Distributed Lab

ZKDL Lecture Notes

Version 0.2

Ukraine

2025

Abstract

Due to the rise of zero-knowledge technologies and their applications in various fields such as Blockchain or anonymous identity management, it is essential to develop a comprehensive understanding of the underlying mechanisms. However, the existing resources on the topic are either too high-level or too low-level, making it hard for regular practicing engineers to understand the practical implications of zero-knowledge protocols.

This book aims to bridge this gap by providing a complete, practical guide to the state-of-the-art techniques in zero-knowledge cryptography, such as Σ-protocols, zk-SNARKs (Groth16 in particular), PlonK and more. We gathered all the necessary information in one place, and tried to make it easy to follow, with numerous examples and code snippets. We attach exercises to each chapter to help you understand the material better. Despite the book's practical focus, we preserve the mathematical rigor where suitable and necessary.

Contents

1 Group Theory and Polynomials

1.1 Notation

Before going into the details, let us introduce some notation.

1.1.1 Set Theory

First, let us enumerate some fundamental sets:

- $\mathbb{N}$ – a set of natural numbers. Examples: $10, 13, 193, \dots$.
- $\mathbb{Z}$ – a set of integers. Examples: $-2, -6, 0, 62, 103, \dots$.
- $\mathbb{Q}$ – a set of rational numbers. Examples: $\{\frac{n}{m} : n \in \mathbb{Z}, m \in \mathbb{N}\}$.
- $\mathbb{R}$ – a set of real numbers. Examples: $2.2, 1.4, -6.7, \dots$.
- $\mathbb{R}_{>0}$ – a set of positive real numbers. Examples: $2.6, 10.4, 100.2$.
- $\mathbb{C}$ – a set of complex numbers[1]. Examples: $1+2i, 5i, -7-5.7i, \dots$.

Typically we write $a \in A$ to say "element a is in set A". To represent the number of elements in a set A, we write $|A|$. If the set is finite, $|A| \in \mathbb{N}$, otherwise $|A| = \infty$. $A \subset B$ denotes "A is a subset of B" (meaning that all elements of A are also in B, e.g., $\mathbb{Q} \subset \mathbb{R}$).

$A \cap B$ means the intersection of A and B (a set of elements belonging to both A and B), while $A \cup B$ – the union of A and B (the set of elements belonging to either A or B). $A \setminus B$ denotes the set difference (the set of elements belonging to A, but not B). $\overline{A}$ denotes the complement of A (the set of elements not belonging to A). All operations are illustrated in Figure 1.1 (this picture is typically called the *Venn Diagram*).

To define the set, we typically write $\{f(a) : \phi(a)\}$, where $f(a)$ is some function and $\phi(a)$ is a predicate (function, inputting a and returning true/false if a certain condition on a is met). For example, $\{x^3 : x \in \mathbb{R}, x^2 = 4\}$ is "a set of values x^3 which are the real solutions to equation $x^2 = 4$". It is quite easy to see that this set is simply $\{2^3, (-2)^3\} = \{8, -8\}$.

The notation $A \times B$ means a set of pairs (a, b) where $a \in A$ and $b \in B$ (or, written shortly, $A \times B = \{(a, b) : a \in A, b \in B\}$), called a Cartesian product. We additionally introduce notation $A^n := \underbrace{A \times A \times \dots \times A}_{n \text{ times}}$ – Cartesian product n times. For example, $\mathbb{Q}^3$ is a set of triplets (a, b, c) where $a, b, c \in \mathbb{Q}$, while $\mathbb{Q}^2 \times \mathbb{R}$ is a set of triplets (a, b, c) where $a, b \in \mathbb{Q}$ and $c \in \mathbb{R}$.

