Baby-JubJub-like curve over the scalar field of the "large" BLS6 prime order subgroup, which is $\mathbb{F}_{13}$.

Since $13 \bmod 4 = 1$ we would go with A.1. As we will only find a few curves, we will tweak the algorithm and run

```
sage: F13 = GF(13)
sage: for A in xrange(3, 13):
....:     if (A-2) % 4 != 0:
....:         continue
....:     try:
....:         E = EllipticCurve(F13, [0, A, 0, 1, 0]) #
   Montgomery form
```

```
....:         E
....:         E.order()
....:     except:
....:         continue
```

So we get two curves in Montgomery form $y^2 = x^3 + 6 * x^2 + x$ which has order 8 and $y^2 = x^3 + 10 * x^2 + x$, which has order 16. We could transform one of them into an Edwards curve, however

So to find our Edwards curve, we will do exhaustive search rather

```
sage: for d in F13:
....:     j= ZZ(0)
....:     for x in F13:
....:         for y in F13:
....:             if x^2+y^2 == 1+d*x^2*y^2:
....:                 j=j+1
....:     print('d=',d)
....:     print('order=',j)
```

and get $x^2 + y^2 = 1 + 7 \cdot x^2 y^2$ which has 20 points. The associated Montgomery curve is then using XXX given by $8y^2 = x^3 + 6 \cdot x^2 + x$.

So we define our Baby-JubJub Edwards curve to be

$$EdBJJ/\mathbb{F}_{13} : x^2 + y^2 = 1 + 7 \cdot x^2 y^2$$

with associated Montgomery form to be

$$MBJJ/\mathbb{F}_{13} : 8y^2 = x^3 + 6 \cdot x^2 + x$$

As $20 = 2 \cdot 2 \cdot 5$, we have a "large" prime order subgroup of order 5 and a cofactor 4. The group of rational points is

```
sage: for x in F13:
....:     for y in F13:
....:         if x^2+y^2 == F13(1)+F13(7)*x^2*y^2:
....:             print(x,y)
```

$(0,1),(0,12),(1,0),(2,4),(2,9),(4,2),(4,11),(5,6),(5,7),(6,5),(6,8),(7,5),(7,8),(8,6),(8,7),(9,2),(9,11$

with neutral element $(0,12)$

As expected we have a prime order subgroup of size 5, which can be generated by $(11,9)$. We get $\{(11,9) \to (6,8) \to (7,8) \to (2,9) \to (0,1))\}$.

```
sage: def Edwards_add((x1,y1),(x2,y2),d):
....:     x3 = F13((F13(x1)*F13(y2)+F13(y1)*F13(x2))/((F13(1)+
    F13(d)*F13(x1)*F13
....:  (x2)*F13(y1)*F13(y2))))
....:     y3 = F13((F13(y1)*F13(y2)-F13(x1)*F13(x2))/((F13(1)-
    F13(d)*F13(x1)*F13
....:  (x2)*F13(y1)*F13(y2))))
....:     return (x3,y3)
```

**Hashing to the pairing groups** We give various constructions to hash into $\mathbb{G}_1$ and $\mathbb{G}_2$.

We start with hashing to the scalar field... TO APPEAR

Non of these techniques work for hashing into $\mathbb{G}_2$. We therefore implement Pederson's Hash for BLS6.

We start with $\mathbb{G}_1$. Our goal is to define an 12-bit bounded hash function

$$H_1 : \{0,1\}^{12} \to \mathbb{G}_1$$

Since $12 = 3 \cdot 4$ we "randomly" select 4 uniformly distributed generators $\{(38,15),(35,28),(27,34),(38,28)\}$ from $\mathbb{G}_1$ and use the pseudo-random function from XXX. For every genrator we therefore have to choose a set of 4 randomly generated invertible elements from $\mathbb{F}_{13}$. We choose

$$\begin{array}{lcl} (38,15) & : & \{2,7,5,9\} \\ (35,28) & : & \{11,4,7,7\} \\ (27,34) & : & \{5,3,7,12\} \\ (38,28) & : & \{6,5,1,8\} \end{array}$$

So our hash function is computed like this:

$$\begin{aligned} H_1(x_{11},x_1,\ldots,x_0) = [2 \cdot 7^{x_{11}} \cdot 5^{x_{10}} \cdot 9^{x_9}](38,15) + [11 \cdot 4^{x_8} \cdot 7^{x_7} \cdot 7^{x_6}](35,28) + \\ [5 \cdot 3^{x_5} \cdot 7^{x_4} \cdot 12^{x_3}](27,34) + [6 \cdot 5^{x_2} \cdot 1^{x_1} \cdot 8^{x_0}](38,28) \end{aligned}$$

Note that $a^x = 1$ whe $x = 0$ and hence those terms can be omitted in the computation. In particular the hash of the 12-bit zero string is given by

$$\begin{aligned} WRONG-ORDERING-REDOH_1(0) = [2](38,15)+[11](35,28)+[5](27,34)+[6](38,28) = \\ (27,34)+(26,34)+(35,28)+(26,9) = (33,9)+(13,28) = (38,28) \end{aligned}$$

The hash of 011010101100 is given by

$$\begin{aligned} H_1(011010101100) = WRONG-ORDERING-REDO \\ [2 \cdot 7^0 \cdot 5^1 \cdot 9^1](38,15)+[11 \cdot 4^0 \cdot 7^1 \cdot 7^0](35,28)+[5 \cdot 3^1 \cdot 7^0 \cdot 12^1](27,34)+[6 \cdot 5^1 \cdot 1^0 \cdot 8^0](38,28) = \\ [2 \cdot 5 \cdot 9](38,15)+[11 \cdot 7](35,28)+[5 \cdot 3 \cdot 12](27,34)+[6 \cdot 5](38,28) = \\ [12](38,15)+[12](35,28)+[11](27,34)+[4](38,28) = \\ TO_APPEAR \end{aligned}$$

We can use the same technique to define a 12-bit bounded hash function in $\mathbb{G}_2$:

$$H_2 : \{0,1\}^{12} \to \mathbb{G}_2$$

Again we "randomly" select 4 uniformly distributed generators $\{(7v^2,16v^3),(42v^2,16v^3),(17v^2,15v^3),(10v^2,1$ from $\mathbb{G}_2$ and use the pseudo-random function from XXX. For every genrator we therefore have to choose a set of 4 randomly generated invertible elements from $\mathbb{F}_{13}$. We choose

$$\begin{array}{lcl} (7v^2,16v^3) & : & \{8,4,5,7\} \\ (42v^2,16v^3) & : & \{12,1,3,8\} \\ (17v^2,15v^3) & : & \{2,3,9,11\} \\ (10v^2,15v^3) & : & \{3,6,9,10\} \end{array}$$

So our hash function is computed like this:

$$H_1(x_{11}, x_{10}, \ldots, x_0) = [8 \cdot 4^{x_{11}} \cdot 5^{x_{10}} \cdot 7^{x_9}](7v^2, 16v^3) + [12 \cdot 1^{x_8} \cdot 3^{x_7} \cdot 8^{x_6}](42v^2, 16v^3) + \\ [2 \cdot 3^{x_5} \cdot 9^{x_4} \cdot 11^{x_3}](17v^2, 15v^3) + [3 \cdot 6^{x_2} \cdot 9^{x_1} \cdot 10^{x_0}](10v^2, 15v^3)$$

We extend this to a hash function that maps unbounded bitstring to $\mathbb{G}_2$ by precomposing with an actual haah function like $MD5$ and feet the first 12 bits of its outcome into our previously defined hash function.

$$TinyMD5_{\mathbb{G}_2} : \{0,1\}^* \to \mathbb{G}_2$$

with $TinyMD5_{\mathbb{G}_2}(s) = H_2(MD5(s)_0, \ldots MD5(s)_{11})$. For example, since $MD5(\text{""}) = 0xd41d8cd98f00b204e98$ and the binary representation of the hexadecimal number $0x27e$ is $001001111110$ we compute $TinyMD5_{\mathbb{G}_2}$ of the empty string as $TinyMD5_{\mathbb{G}_2}(\text{""}) = H_2(MD5(s)_{11}, \ldots MD5(s)_0) = H2(001001111110) =$

### Baby-JubJub-2

To give an understanding what the Baby-JubJub curve is, we want to parallel its development here to find a Baby-Jubjub like curve for pen and paper.

The original Baby-JubJub is a twisted Edwards curve over $\mathbb{F}_?$ with $a = -1$ and $d = ?$.

As with the original large Baby-JubJub curve we apply the method from to define a pen& paper Baby-JubJub-like curve over the scalar field of the "large" BLS6 prime order subgroup, which is $\mathbb{F}_{13}$.

Since 13 mod $4 = 1$ we would go with A.1. As we will only find a few curves, we will tweak the algorithm and run

```
sage: F13 = GF(13)
sage: for A in xrange(3, 13):
....:     if (A-2) % 4 != 0:
....:         continue
....:     try:
....:         E = EllipticCurve(F13, [0, A, 0, 1, 0]) #
   Montgomery form
....:         E
....:         E.order()
....:     except:
....:         continue
```

So we get two curves in Montgomery form $y^2 = x^3 + 6 * x^2 + x$ which has order 8 and $y^2 = x^3 + 10 * x^2 + x$, which has order 16. We could transform one of them into an Edwards curve, however

So to find our Edwards curve, we will do exhaustive search rather

```
sage: j = ZZ(0)
sage: for a in F13:
....:     for d in F13:
....:         j = 0
....:         for x in F13:
....:             for y in F13:
....:                 if a*x^2 + y^2 == 1+d*x^2*y^2:
....:                     j=j+1
....:         print('curve: a=',a,'d=',d,'order:',j)
```

We want to choose a curve that has a large prime order subgroup and a small cofactor. So we go with $2x^2 + y^2 = 1 + 3 \cdot x^2 y^2$ which has order 14.

The associated Montgomery curve is then using XXX given by $9y^2 = x^3 + 2x^2 + x$.

So we define our Baby-JubJub Edwards curve to be

$$EdBJJ/\mathbb{F}_{13} : 2x^2 + y^2 = 1 + 3 \cdot x^2 y^2$$

with associated Montgomery form to be

$$MBJJ/\mathbb{F}_{13} : 9y^2 = x^3 + 2x^2 + x$$

As $14 = 2 \cdot 7$, we have a "large" prime order subgroup of order 7 and a cofactor 2. The group of rational points is

```
sage: for x in F13:
....:     for y in F13:
....:         if F13(2)*x^2+y^2 == F13(1)+F13(11)*x^2*y^2:
....:             print(x,y)
```

$(0,1),(0,12),(2,4),(2,9),(4,5),(4,8),(5,2),(5,11),(8,2),(8,11),(9,5),(9,8),(11,4),(11,9)$

with neutral element $(0,1)$

As expected we have a prime order subgroup of size 5, which can be generated by $(11,9)$. We get $\{(11,9) \to (6,8) \to (7,8) \to (2,9) \to (0,1))\}$.

```
sage: def Edwards_add((x1,y1),(x2,y2),a,d):
....:     x3 = F13((F13(x1)*F13(y2)+F13(y1)*F13(x2))/((F13(1)+
   F13(d)*F13(x1)*F13(x2)*F13(y1)*F13(y2))))
....:     y3 = F13((F13(y1)*F13(y2)-F13(a)*F13(x1)*F13(x2))/((
   F13(1)-F13(d)*F13(x1)*F13(x2)*F13(y1)*F13(y2))))
....:     return (x3,y3)
```

### 6.5.2 MNT4 MNT6 Cycles

**Theorem 6.5.2.1.** *Let $q$ be a prime and $E/\mathbb{F}_q$ be an ordinary elliptic curve such that $r = |E(Fq)|$ is a prime greater than 3.*

- *$E$ has embedding degree $k = 4$ if and only if there exists $x \in \mathbb{Z}$ such that $t = -x$ or $t = x+1$, and $q = x^2 + x + 1$.*
- *$E$ has embedding degree $k = 6$ if and only if there exists $x \in \mathbb{Z}$ such that $t = 1 \pm 2x$ and $q = 4x^2 + 1$.*
- *There is an elliptic curve $E/\mathbb{F}_q$ with embedding degree 6, discriminant $D$, and $|E(Fq)| = r$ if and only if there is an elliptic curve $E'/\mathbb{F}_r$ with embedding degree 4, discriminant $D$, and $|E'(\mathbb{F}_r)| = q$.*

We can use this theorem to find an MNT6-MNT4 cycle over very small prime fields with characteristics $> 3$:

**MNT4** For our MNT4 curve, we can choose $x = 2$. Then $q = 7$ and if we choose $t = x+1$ then $r = q+1-t = 7+1-3 = 5$. Therefore our MNT4 curve is a curve $y^2 = x^3 + ax + b$ defined over $\mathbb{F}_7$ that consists of 5 points.

To construct the actual curve we could use the complex multiplication method again, but since the parameters $a$ and $b$ are from $\mathbb{F}_7$ there are only 48 possibilities so we simply loop through all possible $a$'s and $b$'s and count the curve points until we find a curve that has 5 rational points. We get

$$y^2 = x^3 + 4x + 1$$

defined over $\mathbb{F}_7$, with scalar field $\mathbb{F}_5$. Since $7 = 2^2 + 2 + 1$, we know from theorem XXX, that this curve has embedding degree 4 and hence qualifies as a pen&paper pairing friendly elliptic curve. Since the curve's order is a prime and therefore has no non trivial factors, it has no non trivial subgroups. The curve has the following set of elements

$$MNT4 = \{(0,1) \to (0,6) \to (4,2) \to (4,5) \to \mathcal{O}\}$$

```
sage: F7 = GF(7)
sage: MNT4 = EllipticCurve (F7,[4 ,1])
sage: [P.xy() for P in MNT4.points() if P.order() > 1]
[(0, 1), (0, 6), (4, 2), (4, 5)]
```

The multiplication table is

| $\cdot$ | $\mathcal{O}$ | $(0,1)$ | $(4,5)$ | $(4,2)$ | $(0,6)$ |
|---|---|---|---|---|---|
| $\mathcal{O}$ | $\mathcal{O}$ | $(0,1)$ | $(4,5)$ | $(4,2)$ | $(0,6)$ |
| $(0,1)$ | $(0,1)$ | $(4,5)$ | $(4,2)$ | $(0,6)$ | $\mathcal{O}$ |
| $(4,5)$ | $(4,5)$ | $(4,2)$ | $(0,6)$ | $\mathcal{O}$ | $(0,1)$ |
| $(4,2)$ | $(4,2)$ | $(0,6)$ | $\mathcal{O}$ | $(0,1)$ | $(4,5)$ |
| $(0,6)$ | $(0,6)$ | $\mathcal{O}$ | $(0,1)$ | $(4,5)$ | $(4,2)$ |

In what follows we choose our generator to be $g_{MNT4} = (0,1)$.

In what follows we want to compute type 2 pairings on our MNT4 curve. We therefore need to extract the subgroup $\mathbb{G}_1$ as well as $\mathbb{G}_2$ from the full 5-torsion group. Since the order of MNT4 is a prime number, we already know from XXX that $\mathbb{G}_1$ is given by

$$\mathbb{G}_1 = \{(0,1) \to (0,6) \to (4,2) \to (4,5) \to \mathcal{O}\}$$

In type 2 pairings, the group $\mathbb{G}_2$ is defined by those elements $P$ of the full 5-torsion group, that are mapped to $7 \cdot P$ under the Frobenius endomorphism XXX. Since $MNT4/\mathbb{F}_{7^4}$ only contains 2475 elements, we can loop through all elements, to find the full 5-torsion group and extract all elements from $\mathbb{G}_2$:

```
sage: F7t.<t> = F7[]
sage: F7_4.<u> = GF(7^4, name='u', modulus=t^4+t+1) # embedding degree is 4
sage: MNT4 = EllipticCurve (F7_4,[4 ,1])
sage: INF = MNT4(0) # point at infinity
```

```
sage: for P in INF.division_points(5): # PI(P) == [q]P
....:     if P.order() == 5: # exclude point at infinity
....:         PiP = MNT4([a.frobenius() for a in P])
....:         qP = 7*P
....:         if PiP == qP:
....:             print(P.xy())
```

Choose $g_2 = (2u^3+5u^2+4u+2, 2u^3+3u+5)$ as generator of $\mathbb{G}_2$, we get

$$\mathbb{G}_2 = \{(2u^3+5u^2+4u+2, 2u^3+3u+5) \to (5u^3+2u^2+3u+6, 2u^2+3u) \to (5u^3+2u^2+3u+6, 5u^2+4u) \to (2u^3+5u^2+4u+2, 5u^3+4u+2) \to \mathcal{O}\}$$

e.g. $[3]g_2 = (5u^3+2u^2+3u+6, 5u^2+4u)$.

Having those groups we can do pairings. We choose the Weil pairing and invoke sagemath. For example the Weil pairing between our two generators is

$$e(g_1, g_2) = 5u^3+2u^2+6u$$

```
sage: g1 = MNT4([0,1])
sage: g2 = MNT4(2*u^3 + 5*u^2 + 4*u + 2, 2*u^3 + 3*u + 5)
sage: g1.weil_pairing(g2,5)
5*u^3 + 2*u^2 + 6*u
```

The full pairing table can the be written as

| $e(\cdot,\cdot)$ | $\mathcal{O}$ | $g_1$ | $[2]g_1$ | $[3]g_1$ | $[4]g_1$ |
|---|---|---|---|---|---|
| $\mathcal{O}$ | 1 | 1 | 1 | 1 | 1 |
| $g_2$ | 1 | $5u^3+2u^2+6u$ | $6u^3+5u^2+6$ | $2u^3+u^2+2u+3$ | $u^3+6u^2+6u+4$ |
| $[2]g_2$ | 1 | $6u^3+5u^2+6$ | $u^3+6u^2+6u+4$ | $5u^3+2u^2+6u$ | $2u^3+u^2+2u+3$ |
| $[3]g_2$ | 1 | $2u^3+u^2+2u+3$ | $5u^3+2u^2+6u$ | $u^3+6u^2+6u+4$ | $6u^3+5u^2+6$ |
| $[4]g_2$ | 1 | $u^3+6u^2+6u+4$ | $2u^3+u^2+2u+3$ | $6u^3+5u^2+6$ | $5u^3+2u^2+6u$ |

**MNT6** For our MNT6 curve, we can choose $x = 1$. Then $q = 5$ and if we choose $t = 1+2x$ then $r = q+1-t = 5+1+1 = 7$. Therefore our MNT6 curve is a curve $y^2 = x^3+ax+b$ defined over $\mathbb{F}_5$ that consists of 7 points.

To construct the actual curve we could use the complex multiplication method again, but since the parameters $a$ and $b$ are from $\mathbb{F}_5$ there are only 24 possibilities, we simply loop through all possible $a$'s and $b$'s and count the curve points until we find a curve that has 7 rational points. We get

$$y^2 = x^3+2x+1$$

defined over $\mathbb{F}_5$. Since $5 = 4\cdot 1+1$, we know from theorem XXX, that this curve has embedding degree 6 and hence qualifies as a pen&paper pairing friendly elliptic curve.

The curve has the following set of elements

$$MNT6 = \{(1,2) \to (3,3) \to (0,1) \to (0,4) \to (3,2) \to (1,3) \to \mathcal{O}\}$$

The multiplication table is

| $\cdot$ | $\mathcal{O}$ | $(1,2)$ | $(3,3)$ | $(0,1)$ | $(0,4)$ | $(3,2)$ | $(1,3)$ |
|---|---|---|---|---|---|---|---|
| $\mathcal{O}$ | $\mathcal{O}$ | $(1,2)$ | $(3,3)$ | $(0,1)$ | $(0,4)$ | $(3,2)$ | $(1,3)$ |
| $(1,2)$ | $(1,2)$ | $(3,3)$ | $(0,1)$ | $(0,4)$ | $(3,2)$ | $(1,3)$ | $\mathcal{O}$ |
| $(3,3)$ | $(3,3)$ | $(0,1)$ | $(0,4)$ | $(3,2)$ | $(1,3)$ | $\mathcal{O}$ | $(1,2)$ |
| $(0,1)$ | $(0,1)$ | $(0,4)$ | $(3,2)$ | $(1,3)$ | $\mathcal{O}$ | $(1,2)$ | $(3,3)$ |
| $(0,4)$ | $(0,4)$ | $(3,2)$ | $(1,3)$ | $\mathcal{O}$ | $(1,2)$ | $(3,3)$ | $(0,1)$ |
| $(3,2)$ | $(3,2)$ | $(1,3)$ | $\mathcal{O}$ | $(1,2)$ | $(3,3)$ | $(0,1)$ | $(0,4)$ |
| $(1,3)$ | $(1,3)$ | $\mathcal{O}$ | $(1,2)$ | $(3,3)$ | $(0,1)$ | $(0,4)$ | $(3,2)$ |

In what follows we choose our generator to be $g_{MNT6} = (1,2)$.

In what follows we want to compute type 2 pairings on our MNT6 curve. We therefore need to extract the subgroup $\mathbb{G}_1$ as well as $\mathbb{G}_2$ from the full 7-torsion group. Since the order of MNT6 is a prime number, we already know from XXX that $\mathbb{G}_1$ is given by

$$\mathbb{G}_1 = \{(1,2) \to (3,3) \to (0,1) \to (0,4) \to (3,2) \to (1,3) \to \mathcal{O}\}$$

In type 2 pairings, the group $\mathbb{G}_2$ is defined by those elements $P$ of the full 7-torsion group, that are mapped to $5 \cdot P$ under the Frobenius endomorphism XXX. Since $MNT6/\mathbb{F}_{5^6}$ contains 15680 elements, we can still loop through all elements, to find the full 7-torsion group and extract all elements from $\mathbb{G}_2$

```
sage: G.<x> = GF(5^6) # embedding degree is 6
sage: MNT6 = EllipticCurve (G,[2 ,1])
sage: INF = MNT6(0) # point at infinity
sage: for P in INF.division_points(7): # PI(P) == [q]P
....:     if P.order() == 7: # exclude point at infinity
....:         PiP = MNT6([a.frobenius() for a in P])
....:         qP = 5*P
....:         if PiP == qP:
....:             print(P.xy())
```

$$\begin{aligned}\mathbb{G}_2 = \{&(x^3+2x^2+4x, x^5+2x^4+4x^3+3x^2+3) \to (x^5+4x^4+2x^3+3x^2+x+2, 3x^4+2x^3+x) \to \\ &(4x^5+x^4+2x^3, 3x^5+x^4+x^3+4x+4) \to (4x^5+x^4+2x^3, 2x^5+4x^4+4x^3+x+1) \to \\ &(x^5+4x^4+2x^3+3x^2+x+2, 2x^4+3x^3+4x) \to (x^3+2x^2+4x, 4x^5+3x^4+x^3+2x^2+2) \to \mathcal{O}\}\end{aligned}$$

We choose the generator $g_2 = (x^3+2x^2+4x, x^5+2x^4+4x^3+3x^2+3)$

**Remark 2.** *Note however that our MNT6 curve discriminant $D = -16(4a^3+27b^2) = -16(4 \cdot 2^3 + 27 \cdot 1^2) = -944$, while our MNT4 curve has discriminsnt XXX. Hence our example curves are not those guranteed by theorem XXX. Those curve are both given by $y^2 = x^3+2x+1$ over $\mathbb{F}_5$ and $\mathbb{F}_7$, respectively. However as both curves have the same defining equation, we rather choose examples that are visually distinguishable by their defining equations.*

### 6.5.3 Edwards curve cycles

# 7 Zk-Proof Systems

TODO:
Barrett reduction
Montgomery modular multiplication (Montgomery domain)
Some philosophical stuff about compuational models for snarks. Bounded computability...

## 7.1 Computational Models

Proofs are the evidence of correctness of the assertions, and people can verify the cor-rectness by reading the proof. However, we obtain much more than the correctness itself:After you read one proof of an assertion, you know not only the correctness, but also why itis correct. Is it possible to solely show the correctness of an assertion without revealing theknowledge of proofs? It turns out that it is indeed possible, and this is the topic of today'slecture: Zero Knowledge Systems.

**Example 69** (Generalized factorization snark)**.** *As one of our major running examples we want to derive a zk-SNARK for the following generalized factorization problem:*

*Given two numbers $a,b \in \mathbb{F}_{13}$, find two additional numbers $x,y \in \mathbb{F}_{13}$, such that*

$$(x \cdot y) \cdot a = b$$

*and proof knowledge of those numbers, without actually revealing them.*

*Of course this example reduces to the classic factorization problem (over $\mathbb{F}_{13}$ by setting $y = 1$)*

*This zero knowledge system deals with the following situation: "Given two publicly known numbers $a,b \in \mathbb{F}_{13}$ a proofer can show that they know two additional numbers $x,y \in \mathbb{F}_{13}$, such that $(x \cdot y) \cdot a = b$, without actually revealing $x$ or $y$."*

*Of course our choice of what information to hide and what to reveal was completely arbitrary. Every other split would also be possible, but eventually gives a different problem.*

*For example the task could be to not hide any of the variables. Such a system has no zero knowledge and deals with verifiable computations: "A worker can proof that they multiplied three publicly known numbers $a,b,x \in \mathbb{F}_{13}$ and that the result is $z \in \mathbb{F}_{13}$, in such a way that no verifier has to repeat the computation."*

### 7.1.1 Formal Languages

Roughly speaking a formal language is nothing but a set of words, that are strings of letters taken from some alphabet and formed according to some defining rules of that language.

In computer science, formal languages are used for defining the grammar of programming languages in which the words of the language represent concepts that are associated with particular meanings or semantics. In computational complexity theory, decision problems are typically defined as formal languages, and complexity classes are defined as the sets of the formal languages that can be parsed by machines with limited computational power.