1.1.2 Logic

Statement beginning with $\forall$ means "for all...". For instance, $(\forall a \in A \subset \mathbb{R}) : \{a < 1\}$ is read as: "For any a in set A (which is a subset of real numbers), it is

[1]Complex number is an expression in a form $x + iy$ for $i^2 = -1$

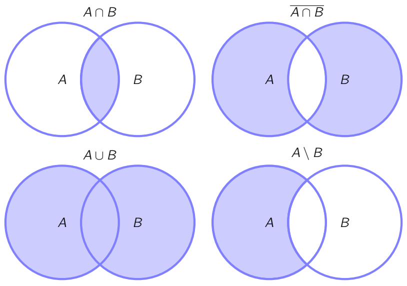


Figure 1.1: Set operations illustrated with Venn diagrams.

true that $a < 1$". Or, more shortly, "Any (real) a from A is less than 1".

Statement beginning from $\exists$ means "there exists such...". Let us consider the following example: $(\exists \varepsilon > 0)(\forall a \in A) : \{a > \varepsilon\}$ is read as "there exists such a positive ε such that for any element a from A, a is greater than ε", or, more concisely, "there exists a positive constant ε such that any element from A is greater than ε".

Statement beginning from $\exists!$ means "there exists a unique...". For example, $(\exists! x \in \mathbb{R}_{>0}) : \{x^2 = 4\}$ is read as "there exists a unique positive real x such that $x^2 = 4$".

Symbol $\wedge$ means "and". For example, $\{x \in \mathbb{R} : x^2 = 4 \wedge x > 0\}$ is read as "a set of real x such that $x^2 = 4$ and x is positive". Of course, $\{x \in \mathbb{R} : x^2 = 4 \wedge x > 0\} = \{2\}$.

Symbol $\vee$ means "or". For example, $\{x \in \mathbb{R} : x^2 = 4 \vee x^2 = 9\}$ is read as "a set of real x such that either $x^2 = 4$ or $x^2 = 9$". Here, this set is equal to $\{-2, 2, -3, 3\}$.

1.1.3 Randomness and Probability

To denote the probability of an event A happening, we write $\Pr[A]$. For example, if event A represents that a coin lands heads, then $\Pr[A] = 0.5$.

Fix some set A. To denote that we are uniformly randomly picking some element from A, we write $a \overset{R}{\leftarrow} A$. For example, $a \overset{R}{\leftarrow} \{1, 2, 3, 4, 5, 6\}$ means that we are picking a number from 1 to 6 uniformly at random.

1.1.4 Sequences and Vectors

To denote the infinite sequence $\{x_1, x_2, x_3, \dots\}$ we write $\{x_n\}_{n \in \mathbb{N}}$. To denote the finite sequence $\{x_1, x_2, \dots, x_n\}$ we write $\{x_k\}_{k=1}^n$.

Vector is a collection of elements $\mathbf{x} = (x_1, \dots, x_n) \in A^n$. Finally, the scalar product[2] is denoted as $\langle \mathbf{x}, \mathbf{y} \rangle := \sum_{k=1}^n x_k y_k$.

[2]It is totally normal if you do not know what that is, we will explain more in the Bulletproof

1.2 Introduction to Abstract Algebra

1.2.1 Groups

Throughout the lectures, probably the most important topic is the *group theory*.

As you can recall from the high school math, typically real-world processes are described using real numbers, denoted by $\mathbb{R}$. For example, to describe the position or the velocity of an object, you would rather use real numbers.

When it comes to working with computers though, real numbers become very inconvenient to work with. For instance, different programming languages might output different values for quite a straightforward operation 2.01 + 2.00. This becomes a huge problem when dealing with cryptography, which must check *precisely* whether two quantities are equal. For example, if the person's card number is N and the developed system operates with a different, but very similar card with number $N + k$ for $k \ll N$, then this system can be safely thrown out of the window. See Figure 1.2.