**Definition 7.1.1.1** (Formal Language)**.** *Let $\Sigma$ be a set and $\Sigma^*$ the set of all finite strings of elements from $\Sigma$. Then a **formal language** $L$ is a subset of $\Sigma^*$. The set $\Sigma$ is called the **alphabet** of $L$ and elements from $L$ are called **words**. The rules that specify which strings from $\Sigma^*$ belong to $L$ are called the **grammar** of $L$.*

*In the context of proofing systems we often call words **statements**.*

**Example 70** (Generalized factorization snark)**.** *Consider example 69 again. Definition 7.1.1.1 is not quite suitable yet to define the example, since there is not distinction between public input and private input.*

*However if we assume for the moment that the task in example 69 is to simply find $a,b,x,y \in \mathbb{F}_{13}$ such that that $x \cdot y \cdot a \cdot = b$, then we can define the entire solution set as a language $L_{factor}$ over the alphabet $\Sigma = \mathbb{F}_{13}$. We then say that a string $w \in \Sigma^*$ is a statement in our language $L_{factor}$ if and only if $w$ consists of 4 letters $w_1, w_2, w_3, w_4$ that satisfy the equation $w_1 \cdot w_2 \cdot w_3 = w_4$.*

**Example 71** (Binary strings)**.** *If we take the set $\{0,1\}$ as our alphabet $\Sigma$ and imply no rules at all to form words in this set. Then our language $L$ is the set $\{0,1\}^*$ of all finite binary strings. So for example $(0,0,1,0,1,0,1,1,0)$ is a word in this language.*

**Example 72** (Programing Language)**.**

**Example 73** (Compiler)**.**

As we have seen in general not all strings from an alphabet are words in a language. So an important question is, weather a given string belongs to a language or not.

**Definition 7.1.1.2** (Relation, Statement, Instance and Witness)**.** *Let $\Sigma_I$ and $\Sigma_W$ be two alphabets. Then the binary relation $R \subset \Sigma_I^* \times \Sigma_W^*$ is called a **checking relation** for the language*

$$L_R := \{(i,w) \in \Sigma_I^* \times \Sigma_W^* \, | R(i,w) \, \}$$

*of all **instances** $i \in \Sigma_I^*$ and **witnesses** $i \in \Sigma_I^*$, such that the **statement** $(i,w)$ satisfies the checking relation.*

**Remark 3.** *To summarize the definition, a statement is nothing but a membership claim of the form $x \in L$. So statements are really nothing but strings in an alphabet that adhere to the rules of a language.*

*However in the context of checking relations, there is another interpretations in terms of a knowledge claim of the form "In the scope of relation R, I know a witness for instance x." This is of particular importance in the context of zero knowledge proofing systems, where the instance represents public knowledge, while the witness represents the data that is hidden (the zero-knowledge part).*

*For some cases, the knowledge and membership types of statements can be informally considered interchangeable, but formally there are technical reasons to distinguish between the two notions (See for example XXX )*

**Example 74** (Generalized factorization snark)**.** *Consider example 69 and our associate formal language 70. We can define another language $L_{zk-factor}$ for that example by defining the alphabet $\Sigma_I \times \Sigma_W$ to be $\mathbb{F}_{13} \times \mathbb{F}_{13}$ and the checking relation $R_{zk-factor}$ such that $R(i,w)$ holds if and only if instance $i$ is a two letter string $i = (a,b)$ and witness $w$ is a two letter string $w = (x,y)$, such that the equation $x \cdot y \cdot a = b$ holds.*

*So to summarize four elements $x,y,a,b \in \mathbb{F}_{13}$ form a statement $((x,y),(a,b))$ in $L_{zk-factor}$ with instance $(a,b)$ and witness $x,y$, precisely if, given a and b, the values x and y are a solution to the generalized factorization problem $x \cdot y \cdot a = b$.*

**Example 75** (SHA256 relation)**.** *ssss*

As the following example shows checking relations and their languages are quite general and able to express in particular the class of all terminating computer programs:

**Example 76** (Computer Program)**.** *Let A be a terminating algorithm that transforms a binary string of inputs in finite execution steps into a binary output string. We can then interpret A as a map*

$$A : \{0,1\}^* \to \{0,1\}^*$$

*Algorithm A then defines a relation* $R \subset \{0,1\}^* \times \{0,1\}^*$ *in the following way: instance string* $i \in \{0,1\}^*$ *and witness string* $w \in \{0,1\}^*$ *satisfy the relation R, that is* $R(i,w)$*, if and only if w is the result of algorithm A executed on input instance i.*

### 7.1.2 Circuits

**Definition 7.1.2.1** (Circuits)**.** *Let* $\Sigma_I$ *and* $\Sigma_W$ *be two alphabets. Then a directed, acyclic graph C is called a **circuit** over* $\Sigma_I \times \Sigma_W$*, if the graph has an ordering and every node has a label in the following way:*

- *Every source node (called input) has a letter from* $\Sigma_I \times \Sigma_W$ *as label.*
- *Every sink node (called output) has a letter from* $\Sigma_I \times \Sigma_W$ *as label.*
- *Every other node (called gate) with j incoming edges has a label that consist of a function* $f : (\Sigma_I \times \Sigma_W)^j \to \Sigma_I \times \Sigma_W$*.*

**Remark 4** (Circuit-SAT)**.** *Every circuit with n input nodes and m output nodes can be seen a function that transforms strings of size n from* $\Sigma_I \times \Sigma_W$ *into strings of size m over the same alphabet. The transformation is done by sending the strings from a node along the outgoing edges to other nodes. If those nodes are gates, then the string is transformed according to the label.*

*By executing the previous transformation, every node of a circuit has an associated letter from* $\Sigma_I \times \Sigma_W$ *and this defines a checking relation over* $\Sigma_I^* \times \Sigma_W^*$*. To be more precise, let C be a circuit with n nodes and* $(i,w) \in \Sigma_I^j \times \Sigma_W^k$ *a string. Then* $R_C(i,w)$ *iff THE CIRCUIT IS SATISFIED WHEN ALL LABELS ARE ASSOCIATED TO ALL NODES IN THE CIRCUIT.... BUT MORE PRECISE*

*MODULO ERRORS. TO BE CONTINUED.....*

*An Assignment associates field elements to all edges (indices) in an algebraic circuit. An Assignment is valid, if the field element arise from executing the circuit. Every other assignment is invalid.*

*The checking relation for circuit-SAT then is satidfied if valid asignment (TODO: THE WITNESS/INSTANCE SPLITTING)*

*Valid assignments are proofs for proper circuit execution.*

So to summarize, algebraic circuits (over a field $\mathbb{F}$) are directed acyclic graphs, that express arbitrary, but bounded computation. Vertices with only outgoing edges (leafs, sources) represent inputs to the computation, vertices with only ingoing edges (roots, sinks) represent outputs from the computation and internal vertices represent field operations (Either addition or multiplication). It should be noted however that there are many circuits that can represent the same laguage...

Circuits have a notion of execution, where input values are send from leafs along edges, through internal vertices to roots.

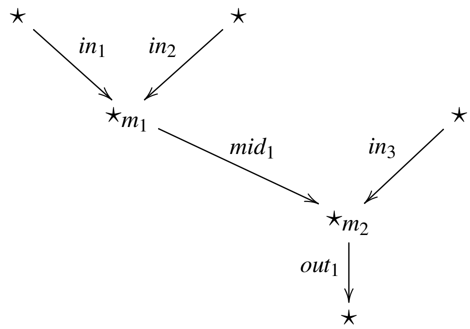
**Remark 5.** *Algebraic circuits are usually derived by Compilers, that transform higher languages to circuits. An example of such a compiler is XXX. Note: Different Compiler give very different circuit representations and Compiler optimization is important.*

**Example 77** (Generalized factorization snark)**.** *Consider our generalized factorization example 69 with associated language 74.*

*To write this example in circuit-SAT, consider the following function*

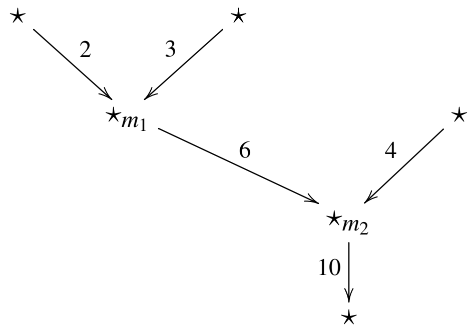
$$f:\mathbb{F}_{13}\times\mathbb{F}_{13}\times\mathbb{F}_{13}\to\mathbb{F}_{13};(x_1,x_2,x_3)\mapsto(x_1\cdot x_2)\cdot x_3$$

*A valid circuit for* $f:\mathbb{F}_{11}\times\mathbb{F}_{11}\times\mathbb{F}_{11}\to\mathbb{F}_{11};(x_1,x_2,x_3)\mapsto(x_1\cdot x_2)\cdot x_3$ *is given by:*



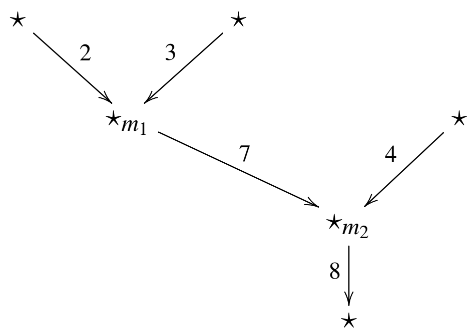

*with edge-index set* $I:=\{in_1,in_2,in_3,mid_1,out_1\}$.

*To given a valid assignment, consider the set* $I_{valid}:=\{in_1,in_2,in_3,mid_1,out_1\}=\{2,3,4,6,10\}$




*Appears from multiplying the input values at* $m_1$*,* $m_2$ *in* $\mathbb{F}_{13}$*, hence by executing the circuit.*

*Non valid assignment:* $I_{err}:=\{in_1,in_2,in_3,mid_1,out_1\}=\{2,3,4,7,8\}$




*Can not appear from multiplying the input values at* $m_1$*,* $m_2$ *in* $\mathbb{F}_{13}$

*To match the requirements of the inital task 69, we have to split the statement into instance and witness. So given index set* $I:=\{in_1,in_2,in_3,mid_1,out_1\}$*, we assume that every step in the computation other then* $in_3$ *and* $out_1$ *are part of the witness. So we choose:*

- *Instance* $S = \{in_3, out_1\}$.
- *Witness* $W = \{in_1, in_2, mid_1\}$.

**Example 78** (Baby JubJub for BLS6-6)**.**

**Example 79** (ECDH as a circuit)**.** *over BLS6*

**Example 80** (BLS Signature)**.** *example of one layer recursion over MNT4 and MNT6*

**Example 81** (Boolean Circuits)**.**

**Example 82** (Algebraic (Aithmetic) Circuits)**.**

Any program can be reduced to an arithmetic circuit (a circuit that contains only addition and multiplication gates). A particular reduction can be found for example in [BSCG+13]

### 7.1.3 Rank-1 Constraint Systems

**Definition 7.1.3.1** (Rank-1 Constraint system)**.** *Let $\mathbb{F}$ be a Galois field, $i, j, k$ three numbers and A, B and C three $(i+j+1) \times k$ matrices with coefficients in $\mathbb{F}$. Then any vector $x = (1, \phi, w) \in \mathbb{F}^{1+i+j}$ that satisfies the* ***rank-1 constraint system*** *(R1CS)*

$$Ax \odot Bx = Cx$$

*(where $\odot$ is the Hadamard/Schur product) is called a* ***statement*** *of that system, with* ***instance*** *$\phi$ and* ***witness*** *$w$.*

*We call $k$ the* ***number of constraints****, $i$ the* ***instance*** *size and $j$ the* ***witness*** *size.*

**Remark 6.** *Any Rank-1 constraint system defines a formal language in the following way: Consider the alphabets $\Sigma_I := \mathbb{F}$ and $\Sigma_W : \mathbb{F}$. Then a checking relation $R_{R1CS} \subset \Sigma_I^i \times \Sigma_W^j \subset \Sigma_I^* \times \Sigma_W^*$ is defined by*

$$R_{R1CS}(i, w) \Leftrightarrow (i, w) \text{ satisfies the R1CS}$$

*As shown in XXX such a checking relation defines a formal language. We call this language* ***R1CS satisfiability****.*

**Example 83** (Generalized factorization snark)**.** *Defining the 5-dimensional affine vector $w = (1, in_1, in_2, in_3, m_1, out_1)$ for $in_1, in_2, in_3, m_1, out_1 \in \mathbb{F}_{13}$ and the $6\times$?-matrices*

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

*We can instantiate the general R1CS equation $Aw \odot Bw = Cw$ as*

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ in_1 \\ in_2 \\ in_3 \\ m_1 \\ out_1 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ in_1 \\ in_2 \\ in_3 \\ m_1 \\ out_1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ in_1 \\ in_2 \\ in_3 \\ m_1 \\ out_1 \end{pmatrix}$$

*So evaluating all three matrix products and the Hadarmat prodoct we get two constraint equations*

$$\begin{aligned} in_1 \cdot in_2 &= m_1 \\ m_1 \cdot in_3 &= out_1 \end{aligned}$$

So from the way this R1CS is constructed, we know that whatever the underlying field $\mathbb{F}$ is, the only solutions to this equations are

$$\{(0,0,0), (0,1,0), (1,0,0), (1,1,1)\}$$

**Gadgets** Rank 1 contraints systems can become very large ....

### Boolean Algebra

Sometimes it is necessary to assume that a statement describes boolean variables. However by definition the alphabet of a statement is a finite field, which is often the scalar field of a large prime order cyclic group. So developers need a way to simulate boolean algebra inside other finite fields.