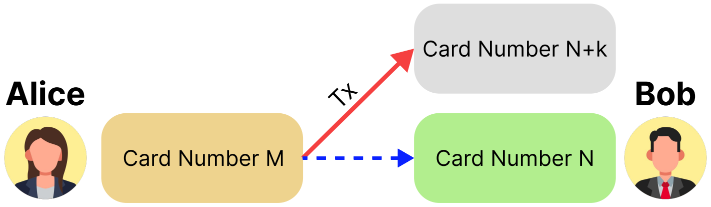


Figure 1.2: Alice pays to Bob to a card number N, but our awesome system pays to $N + k$ instead. Bob would not be happy...

This motivates us to work with integers (denoted by $\mathbb{Z}$), instead. This solves the problem with card numbers, but for cryptography this object is still not really suitable since it is hard to build a secure and reliable protocol exploiting pure integers (without using a more complex structure).

This motivates us to use a different primitive for dealing with cryptographic systems. Similarly to programmers working with interfaces (or traits, if you are the *Rust* developer), mathematicians also use the so-called *groups* to represent objects obeying a certain set of rules. The beauty is that we do not concretize *how* operations in this set are performed, but rather state the fact that we can somehow combine elements with the pre-defined properties. We can then discover properties of such objects and whenever we apply the concrete "implementation" (spoiler, group of points on elliptic curve), these properties would still hold.

lecture

Remark. Further discussion with abstract objects should be regarded as "interfaces" which do not concretize the "implementation" of an object. It merely shows the nature of an object without going into the details.

Now, let us get dirty and define what the **group** is.

Definition 1.1. Group, denoted by $(\mathbb{G}, \oplus)$, is a set with a binary operation $\oplus$, obeying the following rules:

1. **Closure:** Binary operations always outputs an element from $\mathbb{G}$, that is $\forall a, b \in \mathbb{G} : a \oplus b \in \mathbb{G}$.
2. **Associativity:** $\forall a, b, c \in \mathbb{G} : (a \oplus b) \oplus c = a \oplus (b \oplus c)$.
3. **Identity element:** There exists a so-called identity element $e \in \mathbb{G}$ such that $\forall a \in \mathbb{G} : e \oplus a = a \oplus e = a$.
4. **Inverse element:** $\forall a \in \mathbb{G}\ \exists b \in \mathbb{G} : a \oplus b = b \oplus a = e$. We commonly denote the inverse element as $(\ominus a)$.

Quite confusing at first glance, right? The best way to grasp this concept is to consider a couple of examples.

Example. A group of integers with the regular addition $(\mathbb{Z}, +)$ (also called the *additive* group of integers) is a group. Indeed, an identity element is $e_{\mathbb{Z}} = 0$, associativity obviously holds, and an inverse for each element $a \in \mathbb{Z}$ is $(\ominus a) := -a \in \mathbb{Z}$.

Remark. We use the term **additive group** when we mean that the binary operation is addition $+$, while **multiplicative group** means that we are multiplying two numbers via $\times$[a].

[a]In this section, regard $\cdot$ and $\times$ as the same operation of multiplication.

Example. The multiplicative group of positive real numbers $(\mathbb{R}_{>0}, \times)$ is a group for similar reasons. An identity element is $e_{\mathbb{R}_{>0}} = 1$, while the inverse for $a \in \mathbb{R}_{>0}$ is defined as $\frac{1}{a}$.

Example. The additive set of natural numbers $(\mathbb{N}, +)$ is not a group. Although operation of addition is closed, there is no identity element nor inverse element for, say, 2 or 10.

Example. That is possible to have the situation when the element $a \in \mathbb{G}$ can be its own inverse, meaning $a = a^{-1}$. This happens when $a^2 = e$. Additionally, we can mention that for any group $\mathbb{G} = \{g, e\}$ with the order $|\mathbb{G}| = 2$ we have $g^2 = e$.

One might ask a reasonable question: suppose you pick $a, b \in \mathbb{G}$. Is $a \oplus b$ the same as $b \oplus a$? Unfortunately, for some groups, this is not true.

For this reason, it makes sense to give a special name to a group in which the operation is commutative (meaning, we can swap the elements in the operation).

Definition 1.2. A group $(\mathbb{G}, \oplus)$ is called **abelian** if $\forall a, b \in \mathbb{G} : a \oplus b = b \oplus a$.

Example. The additive group of integers $(\mathbb{Z}, +)$ is an abelian group. Indeed, $a + b = b + a$ for any $a, b \in \mathbb{Z}$.

Example. The set of 2×2 matrices with real entries and determinant 1 (denoted by $\mathrm{SL}(2, \mathbb{R})$) is a group with respect to matrix multiplication. However, this group is not abelian! Take

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Then, it is easy to verify that

$$AB = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}, \quad BA = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix},$$

so clearly $AB \neq BA$ – the elements of $\mathrm{SL}(2, \mathbb{R})$ do not commute.

Remark. Further, we will write ab instead of $a \times b$ and a^{-1} instead of $\ominus a$ for the sake of simplicity (and because it is more common in the literature). As mentioned before, it is usually called the *multiplicative notation*.

Finally, for cryptography it is important to know the number of elements in a group. This number is called the *order* of the group.

Definition 1.3. The **order** of a finite group $\mathbb{G}$ is the number of elements in the group. We denote the order of a group as $|\mathbb{G}|$.

Example. Integers modulo 13, denoted by $\mathbb{Z}_{13}$, is a group with respect to addition modulo 13 (e.g., $5 + 12 = 4$ in $\mathbb{Z}_{13}$). The order of this group is 13.

Despite the aforementioned definitions, many things are not generally obvious. For example, one might ask whether the identity element is unique. Or, whether the inverse element is unique for each group element. For that reason, we formulate the following lemma.

Lemma 1.4. Suppose $\mathbb{G}$ is a group. Then, the following statements hold:

1. The identity element is unique.
2. The inverse element is unique for each element: $\forall a \in \mathbb{G}\, \exists! a^{-1} \in \mathbb{G} : aa^{-1} = a^{-1}a = e$.
3. For all $a, b \in \mathbb{G}$ there is a unique $x \in \mathbb{G}$ such that $ax = b$.
4. If $ab = ac$ then $b = c$. Similarly, if $xy = zy$ then $x = z$.

Since this guide is not a textbook on abstract algebra, we will not prove all the statements. However, we will prove the first and second one to show the nature of the proofs in abstract algebra.

First Statement Proof. Suppose $e_1, e_2 \in \mathbb{G}$ are both identity elements. Consider e_1e_2. From the definition of the identity element, we know that $e_1e_2 = e_1$ and $e_1e_2 = e_2$. Therefore, $e_1 = e_2$.

Second Statement Proof. Take $g \in \mathbb{G}$ and suppose $a, b \in \mathbb{G}$ are both inverses of g. By defininition,

$$ag = ga = e, \quad bg = gb = e.$$

Now, notice that

$$a = ae = a(gb) = (ag)b = eb = b$$

Thus, $a = b$.

Exercise. Prove the third and fourth statements.

1.2.2 Subgroups

When we are finally comfortable with the concept of a group, we can move on to the concept of a *subgroup*.

Suppose we have a group $(\mathbb{G}, \oplus)$. Suppose one takes the subset $\mathbb{H} \subset \mathbb{G}$. Of course, since all elements in $\mathbb{H}$ are still elements in $\mathbb{G}$, we can conduct operations between them via $\oplus$. The natural question to ask is whether $\mathbb{H}$ is a group itself. Yes, but at the same time $\mathbb{H}$ is called a **subgroup** of $\mathbb{G}$.

Definition 1.5. A subset $\mathbb{H} \subset \mathbb{G}$ is called a **subgroup** of $\mathbb{G}$ if $\mathbb{H}$ is a group with respect to the same operation $\oplus$. We denote this as $\mathbb{H} \leq \mathbb{G}$.