The most common way to do this, is to interpret the additive and multiplicate neutral element $\{0,1\} \subset F$ as boolean values. This is convinient because they are defined in any field.

In what follows we will define a few of the most basic R1CS to check boolean expressions in R1CS satidfyability. We will leave other basic constructions as exercises to the reader.

We start with actually constraining field elements to boolean values then Once field elements are boolean constraint, we need constraints that are able to enforce boolean algebra on them. We therefore give constraints for the functionally complete set of Boolean operators give by $AND$ and $NOT$. As all other boolean operations can be constructed from $AND$ and $NOT$, this sufficies. However in actual implementations it is of high importance to limit the number of constraints as much as possible. In reality it is therefor advantageous to implement all logic operators in constraints.

**Boolean Constraint** So when a developer needs boolean variables as part of their statement, a R1CS is required on those variables, that enforces the variable to be either 1 or 0. So to "constrain a field element $x \in \mathbb{F}$ to be 1 or 0 what we need is a system of equation $(A_i x)\cdot(B_i x) = C_i x$ for some $A_i, B_i, C_i \in \mathbb{F}$, such that the only possible solutions for $x$ are 0 or 1. As it turns out such a system can be realized by a single equation $x\cdot(1-x) = 0$ We see that indeed 0 and 1 are the only solutions here, since for the right side to be zero, at least one factor on the left side needs to be zero and this only happens for 0 and 1.

So now that we have found a correct equation for a boolean constrain, we have to translate it into the associated R1CS format, which is given by

$$\begin{pmatrix}0 & 1\end{pmatrix}\begin{pmatrix}1\\ x\end{pmatrix} \odot \begin{pmatrix}1 & -1\end{pmatrix}\begin{pmatrix}1\\ x\end{pmatrix} = \begin{pmatrix}0 & 0\end{pmatrix}\begin{pmatrix}1\\ x\end{pmatrix}$$

So we get the following statement $\phi = (1,i,w) = (1,x)$, with instance (public input) $i = x$ and now witness (private input) $w$. In addition we get the matrices $A = \begin{pmatrix}0 & 1\end{pmatrix}$, $B = \begin{pmatrix}1 & -1\end{pmatrix}$ and $C = \begin{pmatrix}0 & 0\end{pmatrix}$.

To make those constraints easily accesable for R1CS developers, a gadget is convinient:

**AND-constraints** Given three field elements $x,y,z \in \mathbb{F}$ that represent boolean variables, we want to find a R1CS, such that $w = (1,x,y,z)$ satisfies the constraint system if and only if $x\,AND\,y = z$.

So first we have to constrain $x$, $y$ and $z$ to be boolean as explained in XXX. The next thin is we need to find a R1CS that enforces the $AND$ logic. We can simply choose $x\cdot y = z$, since (for boolean constraint values) $x\cdot y$ equals 1 if and only if both $x$ and $y$ are 1.

Now that we have found a correct equation for a boolean constrain, we have to translate it into the associated R1CS format, which is given by

$$\begin{pmatrix}0 & 1 & 0 & 0\end{pmatrix}\begin{pmatrix}1\\x\\y\\z\end{pmatrix}\odot\begin{pmatrix}0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}1\\x\\y\\z\end{pmatrix}=\begin{pmatrix}0 & 0 & 0 & 1\end{pmatrix}\begin{pmatrix}1\\x\\y\\z\end{pmatrix}$$

Combining this R1CS with the required fthree boolean constraints for $x$, $y$ and $z$ we get

$$\begin{pmatrix}0 & 1 & 0 & 0\\ \hline 0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{pmatrix}\begin{pmatrix}1\\x\\y\\z\end{pmatrix}\odot\begin{pmatrix}0 & 0 & 1 & 0\\ \hline 1 & -1 & 0 & 0\\1 & 0 & -1 & 0\\1 & 0 & 0 & -1\end{pmatrix}\begin{pmatrix}1\\x\\y\\z\end{pmatrix}=\begin{pmatrix}0 & 0 & 0 & 1\\ \hline 0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{pmatrix}\begin{pmatrix}1\\x\\y\\z\end{pmatrix}$$

So from the way this R1CS is constructed, we know that whatever the underlying field $\mathbb{F}$ is, the only solutions to this equations are

$$\{(0,0,0),(0,1,0),(1,0,0),(1,1,1)\}$$

which is the set of all $(x,y,z)\in\{0,1\}^3$ such that $x\,AND\,y=z$.

**NOT constraint** Given two field elements $x,y\in\mathbb{F}$ that represent boolean variables, we want to find a R1CS, such that $w=(1,x,y)$ satisfies the constraint system if and only if $x=\neg y$.

So again we have to constrain $x$ and $y$ to be boolean as explained in XXX. The next think is we need to find a R1CS that enforces the *NOT* logic. We can simply choose $(1-x)=y$, since (for boolean constraint values) this enforces that $y$ is always the boolean opposite of $x$.

Now that we have found a correct equation for a boolean constrain, we have to translate it into the associated R1CS format, which is given by

$$\begin{pmatrix}1 & -1 & 0\end{pmatrix}\begin{pmatrix}1\\x\\y\end{pmatrix}\odot\begin{pmatrix}1 & 0 & 0\end{pmatrix}\begin{pmatrix}1\\x\\y\end{pmatrix}=\begin{pmatrix}0 & 0 & 1\end{pmatrix}\begin{pmatrix}1\\x\\y\end{pmatrix}$$

So actually we wrote the linear equation $1-x=y$ like $(1-x)\cdot 1=y$ and translated that into the matrix equation.

Combining this R1CS with the required fthree boolean constraints for $x$, $y$ and $z$ we get

$$\begin{pmatrix}1 & -1 & 0\\ \hline 0 & 1 & 0\\0 & 0 & 1\end{pmatrix}\begin{pmatrix}1\\x\\y\end{pmatrix}\odot\begin{pmatrix}1 & 0 & 0\\ \hline 1 & -1 & 0\\1 & 0 & -1\end{pmatrix}\begin{pmatrix}1\\x\\y\end{pmatrix}=\begin{pmatrix}0 & 0 & 1\\ \hline 0 & 0 & 0\\0 & 0 & 0\end{pmatrix}\begin{pmatrix}1\\x\\y\end{pmatrix}$$

So from the way this R1CS is constructed, we know that whatever the underlying field $\mathbb{F}$ is, the only solutions to this equations are

$$\{(0,1),(1,0)\}$$

which is the set of all $(x,y)\in\{0,1\}^2$ such that $x=\neg y$.

EXERCISE: DO OR; XOR; NAND

More complicated logical constraints can then be optained by combining all sub-R1CS together. For example if the task is to enforce $(in_1\,AND\neg in_2)ANDin_3=out_1$ we first apply the FLATTENING technique from XXX, which gives is

$$\begin{array}{rcl}\neg in_2 & = & mid_1\\ in_1\,AND\,mid_1 & = & mid_2\\ mid_2\,AND\,in_3 & = & out_1\end{array}$$

So we have the statement $w=(1,in_1,in_2,in_3,mid_1,mid_2,out_1)$, 6 boolean constraints for the variables, 2 constraints for the 2 *AND* operations and 1 constraint for the *NOT* operation.

### Binary representations

In circuit computations its is often necessary to use the binary representation of a prime field element. Binary representations of prime field elements work execactly like binary representations of ordinary unsigned integers. Only the algebraic operations are different. To compute the binary representation of some number $x \in \mathbb{F}_p$ we need to know the number of bits in the binary representation of $p$ first. We write this as $m = |p_{bin}|$.

Then a bitstring $(b_0, \ldots, b_m) \in \{0,1\}^m$ is the binary representation of the field element $x$, if and only if

$$x = b_0 \cdot 2^0 + b_1 \cdot 2^1 + \ldots + b_m \cdot 2^m$$

Note that, since $p$ is a prime number that has a leading bit 1 at position $m$. Moreover every prime number $p > 2$ is odd and hence has least significat bit set to 1. Hence all numbers $2^j$ for $0 \leq j \leq m$ are elements of $\mathbb{F}_p$ and the equation is well defined. We can therefore enforce this equation as a R1CS, by flattening the equation:

$$\begin{array}{lcl}
b_0 \cdot 1 & = & mid_0 \\
b_1 \cdot 2 & = & mid_1 \\
\cdots & = & \cdots \\
b_m \cdot 2^m & = & mid_m \\
(mid_0 + mid_1 + \ldots + mid_m) \cdot 1 & = & x
\end{array}$$

So we have the statement $w = (1, x, b_0, \ldots, b_m, mid_0, \ldots, mid_m)$ and we need $(m+1)$ constraints to enforce the binary representation in addition to the $m$ constraints that enforce booleanness.

At this point we see, that writing more complex R1CS becomes clumbsy and in actual implementations people therefore use languages to makes the constraint system more readable. In this example we could write for example something like this:

keeping in mind that this is a meta level algorith to **generate** the R1CS, not the R1CS itself, as constructs like for loops have not direct meening on the level of the R1CS itself.

**Example 84.** *Considering the prime field $\mathbb{F}_{13}$, we want to enforce the binary representation of $7 \in \mathbb{F}_{13}$. To find the number of bits that we need to consider in our R1Cs, we start with the binary representation of 13, which is $(1,0,1,1)$ since $13 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3$. So $m = 4$ and we have to enforce a 4-bit representation for 7, which is $(1,1,1,0)$, since $7 = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3$.*

*A valid statement is then given by $w = (1,7,1,1,1,0,1,2,4,0)$ and indeed we satify the 9 required constraints*

$$\begin{array}{lcll}
1 \cdot (1-1) & = & 0 & \textit{// boolean constraints} \\
1 \cdot (1-1) & = & 0 & \\
1 \cdot (1-1) & = & 0 & \\
0 \cdot (1-0) & = & 0 & \\
 & & & \\
1 \cdot 1 & = & 1 & \\
1 \cdot 2 & = & 2 & \\
1 \cdot 4 & = & 4 & \\
0 \cdot 8 & = & 0 & \\
(1+2+4+0) \cdot 1 & = & 7 &
\end{array}$$

### Conditional (ternary) operator

It is often required to implement the terniary conditional operator ? : as a R1CS. In general this operator takes three arguments, a boolean value $b$ and two expressions $if\_true$ and $if\_false$, usually written as $b$ ? $c$ : $d$ and executes $c$ and $d$ according to the value of $b$.

If we assume all three arguments to be values from a finite field, such that $b$ is boolean constraint (XXX), we can enforce a field element $x$ to be the result of the conditional operator as

$$x = b \cdot c + (1-b) \cdot d$$

Flattening the code gives

$$\begin{array}{lcl} b \cdot c & = & mid_0 \\ (1-b) \cdot d & = & mid_1 \\ (mid_0 + mid_1) \cdot 1 & = & x \end{array}$$

So we have the statement $w = (1, x, b, c, d, mid_0, mid_1)$ and we need 3 constraints to enforce the conditional operator in addition to 1 constraint that enforces booleanness of $b$.

NOTE: THERE WAS THIS PODCAST WITH ANNA AND THE GUY JAN TALKE TO WHERE HE SAID; CONDITIONALS CAN BE IMPLEMENTED SUCH THAT NOT BOTH BRANCHES ARE EXCUTED: LOOK THAT UP

### Range Proofs

$x > 5$...

### UintN

STUFF ABOUT HOW UINTN COMPUTATIONS ARE NOT STANDARDIZED AND THAT THERE ARE IMPLEMENTATIONS OTHER THEN MOD-N.... WE FIX ON MOD-N. WHAT DO ZEXE CIRCOM ECT FIX ON?

As we know circuits are not defined over integers but over finite fields instead. We therefore have no notation of integers in circuits. However on computers we also not use integers natively but Uint's instead.

As we know a UintN type is a representation of integers in the range of $0 \ldots 2^N$ with the exception that algebraic operations like addition and multiplication deviate from actual integers, whenever the result exceeds the largest representable number $2^N - 1$.

In circuit design it is therefore important to distinguish between various things tht might look like integers, but are actually not. For example Haskells type NAT is an actual implementation of natural numbers. In particular this means ....

**Example 85** (Uint8)**.** *What is* $0xFFF0 + 0xFFF0$ *and so on...*

**Bit constraints** In prime fields, addition and multiplication behaves exactly like addition and multiplication with integers as long as the result does not exceed the modulus.

This makes the representation of UintNs in a prime field $\mathbb{F}_p$ potentially ambigious, as there are two possible representations, whenever $2^N - 1 < p$. In that case any element of $UintN$ could be interpreted as an element of $\mathbb{F}_p$. This however is dangerous as the algebraic laws like addition and multiplication behave very different in general.

It is therefore common to represent UintN types in circuits as binary constraints strings of field elements of length $N$.