Example. Of course, not every subset of $\mathbb{G}$ is a subgroup. Take $(\mathbb{Z}, +)$. If we cut, say, 3 out of $\mathbb{Z}$ (so we get $\mathbb{H} = \mathbb{Z} \setminus \{3\}$), then $\mathbb{H}$ is not a subgroup of $\mathbb{Z}$ since an element -3 does not have an inverse in $\mathbb{H}$. Moreover, it is not closed: take $1, 2 \in \mathbb{H}$. In this case, $1 + 2 = 3 \notin \mathbb{H}$.

Example. Now, let us define some valid subgroup of $\mathbb{Z}$. Take $\mathbb{H} = \{3k : k \in \mathbb{Z}\}$ – a set of integers divisible by 3 (commonly denoted as $3\mathbb{Z}$). This is a subgroup of $\mathbb{Z}$, since it is closed under addition, has an identity element 0, and has an inverse for each element $3k$ (namely, $-3k$). That being said, $3\mathbb{Z} \leq \mathbb{Z}$.

These are good examples, but let us consider a more interesting one, which we call a lemma. It is frequently used further when dealing with cosets and normal subgroups, but currently regard this just as an exercise.

Lemma 1.6. Let $\mathbb{G}$ be a group and $g \in \mathbb{G}$. The centralizer of g is defined to be

$$C_g = \{h \in \mathbb{G} : hg = gh\}$$

Then, C_g is a subgroup of $\mathbb{G}$.

Exercise. Prove the lemma.

1.2.3 Cyclic Groups

Probably, cyclic groups are the most interesting groups in the world of cryptography. But before defining them, we need to know how to add/subtract elements multiple times (that is, multiplying by an integer). Suppose we have a group $\mathbb{G}$ and $g \in \mathbb{G}$. Then, g^n means multiplying (adding) g to itself n times. If n is negative, then we add g^{-1} to itself $|n|$ times. For $n = 0$ we define $g^0 = e$. Now, let us define what the cyclic group is.

Definition 1.7. Given a group $\mathbb{G}$ and $g \in \mathbb{G}$ the cyclic subgroup generated by g is

$$\langle g \rangle = \{g^n : n \in \mathbb{Z}\} = \{\dots, g^{-3}, g^{-2}, g^{-1}, e, g, g^2, g^3, \dots\}.$$

Example. Consider the group of integers modulo 12, denoted by $\mathbb{Z}_{12}$. Consider $2 \in \mathbb{Z}_{12}$, the group generated by 2 is then

$$\langle 2 \rangle = \{2, 4, 6, 8, 10, 0\}$$

Definition 1.8. We say that a group $\mathbb{G}$ is **cyclic** if there exists an element $g \in \mathbb{G}$ such that $\mathbb{G}$ is generated by g, that is, $\mathbb{G} = \langle g \rangle$.

Example. The group of integers $(\mathbb{Z}, +)$ is an infinite cyclic group. Indeed, it is generated by 1.

1.2.4 Isomorphisms and Endomorphisms

Finally, we will define the concept of isomorphisms and endomorphisms. These are important concepts in the world of cryptography, since they allow us to compare different groups. Namely, suppose we have two groups $(\mathbb{G}, \oplus)$ and $(\mathbb{H}, \odot)$. Is there any way to state that these two groups are the same? The answer is yes, and this is done via isomorphisms.

Definition 1.9. A function $\varphi : \mathbb{G} \to \mathbb{H}$ is called an **homomorphism** if it is a function that preserves the group operation, that is,

$$\forall a, b \in \mathbb{G} : \varphi(a \oplus b) = \varphi(a) \odot \varphi(b).$$

Definition 1.10. An **isomorphism** is a bijective homomorphism.

Definition 1.11. If there exists an isomorphism between two groups $\mathbb{G}$ and $\mathbb{H}$, we say that these groups are isomorphic and write $\mathbb{G} \cong \mathbb{H}$.