**Example 86.** *Consider the Uint4 type over the prime field $\mathbb{F}_{17}$. Since $2^4 = 16$, Uint4 can represent the numbers $0,\ldots,15$ and it would be possible to interpret them as elements in $\mathbb{F}_{17}$. However addition*

### Twisted Edwards curves

Sometimes it required to do elliptic curve cryptography "inside of a circuit". This means that we have to implement the algebraic operations (addition, scalar multiplication) of an elliptic curve as a R1CS. To do this efficiently the curve that we want to implement must be defined over the same base field as the field that is used in the R1CS.

**Example 87.** *So for example when we consider an R1CS over the field $\mathbb{F}_{13}$ as we did in example XXX, then we need a curve that is also defined over $\mathbb{F}_{13}$. Moreover it is advantegous to use a (twisted) Edwards curve inside a circuit, as the addition law contains no branching (See XXX). As we have seen in XXX our Baby-Jubjub curve is an Edwards curve defined over $\mathbb{F}_{13}$. So it is well suited for elliptic curve cryptography in our pend and paper examples*

**Twisted Edwards curves constraints** As we have seen in XXX, an Edwards curve over a finite field $F$ is the set of all pairs of points $(x,y) \in \mathbb{F}\times\mathbb{F}$, such that $x$ and $y$ satisfy the equation $a\cdot x^2 + y^2 = 1 + d\cdot x^2y^2$.

We can interpret this equation as a constraint on $x$ and $y$ and rewrite it as a R1CS by applying the flattenin technique from XXX.

$$\begin{aligned} x\cdot x &= x\_sq \\ y\cdot y &= y\_sq \\ x\_sq\cdot y\_sq &= xy\_sq \\ (a\cdot x\_sq + y\_sq)\cdot 1 &= 1 + d\cdot xy\_sq \end{aligned}$$

So we have the statement $w = (1, x, y, x\_sq, y\_sq, xy\_sq)$ and we need 4 constraints to enforce that $x$ and $y$ are points on the Edwards curve $x^2 + y^2 = 1 + d\cdot x^2y^2$. Writing the constraint system in matrix form, we get:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & a & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ x \\ y \\ x\_sq \\ y\_sq \\ xy\_sq \end{pmatrix} \odot \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ x \\ y \\ x\_sq \\ y\_sq \\ xy\_sq \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & d \end{pmatrix} \begin{pmatrix} 1 \\ x \\ y \\ x\_sq \\ y\_sq \\ xy\_sq \end{pmatrix}$$

EXERCISE: WRITE THE R1CS FOR WEIERSTRASS CURVE POINTS

**Example 88** (Baby-JubJub)**.** *Considering our pen and paper Baby JubJub curve over from XXX, we know that the curve is defined over $\mathbb{F}_{13}$ and that $(11,9)$ is a curve point, while $(2,3)$ is not a curve point.*

*Starting with $(11,9)$, we can compute the statement $w = (1,11,9,4,3,12)$. Substituting this into the constraints we get*

$$\begin{aligned} 11\cdot 11 &= 4 \\ 9\cdot 9 &= 3 \\ 4\cdot 3 &= 12 \\ (1\cdot 4 + 3)\cdot 1 &= 1 + 7\cdot 12 \end{aligned}$$

*which is true in $\mathbb{F}_{13}$. So our statement is indeed a valid assignment to the twisted Edwards curve constraining system.*

*Now considering the non valid point $(2,3)$, we can still come up with some kind of statement $w$ that will satisfy some of the constraints. But fixing $x = 2$ and $y = 3$, we can never satisfy all constraints. For example $w = (1,2,3,4,9,10)$ will satisfy the first three constraints, but the last constrain can not be satisfied. Or $w = (1,2,3,4,3,12)$ will satisfy the first and the last constrain, but not the others.*

**Twisted Edwards curves addition** As we have seen in XXX one the major advantages of working with (twisted) Edwards curves is the existence of an addition law, that contains no branching and is valid for all curve points. Moreover the neutral element is not "at infinity" but the actual curve poin $(0,1)$.

As we know from XXX, give two points $(x_1,y_1)$ and $(x_2,y_2)$ on a twisted Edwards curve their sum is given by

$$(x_3,y_3) = \left(\frac{x_1y_2+y_1x_2}{1+d\cdot x_1x_2y_1y_2}, \frac{y_1y_2-a\cdot x_1x_2}{1-d\cdot x_1x_2y_1y_2}\right)$$

We can realize this equation as a R1CS as follows: First not that we can rewrite the addition law as

$$\begin{array}{lcl}
x_1\cdot x_2 & = & x_{12}\\
y_1\cdot y_2 & = & y_{12}\\
x_1\cdot y_2 & = & xy_{12}\\
y_1\cdot x_2 & = & yx_{12}\\
x_{12}\cdot y_{12} & = & xy_{1212}\\
x_3\cdot(1+d\cdot xy_{1212}) & = & xy_{12}+yx_{12}\\
y_3\cdot(1-d\cdot xy_{1212}) & = & y_{12}-a\cdot x_{12}
\end{array}$$

So we have the statement $w = (1,x_1,y_1,x_2,y_2,x_3,y_3,x_{12},y_{12},xy_{12},yx_{12},xy_{1212})$ and we need 7 constraints to enforce that $(x_1,y_1)+(x_2,y_2) = (x_3,y_3)$

**Example 89** (Baby-JubJub)**.** *Considering our pen and paper Baby JubJub curve over from XXX. We recall from XXX that $(11,9)$ is a generator for the large prime order subgroup. We therefor already know from XXX that $(11,9)+(7,8) = (11,9)+[3](11,9) = [4](11,9) = (2,9)$. So we compute a valid statement as $w = (1,11,9,7,8,2,9,12,7,10,11,6)$. Indeed*

$$\begin{array}{lcl}
11\cdot 7 & = & 12\\
9\cdot 8 & = & 7\\
11\cdot 8 & = & 10\\
9\cdot 7 & = & 11\\
10\cdot 11 & = & 6\\
2\cdot(1+7\cdot 6) & = & 10+11\\
9\cdot(1-7\cdot 6) & = & 7-1\cdot 12
\end{array}$$

There are optimizations for this using only 6 constraints, available:

**Twisted Edwards curves inversion** Similar to elliptic curves in Weierstrass form, inversion is cheap on Edwards curve as the negative of a curve point $-(x,y)$ is given by $(-x,y)$. So a curve point $(x_2,y_2)$ is the additive inverse of another curve point $(x_1,y_1)$ precisely if the equation $(x_1,y_1) = (-x_2,y_2)$ holds. We can write this as

$$\begin{array}{lcl}
x_1\cdot 1 & = & -x_2\\
y_1\cdot 1 & = & y_2
\end{array}$$

We therefor have a statement of the form $w = (1, x_1, y_1, x_2, y_2)$ and can write the constraints into a matrix equation as

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ y_1 \\ x_2 \\ y_2 \end{pmatrix} \odot \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ y_1 \\ x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ y_1 \\ x_2 \\ y_2 \end{pmatrix}$$

In addition we need the following constraints:

$$\begin{aligned} x_1 \cdot 1 &= -x_2 \\ y_1 \cdot 1 &= y_2 \end{aligned}$$

**Twisted Edwards curves scalar multiplication** Although there are highly optimzed R1CS implementations for scal multiplication on elliptic curves, the basic idea is somewhat simple: Given an elliptic curve $E/\mathbb{F}_r$, a scalar $x \in \mathbb{F}_r$ with binary representation $(b_0, \ldots, b_m)$ and a curve point $P \in E/\mathbb{F}_r$, the scalar multiplication $[x]P$ can be written as

$$[x]P = [b_0]P + [b_1]([2]P) + [b_2]([4]P) + \ldots + [b_m]([2^m]P)$$

and since $b_j$ is either 0 or 1, $[b_j](kP)$ is either the neutral element of the curve or $[2^j]P$. However $[2^j]P$ can be computed inductively by curve point doubling, since $[2^j]P = [2]([2^{j-1}]P)$.

So scalar multiplication can be reduced to a loop of length $m$, where the original curve point is repeadedly douled and added to the result, whenever the appropriate bit in the scalar is equal to one.

So to enforce that a curve point $(x_2, y_2)$ is the scalar product $[k](x_1, y_1)$ of a scalar $x \in F_r$ and a curve point $(x_1, y_1)$, we need an R1CS the defines point doubling on the curve (XXX) and an R1CS that enforces the binary representation of $x$ (XXX).

In case of twisted Edwards curve, we can use ordinary addition for doubling, as the constraints works for both cases (doublin is addition, where both arguments are equal). Moreover $[b](x, y) = (b \cdot x, b \cdot y)$ for boolean $b$. Hence flattening equation XXX gives

$$\begin{aligned} b_0 \cdot x_1 &= x_{0,1} \quad //[b_0]P \\ b_0 \cdot y_1 &= y_{0,1} \end{aligned}$$

In addition we need to constrain $(b_0, \ldots, b_N)$ to be the binary representation of $x$ and we need to constrain each $b_j$ to be boolean.

As we can see a R1CS for scalar multiplication utilizes many R1CS that we have introduced before. For efficiency and readability it is therefore useful to apply the concept of a gadget (XXX). A pseudocode method to derive the associated R1CS could look like this:

**Curve Cycles** A particulary interesting case with far reaching implication is the situation when we have two curve $E_1$ and $E_2$, such that the scalar field of curve $E_1$ is the base field of curve $E_2$ and vice versa. In that case it is possible to implement the group laws of one curve in circuits defined over the scalar field of the other curve.

### The RAM Model

FROM THE PODCAST WITH ANNA R. AND THE GUY FROM JAN....

**Generalizations**

many circuits can be found here:

### 7.1.4 Quadratic Arithmetic Programs

As shown by [Pinocchio] rank-1 constraint systems can be transformed into so called quadratic arithmetic programs assuming $\mathbb{F}$.

taken from the pinocchio paper. For proving arithmetic circuit-sat. Given a R1CS QAPs transform potential solution vectors into two polynomials $p$ and $t$, such that $p$ is divisible by $t$ if and only if the vector is a solution to the R1CS.

They are major building blocks for **succinct** proofs, since with high probability, the divisibility check can be performed in a single point of those polynomials. So computationally expensive polynomial division check is reduced TO WHAT? (IN FIELDS THERE IS ALWAYS DIVISIBILITY)

**Definition 7.1.4.1** (Quadratic Arithmetic Program)**.** *Assume we have a Galois field $\mathbb{F}$, three numbers $i, j, k$ as well as three $(i+j+1)\times k$ matrices $A$, $B$ and $C$ with coefficients in $\mathbb{F}$ that define the R1CS $Ax \odot Bx = Cx$ for some statement $x = (1, i, w)$ and let $m_1, \ldots, m_k \in \mathbb{F}$ be arbitrary field elements.*

*Then a* ***quadratic arithmetic program*** *of the R1CS is the following set of polynomials over $\mathbb{F}$*

$$QAP = \left\{ t \in \mathbb{F}[x], \{a_h, b_h, c_h \in \mathbb{F}[x]\}_{h=1}^{i+j+1} \right\}$$

*where $t(x) := \Pi_{l=1}^{k}(x - m_l)$ is a polynomial f degree $k$, called the* ***target polynomial*** *of the QAP and $a_h(x)$, $b_h(x)$ as well as $c_h(x)$ are the unique degree $k-1$ polynomials that are defined by the equations*

$$a_h(m_l) = A_{h,l} \quad b_h(m_l) = B_{h,l} \quad c_h(m_l) = C_{h,l} \quad h = 1, \ldots, i+j+1, l = 1, \ldots, k$$

The major point is that R1CS-sat can be reformulated into the divisibility of a polynomials defined by any QAP.

**Theorem 7.1.4.2.** *Assume that an R1CS and an associated QAP as defined in XXX are given. Then the affine vector $y = (1, i, w)$ is a solution to the R1CS, if and only if the polynomial*

$$p(x) = \left(\sum y_h \cdot a_h(x)\right) \cdot \left(\sum y_h \cdot b_h(x)\right) - \sum y_h \cdot c_h(x)$$

*is divisible by the target polynomial $t$.*

The polynomials $a_h$, $b_h$ and $c_h$ are uniquely defined by the equations in XXX. However to actually compute them we need some algorithm like the Langrange XXX from XXX.

**Example 90** (Generalized factorization snark)**.** *In this example we want to transform the R1CS from example 74 into an associated QAP.*

*We start by choosing an arbitrary field element for every constraint in the R1CS, since we have 2 constraints we choose $m_1 = 5$ and $m_2 = 7$*

*With this choice we get the target polynomial $t(x) = (x-m_1)(x-m_2) = (x-5)(x-7) = (x+8)(x+6) = x^2+x+9$.*

*Since our statement has structure $w = (1, in_1, in_2, in_3, m_1, out_1)$ we have to compute the following degree 1 polynomials*

$\{a_c, a_{in_1}, a_{in_2}, a_{in_3}, a_{mid_1}, a_{out}\}$ $\{b_c, b_{in_1}, b_{in_2}, b_{in_3}, b_{mid_1}, b_{out}\}$ $\{c_c, c_{in_1}, c_{in_2}, c_{in_3}, c_{mid_1}, c_{out}\}$

*Apply QAP rule XXX to the $a_{k\in I}$ polynomials gives*

$$\begin{array}{llllll} a_c(5)=0, & a_{in_1}(5)=1, & a_{in_2}(5)=0, & a_{in_3}(5)=0, & a_{mid_1}(5)=0, & a_{out}(5)=0 \\ a_c(7)=0, & a_{in_1}(7)=0, & a_{in_2}(7)=0, & a_{in_3}(7)=0, & a_{mid_1}(7)=1, & a_{out}(7)=0 \end{array}$$

$$\begin{array}{llllll} b_c(5)=0, & b_{in_1}(5)=0, & b_{in_2}(5)=1, & b_{in_3}(5)=0, & b_{mid_1}(5)=0, & b_{out}(5)=0 \\ b_c(7)=0, & b_{in_1}(7)=0, & b_{in_2}(7)=0, & b_{in_3}(7)=1, & b_{mid_1}(7)=0, & b_{out}(7)=0 \end{array}$$

$$\begin{array}{llllll} c_c(5)=0, & c_{in_1}(5)=0, & c_{in_2}(5)=0, & c_{in_3}(5)=0, & c_{mid_1}(5)=1, & c_{out}(5)=0 \\ c_c(7)=0, & c_{in_1}(7)=0, & c_{in_2}(7)=0, & c_{in_3}(7)=0, & c_{mid_1}(7)=0, & c_{out}(7)=1 \end{array}$$

*Since our polynomials are of degree* $1$ *only we don't have to invoke Langrange method but can deduce the solutions right away.*

*Polynomials are defined on the two values* $5$ *and* $7$ *here. Linear Polynomial* $f(x)=m\cdot x+b$ *is fully determined by this. Derive the general equation:*

- $5m+b=f(5)$ *and* $7m+b=f(7)$
- $b=f(5)-5m$ *and* $b=f(7)-7m$
- $b=f(5)+8m$ *and* $b=f(7)+6m$
- $f(5)+8m=f(7)+6m$
- $8m-6m=f(7)-f(5)$
- $2m=f(7)+12f(5)$
- $7\cdot 2m=7(f(7)+12f(5))$
- $m=7(f(7)+12f(5))$
- 
- $b=f(5)+8m$
- $b=f(5)+8\cdot(7(f(7)+12f(5)))$
- $b=f(5)+4(f(7)+12f(5))$
- $b=f(5)+4f(7)+9f(5)$
- $b=10f(5)+4f(7)$

*Gives the general equation:* $f(x)=7(f(7)+12f(5))x+10f(5)+4f(7)$

*For* $a_{in_1}$ *the computation looks like this:*

- $a_{in_1}(x)=7(a_{in_1}(7)+12a_{in_1}(5))x+10a_{in_1}(5)+4a_{in_1}(7)=$
- $7(0+12\cdot 1)x+10\cdot 1+4\cdot 0=$
- $7\cdot 12x+10=$
- $6x+10$

- $a_{mid_1}(x) = 7(a_{mid_1}(7) + 12a_{mid_1}(5))x + 10a_{mid_1}(5) + 4a_{mid_1}(7) =$
- $7(1+12\cdot 0)x + 10\cdot 0 + 4\cdot 1 =$
- $7\cdot 1x + 4 =$
- $7x+4$

| | | |
|---|---|---|
| $a_c(x) = 0$ | $b_c(x) = 0$ | $c_c(x) = 0$ |
| $a_{in_1}(x) = 6x+10$ | $b_{in_1}(x) = 0$ | $c_{in_1}(x) = 0$ |
| $a_{in_2}(x) = 0$ | $b_{in_2}(x) = 6x+10$ | $c_{in_2}(x) = 0$ |
| $a_{in_3}(x) = 0$ | $b_{in_3}(x) = 7x+4$ | $c_{in_3}(x) = 0$ |
| $a_{mid_1}(x) = 7x+4$ | $b_{mid_1}(x) = 0$ | $c_{mid_1}(x) = 6x+10$ |
| $a_{out}(x) = 0$ | $b_{out}(x) = 0$ | $c_{out}(x) = 7x+4$ |

*This gives the quadratic arithmetic program for our generalized factorization snark as*

$QAP = \{x^2+x+9, \{0, 6x+10, 0, 0, 7x+4, 0\}, \{0, 0, 6x+10, 7x+4, 0, 0\}, \{0, 0, 0, 0, 6x+10, 7x+4\}\}$

*Now as we recall, the main point for using QAPs in snarks is the fact, that solutions to R1CS are in 1:1 correspondence to the divisibility of a polynomial p, constructed from a R1CS solution and the polynomials of the QAP and the target polynomial.*

*So lets see this in our example. We already know from example XXX, that* $I = \{1, 2, 3, 4, 6, 11\}$ *is a solution to the R1CS XXX of our problem. To see how this translates to polyinomial divisibility we compute the polynomial* $p_I$ *by*

$$
\begin{aligned}
p_I(x) = &(\sum_{h\in|I|} I_h\cdot a_h(x))\cdot(\sum_{h\in|I|} I_h\cdot b_h(x)) - (\sum_{h\in|I|} I_h\cdot c_h(x))\\
=&(2(6x+10)+6(7x+4))\cdot(3(6x+10)+4(7x+4)) - (6(6x+10)+11(7x+4))\\
=&((12x+7)+(3x+11))\cdot((5x+4)+(2x+3)) - ((10x+8)+(12x+5))\\
=&(2x+5)\cdot(7x+7) - (9x)\\
=&(x^2+2\cdot 7x+5\cdot 7x+5\cdot 7) - (9x)\\
=&(x^2+x+9x+9) - (9x)\\
=&x^2+x+9
\end{aligned}
$$

*And as we can see in this particular example* $p_I(x)$ *is equal to the target polynomial* $t(x)$ *and hence it is divisible by t with* $p/t = 1$.

*To give a counter example we already know from XXX that* $I = \{1, 2, 3, 4, 8, 2\}$ *is not a solution to our R1CS. To see how this translates to polyinomial divisibility we compute the polynomial* $p_I$ *by*

$$
\begin{aligned}
p_I(x) = &(\sum_{h\in|I|} I_h\cdot a_h(x))\cdot(\sum_{h\in|I|} I_h\cdot b_h(x)) - (\sum_{h\in|I|} I_h\cdot c_h(x))\\
=&(2(6x+10)+6(7x+4))\cdot(3(6x+10)+4(7x+4)) - (6(6x+10)+11(7x+4))\\
=&8x^2+11x+3
\end{aligned}
$$

*This polynomial is not divisible by the target polynomial t since Not divisible by t:* $(8x^2+11x+3)/(x^2+x+9) = 8 + \frac{3x+8}{x^2+x+9}$

### 7.1.5 Quadratic span programs

## 7.2 proof system

Now a *proof system* is nothing but a game between two parties, where one parties task is to convince the other party, that a given string over some alphabet is a statement is some agreed on language. To be more precise. Such a system is more over *zero knowledge* if this possible without revealing any information about the (parts of) that string.

**Definition 7.2.0.1** ((Interactive) Proofing System)**.** *Let $L$ be some formal language over an alphabet $\Sigma$. Then an **interactive proof system** for $L$ is a pair $(P,V)$ of two probabilistic interactive algorithms, where $P$ is called the **prover** and $V$ is called the **verifier**.*

*Both algorithms are able to send messages to one another. Each algorithm only sees its own state, some shared initial state and the communication messages.*

*The verifier is bounded to a number of steps which is polynomial in the size of the shared initial state, after which it stops in an accept state or in a reject state. We impose no restrictions on the local computation conducted by the prover.*

*We require that, whenever the verifier is executed the following two conditions hold:*

- *(Completeness) If a string $x \in \Sigma^*$ is a member of language $L$, that is $x \in L$ and both prover and verifier follow the protocol; the verifier will accept.*
- *(Soundness) If a string $x \in \Sigma^*$ is not a member of language $L$, that is $x \notin L$ and the verifier follows the protocol; the verifier will not be convinced.*
- *(Zero-knowledge) If a string $x \in \Sigma^*$ is a member of language $L$, that is $x \in L$ and the prover follows the protocol; the verifier will not learn anything about $x$ but $x \in L$.*

In the context of zero knowledge proving systems definition XXX gets a slight adaptation:

- Instance: Input commonly known to both prover (P) and verifier (V), and used to support the statement of what needs to be proven. This common input may either be local to the prover-verifier interaction, or public in the sense of being known by external parties (Some scientific articles use "instance" and "statement" interchangeably, but we distinguish between the two.).
- Witness: Private input to the prover. Others may or may not know something about the witness.
- Relation: Specification of relationship between instances and witness. A relation can be viewed as a set of permissible pairs (instance, witness).
- Language: Set of statements that appear as a permissible pair in the given relation.
- Statement:Defined by instance and relation. Claims the instance has a witness in the relation(which is either true or false).

The following subsections define ways to describe checking relations that are particularly useful in the context of zero knowledge proofing systems

### 7.2.1 Succinct NIZK

Preprocessing style: trusted setup, multi party ceremony

Blum, Feldman and Micali extended the notion tonon-interactivezero-knowledge(NIZK) proofs in the common reference string model. NIZK proofs are useful in theconstruction of non-interactive cryptographic schemes, e.g., digital signatures and CCA-secure public key encryption.

**Definition 7.2.1.1.** *Let $\mathscr{R}$ be a relation generator that given a security parameter $\lambda$ in unary returns a polynomial time decidable binary relation R. For pairs $(i,w) \in R$ we call i the instance*[1] *and w the witness. We define $R_\lambda$ to be the set of possible relations R the relation generator may output given $1^\lambda$. We will in the following for notational simplicity assume $\lambda$ can be deduced from the description of R. The relation generator may also output some side information, an auxiliary input z, which will be given to the adversary. An efficient prover publicly verifiable non-interactive argument for R is a quadruple of probabilistic polynomial algorithms* $(\textsc{Setup}, \textsc{Prove}, \textsc{Vfy}, \textsc{Sim})$ *such*

- *Setup: $(CRS,\tau) \to Setup(R)$: The setup produces a common reference string CRS and a simulation trapdoor $\tau$ for the relation R.*
- *Proof: $\pi \to Prove(R,CRS,i,w)$: The prover algorithm takes as input a common reference string CRS and a statement $(i,w) \in R$ and returns an argument $\pi$.*
- *Verify: $0/1 \to Vfy(R,CRS,i,\pi)$: The verification algorithm takes as input a common reference string CRS, an instance i and an argument $\pi$ and returns 0 (reject) or 1 (accept).*
- *$\pi \to Sim(R,\tau,i)$: The simulator takes as input a simulation trapdoor $\tau$ and instance i and returns an argument $\pi$.*

**Common Reference String Generation** Also called trusted setup phase. The field elements needed in this step are called toxic waste ...

**Trusted third party** The most simple approach to generate a common reference string is a so called *trusted third party*. By assumption the entire systems trusts this party to generate the common reference string exactly accoring to the rules and the party will delete all traces of the toxic waste after CRS generation.

**Player exchangeable Multi Party Ceremonies** Achive soundness if only a single party is honest and correctly deletes toxix waste. Is always zero knowledge.

State of the art works in the random beacon model.

A random beacon produces publicly available and verifyable random values at fixed intervals. The difference between random beacons and random oracles, is that random beacons are not available until certain time slots. Random beacons can be instanciated for example by evaluation of say $2^{40}$ iterations of SHA256 on some high entropy, publically available data like the closing value of the stock market on a certain date, the output of a selected set of national lotteries and so on.

The assumption is that any given random beacon value contains large amounts of entropy that is independent from the influence of an adversary in previous time slots.

[1] Note that in Groth16 this is called the statement. We think the term instance is more consistent with SOMETHING.

### Groth16

Groth's constant size NIZK argument is based on constructing a set of polynomial equations and using pairings to efficiently verify these equations. Gennaro, Gentry,Parno and Raykova [Pinocchio] found an insightful construction of polynomial equations based on Lagrange interpolation polynomials yielding a pairing-based NIZK argumentwith a common reference string size proportional to the size of the statement and wit-ness.

It constructs a snark for arithmetic circuit satisfiability, where a proof consists of only 3 group elements. In addition to being small, the proof is also easy to verify. The verifier just needs to compute a number of exponentiations proportional to the instance size and check a single pairing product equation, which only has 3 pairings.

The construction can be instantiated with any type of pairings including Type III pairings, which are the most efficient pairings. The argument has perfect completeness and perfect zero-knowledge. For soundness ??

In the common reference string model.