Example. Consider the group of integers $(\mathbb{Z}, +)$ and the group of integers modulo 12 $(\mathbb{Z}_{12}, +)$. The function $\varphi : \mathbb{Z} \to \mathbb{Z}_{12}$ defined as $\varphi(x) = x \bmod 12$ is a homomorphism. Indeed:

$$\varphi(a + b) = (a + b) \bmod 12 = (a \bmod 12) + (b \bmod 12) = \varphi(a) + \varphi(b).$$

However, this function is not an isomorphism, since it is not bijective. For example, $\varphi(0) = \varphi(12) = 0$.

Example. Additive group of reals $(\mathbb{R}, +)$ and the multiplicative group of positive reals $(\mathbb{R}_{>0}, \times)$ are isomorphic. The function $\varphi : \mathbb{R} \to \mathbb{R}_{>0}$ defined as $\varphi(x) = e^x$ is an isomorphism. Indeed:

$$\varphi(a+b) = e^{a+b} = e^a \cdot e^b = \varphi(a) \cdot \varphi(b).$$

Thus, φ is a homomorphism. It is also injective since $e^x = e^y \implies x = y$. Finally, it is obviously onto. This means $(\mathbb{R}, +) \cong (\mathbb{R}_{>0}, \times)$.

Example. All groups of order 2 are isomorphic to $\mathbb{Z}_2$. Indeed, let $\mathbb{G} = \{g, e\}$ – any group of order 2, and define $\varphi : \mathbb{Z}_2 \to \mathbb{G}$ as $\varphi(0) = e$ and $\varphi(1) = g$. This is an isomorphism.

A generalization of the above example is the following quite interesting theorem:

Theorem 1.12. Suppose $\mathbb{G} = \langle g \rangle$ is a finite cyclic group, meaning $|G| = n \in \mathbb{N}$. Then, $\mathbb{G} \cong \mathbb{Z}_n$.

Idea of the proof. Define a function $\varphi : \mathbb{Z}_n \to \mathbb{G}$ as $m \mapsto g^m$. One can prove that this is an isomorphism.

Here, it is quite evident that isomorphism tells us that the groups have the same structure. Moreover, it is correct to say that if $\mathbb{G} \equiv \mathbb{H}$, then $\mathbb{G}$ and $\mathbb{H}$ are *equivalent* since $\cong$ is an equivalence relation.

Exercise (*). Prove that $\cong$ is an equivalence relation.

Finally, we will define the concept of an endomorphism and automorphism to finish the section.

Definition 1.13. An **endomorphism** is a function φ which maps set X to itself ($\varphi : X \to X$).

Definition 1.14. An **automorphism** is an isomorphic endomorphism.

Example. Given a group $\mathbb{G}$, fixate $a \in \mathbb{G}$. The map $\varphi : x \mapsto axa^{-1}$ is an automorphism.

Last two definitions are especially frequently used in Elliptic Curves theory.

1.3 Fields

1.3.1 Formal Definition

Although typically one introduces rings before fields, we believe that for the basic understanding, it is better to start with fields.

Notice that when dealing with groups, we had a single operation $\oplus$, which, depending on the context, is either interpreted as addition or multiplication. However, fields allow to extend this concept a little bit further by introducing a new operation, say, $\odot$, which, combined with $\oplus$, allows us to perform the basic arithmetic.

This is very similar to the real or rational numbers, for example. We can add, subtract, multiply, and divide them. This is exactly what fields are about, but in a more abstract way. That being said, let us see the definition.

Definition 1.15. A **field** is a set $\mathbb{F}$ with two operations $\oplus$ and $\odot$ such that:

1. $(\mathbb{F}, \oplus)$ is an abelian group with identity $e_{\oplus}$.
2. $(\mathbb{F} \setminus \{e_{\oplus}\}, \odot)$ is an abelian group.
3. The **distributive law** holds: $\forall a, b, c \in \mathbb{F} : a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$.