Setup:

- random elements $\alpha, \beta, \gamma, \delta, s \in \mathbb{F}_{scalar}$
- Common reference string $CRS_{QAP}$, specific to the $QAP$ and the choice of statement and witness $CRS_{QAP} = (CRS_{\mathbb{G}_1}, CRS_{\mathbb{G}_2})$, with $n = deg(t)$:

$$CRS_{\mathbb{G}_1} = \left\{ \begin{array}{c} [\alpha]g, [\beta]g, [\delta]g, \left\{[s^k]g\right\}_{k=0}^{n-1}, \left\{[\frac{\beta a_k(s)+\alpha b_k(s)+c_k(s)}{\gamma}]g\right\}_{k\in I} \\ \left\{[\frac{\beta a_k(s)+\alpha b_k(s)+c_k(s)}{\delta}]g\right\}_{k\in W}, \left\{[\frac{s^k t(s)}{\delta}]g\right\}_{k=0}^{n-2} \end{array} \right\}$$

$$CRS_{\mathbb{G}_2} = \left\{ [\beta]h, [\gamma]h, [\delta]h, \left\{[s^k]h\right\}_{k=0}^{n-1} \right\}$$

- Toxic waste: Must delete random elements after $CRS_{QAP}$ generation.

**Example 91** (Generalized factorization snark)**.** *In this example we want to compile our main example in Groth16. Input is the R1CS from example 83. We choose the following global parameters*

$$\textit{curve = BLS6-6} \quad \mathbb{G}_1 = \textit{BLS6-6(13)} \quad g = (13,15) \quad \mathbb{G}_2 = \quad h = (7v^2, 16v^3) \textit{ and } \mathbb{G}_T = \mathbb{F}^*_{43^6}.$$

**Example 92** (Trusted third party for the factorization snark)**.** *We consider ourself as a trusted third part to generate the common reference string for our generalized factorization snark. We therefore choose the following secret field elements* $\alpha = 6$, $\beta = 5$, $\gamma = 4$, $\delta = 3$, $s = 2$ *from* $\mathbb{F}_{13}$ *and are very careful to hide them from anyone how hasn't read this book. From those values we can then instantiate the common reference string XXX:*

$$CRS_{\mathbb{G}_1} = \left\{ \begin{array}{c} [6](13,15), [5](13,15), [3](13,15), \left\{[s^k](13,15)\right\}_{k=0}^{1}, \left\{[\frac{5a_k(2)+6b_k(2)+c_k(2)}{4}](13,15)\right\}_{k\in S} \\ \left\{[\frac{5a_k(2)+6b_k(2)+c_k(2)}{3}](13,15)\right\}_{k\in W}, \left\{[\frac{s^k t(2)}{3}](13,15)\right\}_{k=0}^{0} \end{array} \right\}$$

*Since we have instance indices* $I = \{1, in_1, in_2\}$ *and witness indices* $W = \{in_3, mid_1, out_1\}$ *we have The instance parts.*

$$\left[\frac{5a_c(2)+6b_c(2)+c_c(2)}{4}\right](13,15) \quad = \quad \left[\frac{5\cdot 0+6\cdot 0+0}{4}\right](13,15) \quad = \quad [0](13,15) \quad = \quad \mathcal{O}$$

$$\left[\frac{5a_{in_3}(2)+6b_{in_3}(2)+c_{in_3}(2)}{4}\right](13,15)=[(5\cdot 0+6\cdot(7\cdot 2+4)+0)\cdot 10](13,15)=$$
$$[(6\cdot 5)\cdot 10](13,15)=[1](13,15)=(13,15)$$

$$\left[\frac{5a_{out}(2)+6b_{out}(2)+c_{out}(2)}{4}\right](13,15)=[(5\cdot 0+6\cdot 0+(7\cdot 2+4))\cdot 10](13,15)=$$
$$[5\cdot 10](13,15)=[11](13,15)=(33,9)$$

*Witness part:*

$$\left[\frac{5a_{in_1}(2)+6b_{in_1}(2)+c_{in_1}(2)}{3}\right](13,15)=[(5\cdot(6\cdot 2+10)+6\cdot 0+0)\cdot 9](13,15)=$$
$$[(5\cdot 9)\cdot 9](13,15)=[2](13,15)=(33,34)$$

$$\left[\frac{5a_{in_2}(2)+6b_{in_2}(2)+c_{in_2}(2)}{3}\right](13,15)=[(5\cdot 0+6\cdot(6\cdot 2+10)+0)\cdot 9](13,15)=$$
$$[(6\cdot 9)\cdot 9](13,15)=[5](13,15)=(26,34)$$

$$\left[\frac{5a_{mid_1}(2)+6b_{mid_1}(2)+c_{mid_1}(2)}{3}\right](13,15)=[(5\cdot(7\cdot 2+4)+6\cdot 0+0)\cdot 9](13,15)=$$
$$[(5\cdot 5)\cdot 9](13,15)=[4](13,15)=(35,28)$$

*For* $\left\{\left[\frac{s^k t(2)}{3}\right](13,15)\right\}_{k=0}^{0}$ *we get*

$$\left[\frac{2^0 t(2)}{3}\right](13,15)=[t(2)\cdot 9](13,15)=[(2^2+2+9)\cdot 9](13,15)=[5](13,15)=(26,34)$$

*All together, the* $\mathbb{G}_1$ *part of the CRS is:*

$$CRS_{\mathbb{G}_1}=\left\{\begin{array}{c}(27,34),(26,34),(38,15),\{(13,15),(33,34)\},\{\mathcal{O},(13,15),(33,9)\}\\ \{(33,34),(26,34),(35,28)\},\{(26,34)\}\end{array}\right\}$$

*To compute the* $\mathbb{G}_2$ *part*

$$CRS_{\mathbb{G}_2}=\left\{[5](7v^2,16v^3),[4](7v^2,16v^3),[3](7v^2,16v^3),\left\{[2^k](7v^2,16v^3)\right\}_{k=0}^{1}\right\}$$

$$CRS_{\mathbb{G}_2}=\left\{[5](7v^2,16v^3),[4](7v^2,16v^3),[3](7v^2,16v^3),\left\{[1](7v^2,16v^3),[2](7v^2,16v^3)\right\}\right\}$$
$$CRS_{\mathbb{G}_2}=\left\{(16v^2,28v^3),(37v^2,27v^3),(42v^2,16v^3),\left\{(7v^2,16v^3),(10v^2,28v^3)\right\}\right\}$$

*So alltogether our common reference string is*

$$\left(\begin{array}{c}\left\{\begin{array}{c}(27,34),(26,34),(38,15),\{(13,15),(33,34)\},\{\mathcal{O},(13,15),(33,9)\}\\ \{(33,34),(26,34),(35,28)\},\{(26,34)\}\end{array}\right\}\\ \left\{(16v^2,28v^3),(37v^2,27v^3),(42v^2,16v^3),\left\{(7v^2,16v^3),(10v^2,28v^3)\right\}\right\}\end{array}\right)$$

**Example 93** (Player exchangeable multi party ceremony for the factorization snark)**.** *In this example we want to simulate a real world player exchangeable multi party ceremony for our factorization snark XXX as explained in XXX.*

*We use our TinyMD5 hash function XXX to hash to $\mathbb{G}_2$.*

*We assume that we have a coordinator Alice together with three parties Bob, Carol and Dave that want to contribute their randomness to the protocol. Since the degree n of the target polynomial is 2, we need to compute the common reference string*

$$CRS = \{\}$$

*For contributer $j > 0$ in phase $l$ to compute the proof of knowledge XXX, we need to define the $transcript_{l,j-1}$ of the previous round. We define it as sha256 of $MPC_{l,j-1}$. To be more precise we define*

$$transcript_{1,j-1} = MD5('[s]g_1[s]g_2[s^2]g_1[\alpha]g_1[\alpha\cdot s]g_1[\beta]g_1[\beta]g_2[\beta\cdot s]g_1')$$

*The only thing actually important about the transcript, is that it is publically available data that is not accesable for anyone before the MPC-data of round $j-1$ in phase $l$ exists.*

*We start with the first round usually called the 'powers of tau' EXPLAIN THAT TERM... The computation is initialized With $s = 1$, $\alpha = 1$, $\beta = 1$. Hence the computation starts with the following data*

$$MPC_{1,0} = \left\{ \begin{array}{lcl} ([s]g_1,[s]g_2) & = & ((13,15),(7v^2,16v^3)) \\ [s^2]g_1 & = & (13,15) \\ [\alpha]g_1 & = & (13,15) \\ [\alpha\cdot s]g_1 & = & (13,15) \\ ([\beta]g_1,[\beta]g_2) & = & ((13,15),(7v^2,16v^3)) \\ [\beta\cdot s]g_1 & = & (13,15) \end{array} \right\}$$

*Then*

$$\begin{aligned} &transcript_{1,0} = \\ &\quad MD5('(13,15)(7v^2,16v^3)(13,15)(13,15)(13,15)(13,15)(7v^2,16v^3)(13,15)') = \\ &\qquad f2baea4d3dba5eef5c63bb210920e7d9 \end{aligned}$$

*We obtain that hash by computing*

$printf'\%s'"(13,15)(7v\hat{}2,16v\hat{}3)(13,15)(13,15)(13,15)(13,15)(7v\hat{}2,16v\hat{}3)(13,15)"|md5sum$

*Everyone agreed, that the MPC starts on the 21.03.2020 and everyone can contribute for exactly a year until the 20.03.2021.*

*It then proceeds in a round robin style, starting with Bob, who optains that data in $MPC_{1,0}$ and then computes his contribution. Lets assume that Bob is honest and that bought a 13-sided dice (PICTURE OF 13-SIDED DICE) to randomly find three secret field values from our prime field $\mathbb{F}_{13}$. He though the dice and got $\alpha = 4$, $\beta = 8$ and $s = 2$. He then updates $MPC_{1,0}$:*

$$MPC_{1,1} = \left\{ \begin{array}{lclcl} ([s]g_1,[s]g_2) & = & ([2](13,15),[2](7v^2,16v^3)) & = & ((33,34),(10v^2,28v^3)) \\ [s^2]g_1 & = & [4](13,15) & = & (35,28) \\ [\alpha]g_1 & = & [4](13,15) & = & (35,28) \\ [\alpha\cdot s]g_1 & = & [8](13,15) & = & (26,9) \\ ([\beta]g_1,[\beta]g_2) & = & ([8](13,15),[8](7v^2,16v^3)) & = & ((26,9),(16v^2,15v^3)) \\ [\beta\cdot s]g_1 & = & [3](13,15) & = & (38,15) \end{array} \right\}$$

*In addition he compute*

$$POK_{1,1}\left\{\begin{array}{lcl} y_s & = & POK(2,f2baea4d3dba5eef5c63bb210920e7d9) = ((33,34),(16v^2,28v^3)) \\ y_\alpha & = & POK(4,f2baea4d3dba5eef5c63bb210920e7d9) = ((35,28),(10v^2,15v^3)) \\ y_\beta & = & POK(8,f2baea4d3dba5eef5c63bb210920e7d9) = ((26,9),(16v^2,28v^3)) \end{array}\right\}$$

*since* $[s]g_1 = (33,34)$, $[\alpha]g_1 = (35,28)$ *and* $[\beta]g_1 = (26,9)$. *as well as*

$$\begin{aligned} TinyMD5_2('(33,34)f2baea4d3dba5eef5c63bb210920e7d9') = \\ H_2(MD5('(33,34)f2baea4d3dba5eef5c63bb210920e7d9').trunc(3)) = \\ H_2(2066b3b6b6d97c46c3ac6ee2ccd23ad9.trunc(3)) = H_2(ad9) = \\ H_2(101011011001) = \\ [8\cdot 4^1\cdot 5^0\cdot 7^1](7v^2,16v^3)+[12\cdot 1^0\cdot 3^1\cdot 8^1](42v^2,16v^3)+ \\ [2\cdot 3^0\cdot 9^1\cdot 11^1](17v^2,15v^3)+[3\cdot 6^0\cdot 9^0\cdot 10^1](10v^2,15v^3) = \\ [8\cdot 4\cdot 7](7v^2,16v^3)+[12\cdot 3\cdot 8](42v^2,16v^3)+[2\cdot 9\cdot 11](17v^2,15v^3)+[3\cdot 10](10v^2,15v^3) = \\ [8\cdot 4\cdot 7](7v^2,16v^3)+[12\cdot 3\cdot 8](42v^2,16v^3)+[2\cdot 9\cdot 11](17v^2,15v^3)+[3\cdot 10](10v^2,15v^3) = \\ [3](7v^2,16v^3)+[2](42v^2,16v^3)+[3](17v^2,15v^3)+[4](10v^2,15v^3) = \\ [3](7v^2,16v^3)+[2*3](7v^2,16v^3)+[3*7](7v^2,16v^3)+[4*11](7v^2,16v^3) = \\ (42v^2,16v^3)+(17v^2,28v^3)+(16v^2,15v^3)+(16v^2,28v^3) = \\ [3](7v^2,16v^3)+[6](7v^2,16v^3)+[8](7v^2,16v^3)+[5](7v^2,16v^3) = \\ [3+6+8+5](7v^2,16v^3) = (37v^2,16v^3) \end{aligned}$$

*So we get* $[2](37v^2,16v^3) = (16v^2,28v^3)$

====================

$$\begin{aligned} TinyMD5_2('(35,28)f2baea4d3dba5eef5c63bb210920e7d9') = \\ H_2(MD5('(35,28)f2baea4d3dba5eef5c63bb210920e7d9').trunc(3)) = \\ H_2(ad54fa3674f6a84fab9208d7a94c9163.trunc(3)) = H_2(163) = \\ H_2(000101100011) = \\ [8\cdot 4^0\cdot 5^0\cdot 7^0](7v^2,16v^3)+[12\cdot 1^1\cdot 3^0\cdot 8^1](42v^2,16v^3)+ \\ [2\cdot 3^1\cdot 9^0\cdot 11^0](17v^2,15v^3)+[3\cdot 6^0\cdot 9^1\cdot 10^1](10v^2,15v^3) = \\ [8](7v^2,16v^3)+[12\cdot 8](42v^2,16v^3)+[2\cdot 3](17v^2,15v^3)+[3\cdot 9\cdot 10](10v^2,15v^3) = \\ [8](7v^2,16v^3)+[5](42v^2,16v^3)+[6](17v^2,15v^3)+[10](10v^2,15v^3) = \\ [8](7v^2,16v^3)+[5*3](7v^2,16v^3)+[6*7](7v^2,16v^3)+[10*11](7v^2,16v^3) = \\ (16v^2,15v^3)+(10v^2,28v^3)+(42v^2,16v^3)+(17v^2,28v^3) = \\ [8](7v^2,16v^3)+[2](7v^2,16v^3)+[3](7v^2,16v^3)+[6](7v^2,16v^3) = \\ [8+2+3+6](7v^2,16v^3) = (17v^2,28v^3) \end{aligned}$$

*So we get* $[4](17v^2,28v^3) = (10v^2,15v^3)$

$$
\begin{aligned}
TinyMD5_2('(26,9)f2baea4d3dba5eef5c63bb210920e7d9') =\\
H_2(MD5('(26,9)f2baea4d3dba5eef5c63bb210920e7d9').trunc(3)) =\\
H_2(b87b632f7027ad78cadc2452beb30e9a.trunc(3)) = H_2(e9a) =\\
H_2(111010011010) =\\
[8\cdot 4^1\cdot 5^1\cdot 7^1](7v^2,16v^3)+[12\cdot 1^0\cdot 3^1\cdot 8^0](42v^2,16v^3)+\\
[2\cdot 3^0\cdot 9^1\cdot 11^1](17v^2,15v^3)+[3\cdot 6^0\cdot 9^1\cdot 10^0](10v^2,15v^3) =\\
[8\cdot 4\cdot 5\cdot 7](7v^2,16v^3)+[12\cdot 3](42v^2,16v^3)+[2\cdot 9\cdot 11](17v^2,15v^3)+[3\cdot 9](10v^2,15v^3) =\\
[2](7v^2,16v^3)+[10](42v^2,16v^3)+[3](17v^2,15v^3)+[1](10v^2,15v^3) =\\
[2](7v^2,16v^3)+[10*3](7v^2,16v^3)+[3*7](7v^2,16v^3)+[1*11](7v^2,16v^3) =\\
(10v^2,28v^3)+(37v^2,27v^3)+(16v^2,15v^3)+(10v^2,15v^3) =\\
[2](7v^2,16v^3)+[4](7v^2,16v^3)+[8](7v^2,16v^3)+[11](7v^2,16v^3) =\\
[2+4+8+11](7v^2,16v^3) = (7v^2,27v^3)
\end{aligned}
$$

*So we get* $[8](17v^2,28v^3) = (16v^2,28v^3)$

*So Bob publishes* $MPC_{1,1}$ *as well as* $POK_{1,1}$ *and after that its Carols turn. Lets also assume that Carrol is honest. So Carol looks at Bobs data and compute the transcript according to our rules*

$$
\begin{aligned}
&transcript_{1,1} =\\
&\quad MD5('(33,34)(10v^2,28v^3)(35,28)(35,28)(26,9)(26,9)(16v^2,15v^3)(38,15)') =\\
&\qquad fe72e18b90014062682a77136944e362
\end{aligned}
$$

*We obtain that hash by computing*

$printf'\%s'"(33,34)(10v^2,28v^3)(35,28)(35,28)(26,9)(26,9)(16v^2,15v^3)(38,15)"|md5sum$

*Carol then computes here contribution. Since she is honest she chooses randomly three secret field values from our prime field* $\mathbb{F}_{13}$*, by invoking her compter. She found* $\alpha = 3$*,* $\beta = 4$ *and* $s = 9$ *and updates* $MPC_{1,1}$*:*

$$
MPC_{1,2} = \left\{
\begin{array}{lclcl}
([s]g_1,[s]g_2) & = & ([9](33,34),[9](10v^2,28v^3)) & = & ((26,34),(16v^2,28v^3))\\
[s^2]g_1 & = & [9\cdot 9](35,28) & = & (13,28)\\
[\alpha]g_1 & = & [3](35,28) & = & (13,28)\\
[\alpha\cdot s]g_1 & = & [3\cdot 9](26,9) & = & (26,9)\\
([\beta]g_1,[\beta]g_2) & = & ([4](26,9),[4](16v^2,15v^3)) & = & ((27,34),(17v^2,28v^3))\\
[\beta\cdot s]g_1 & = & [4\cdot 9](38,15) & = & (35,28)
\end{array}
\right\}
$$

*In addition he compute*

$$
POK_{1,2}\left\{
\begin{array}{lcl}
y_s & = & POK(9,fe72e18b90014062682a77136944e362) = ((35,15),(17v^2,28v^3))\\
y_\alpha & = & POK(3,fe72e18b90014062682a77136944e362) = ((38,15),(17v^2,15v^3))\\
y_\beta & = & POK(4,fe72e18b90014062682a77136944e362) = ((35,28),(42v^2,27v^3))
\end{array}
\right\}
$$

$$
\begin{aligned}
TinyMD5_2('(35,15)fe72e18b90014062682a77136944e362') = \\
H_2(MD5('(35,15)fe72e18b90014062682a77136944e362').trunc(3)) = \\
H_2(115f145ceffdda73e916dc5ba8ae7354.trunc(3)) = H_2(354) = \\
H_2(001101010100) = \\
[8 \cdot 4^0 \cdot 5^0 \cdot 7^1](7v^2, 16v^3) + [12 \cdot 1^1 \cdot 3^0 \cdot 8^1](42v^2, 16v^3) + \\
[2 \cdot 3^0 \cdot 9^1 \cdot 11^0](17v^2, 15v^3) + [3 \cdot 6^1 \cdot 9^0 \cdot 10^0](10v^2, 15v^3) = \\
[8 \cdot 7](7v^2, 16v^3) + [12 \cdot 8](42v^2, 16v^3) + [2 \cdot 9](17v^2, 15v^3) + [3 \cdot 6](10v^2, 15v^3) = \\
[4](7v^2, 16v^3) + [5](42v^2, 16v^3) + [5](17v^2, 15v^3) + [5](10v^2, 15v^3) = \\
[4](7v^2, 16v^3) + [5 * 3](7v^2, 16v^3) + [5 * 7](7v^2, 16v^3) + [5 * 11](7v^2, 16v^3) = \\
(37v^2, 27v^3) + (10v^2, 28v^3) + (37v^2, 16v^3) + (42v^2, 16v^3) = \\
[4](7v^2, 16v^3) + [2](7v^2, 16v^3) + [9](7v^2, 16v^3) + [3](7v^2, 16v^3) = \\
[4 + 2 + 9 + 3](7v^2, 16v^3) = (16v^2, 28v^3)
\end{aligned}
$$

*So we get* $[9](16v^2, 28v^3) = (17v^2, 28v^3)$

$$
\begin{aligned}
TinyMD5_2('(38,15)fe72e18b90014062682a77136944e362') = \\
H_2(MD5('(38,15)fe72e18b90014062682a77136944e362').trunc(3)) = \\
H_2(cc4da0c02c4c1b15e72d6cc6430206ab.trunc(3)) = H_2(6ab) = \\
H_2(011010101011) = \\
[8 \cdot 4^0 \cdot 5^1 \cdot 7^1](7v^2, 16v^3) + [12 \cdot 1^0 \cdot 3^1 \cdot 8^0](42v^2, 16v^3) + \\
[2 \cdot 3^1 \cdot 9^0 \cdot 11^1](17v^2, 15v^3) + [3 \cdot 6^0 \cdot 9^1 \cdot 10^1](10v^2, 15v^3) = \\
[8 \cdot 5 \cdot 7](7v^2, 16v^3) + [12 \cdot 3](42v^2, 16v^3) + [2 \cdot 3 \cdot 11](17v^2, 15v^3) + [3 \cdot 9 \cdot 10](10v^2, 15v^3) = \\
[7](7v^2, 16v^3) + [10](42v^2, 16v^3) + [1](17v^2, 15v^3) + [10](10v^2, 15v^3) = \\
[7](7v^2, 16v^3) + [10 * 3](7v^2, 16v^3) + [1 * 7](7v^2, 16v^3) + [10 * 11](7v^2, 16v^3) = \\
(17v^2, 15v^3) + (17v^2, 28v^3) + (17v^2, 15v^3) + (17v^2, 28v^3) = \\
[7](7v^2, 16v^3) + [4](7v^2, 16v^3) + [7](7v^2, 16v^3) + [6](7v^2, 16v^3) = \\
[7 + 4 + 7 + 6](7v^2, 16v^3) = (10v^2, 15v^3)
\end{aligned}
$$

*So we get* $[3](10v^2, 15v^3) = (17v^2, 15v^3)$

$$\begin{aligned}
TinyMD5_2('(35,28)fe72e18b90014062682a77136944e362') &= \\
H_2(MD5('(35,28)fe72e18b90014062682a77136944e362').trunc(3)) &= \\
H_2(502323bc55c75f7189fad7999c9f1708.trunc(3)) = H_2(708) &= \\
H_2(011100001000) &= \\
[8\cdot 4^0 \cdot 5^1 \cdot 7^1](7v^2,16v^3) + [12\cdot 1^1 \cdot 3^0 \cdot 8^0](42v^2,16v^3) &+ \\
[2\cdot 3^0 \cdot 9^0 \cdot 11^1](17v^2,15v^3) + [3\cdot 6^0 \cdot 9^0 \cdot 10^0](10v^2,15v^3) &= \\
[8\cdot 5\cdot 7](7v^2,16v^3) + [12](42v^2,16v^3) + [2\cdot 11](17v^2,15v^3) + [3](10v^2,15v^3) &= \\
[7](7v^2,16v^3) + [12](42v^2,16v^3) + [9](17v^2,15v^3) + [3](10v^2,15v^3) &= \\
[7](7v^2,16v^3) + [12*3](7v^2,16v^3) + [9*7](7v^2,16v^3) + [3*11](7v^2,16v^3) &= \\
(17v^2,15v^3) + (42v^2,27v^3) + (10v^2,15v^3) + (17v^2,15v^3) &= \\
[7](7v^2,16v^3) + [10](7v^2,16v^3) + [11](7v^2,16v^3) + [7](7v^2,16v^3) &= \\
[7+10+11+7](7v^2,16v^3) = (37v^2,16v^3)
\end{aligned}$$

*So we get* $[4](37v^2,16v^3) = (42v^2,27v^3)$

*Dave thinks he can outsmart the syste, Since he is the last to contribute, he just makes up an entirely new MPC, that does not contain any randomness from the previous contributers. He thinks he can do that because, no one can distinguish his* $MPC_{1,3}$ *from a correct one. If this is done in a smart way, he will even be able to compute the correct POKs.*

*So Dave choses* $s = 12$, $\alpha = 11$ *and* $\beta = 10$ *and he will keep those values, hoping to be able to use them later to forge false proofs in the factorization snark. He then compute*

$$MPC_{1,3} = \left\{ \begin{array}{lcl}
([s]g_1,[s]g_2) & = & ((13,28),(7v^2,27v^3)) \\
[s^2]g_1 & = & (13,15) \\
[\alpha]g_1 & = & (33,9) \\
[\alpha\cdot s]g_1 & = & (33,34) \\
([\beta]g_1,[\beta]g_2) & = & ((38,28),(42v^2,27v^3)) \\
[\beta\cdot s]g_1 & = & (38,15)
\end{array} \right\}$$

*Dave does not delete* $s$, $\alpha$ *and* $\beta$, *because if this is accepted as phase one of the common reference string computation, Dave controls already* $3/4$*-th of the cheating key to forge proofs. So Dave is careful to get the proofs of knowledge right. He computes the transcript of Carols contribution as*

$$\begin{aligned}
&transcript_{1,2} = \\
&\quad MD5('(26,34)(16v^2,28v^3)(13,28)(13,28)(26,9)(27,34)(17v^2,28v^3)(35,28)') = \\
&\qquad\qquad c8e6308fffd47009f5f65e773ae4b499
\end{aligned}$$

*We obtain that hash by computing*
$printf'\%s'"(26,34)(16v^2,28v^3)(13,28)(13,28)(26,9)(27,34)(17v^2,28v^3)(35,28)"|md5sum$

# 8 Exercises and Solutions

TODO: All exercises we provided should have a solution, which we give here in all detail.