What this definition basically states is that we can perform the following operations:

1. Addition: $a \oplus b$, inherited from group structure $(\mathbb{F}, \oplus)$.
2. Subtraction: $a \oplus (\ominus b)$, inherited from group structure $(\mathbb{F}, \oplus)$.
3. Multiplication: $a \odot b$, inherited from group structure $(\mathbb{F} \setminus \{e_{\oplus}\}, \odot)$.
4. Division: $a \odot b^{-1}$, except for $b = 0$, inherited from group structure $(\mathbb{F} \setminus \{e_{\oplus}\}, \odot)$.

Example. The set of real numbers $(\mathbb{R}, +, \times)$ is obviously a field.

Example. The set of complex numbers $(\mathbb{C}, +, \times)$ is also a field. Indeed, let us see how we can perform operations. Suppose we are given $z = a_0 + a_1 i$ and $w = b_0 + b_1 i$ with $i^2 + 1 = 0$. In this case:

1. Addition: $z + w = (a_0 + b_0) + (a_1 + b_1)i$.
2. Subtraction: $z - w = (a_0 - b_0) + (a_1 - b_1)i$.
3. Multiplication: $z \cdot w = (a_0 b_0 - a_1 b_1) + (a_0 b_1 + a_1 b_0)i$.
4. Division: $z/w = \frac{a_0 b_0 + a_1 b_1}{b_0^2 + b_1^2} + \frac{a_1 b_0 - a_0 b_1}{b_0^2 + b_1^2} i$.

Interestingly though, it is very difficult to come up with some more complicated, non-trivial examples. For that reason, we will simply move to the most central field used in cryptography – finite fields.

1.3.2 Finite Fields

Recall: we do not like reals, we want to operate with integers! But notice that $(\mathbb{Z}, +, \times)$ does not form a field since division is not closed. For that reason, fixate some integer p and consider the set $\mathbb{Z}_p := \{0, 1, 2, \dots, p-2, p-1\}$. Now, we will define operations as follows:

Addition. To add $a, b \in \mathbb{Z}_p$, add them as usual to get $c \leftarrow a + b$. However, this way, operation is not closed, since c might be easily greater than $p-1$ (e.g., for $a = b = p-2$). To fix this, take $c' \in \mathbb{Z}_p$ such that $c \equiv c' \pmod{p}$ (or, written more concisely, $c' = (a+b) \bmod p$).

> **Example.** Take $p = 5$. Then, $3 + 4 = 2$ in $\mathbb{Z}_5$ since $c = 3 + 4 = 7$ and $7 \equiv 2 = c' \pmod{5}$.

Multiplication and subtraction. The algorithm is the same. Find $c \leftarrow ab$ or $c \leftarrow a - b$, respectively, and find $c' \in \mathbb{Z}_p$ such that $c' \equiv c \pmod{p}$.

> **Example.** Again, suppose $p = 5$. Then, $3 \cdot 4 = 2$ in $\mathbb{F}_5$ since $c = 3 \cdot 4 = 12$ and $12 \equiv 2 = c' \pmod{5}$. Similarly, $3 - 4 = 4$ in $\mathbb{F}_5$ since $c = 3 - 4 = -1$ and $-1 \equiv 4 = c' \pmod{5}$.

Inversion. Inversion is a bit more tricky. Recall that $(\mathbb{Z}_p \setminus \{0\}, \times)$ must be an abelian group, meaning that for each $a \in \mathbb{Z}_p$ there should be some $x \in \mathbb{Z}_p$ such that $ax = 1$ (multiplication in a sense of definition above). In other words, we need to solve the modular equation:

$$ax \equiv 1 \pmod{p}.$$

Note that there is no guarantee that for any $a \in \mathbb{Z}_p \setminus \{0\}$ we might find such x. For example, take $p = 10$ and $a = 2$. Then, $2x \equiv 1 \pmod{10}$ has no solution.

The only way to guarantee that for any $a \in \mathbb{Z}_p \setminus \{0\}$ we might find such x is to take p to be a prime number. This is the reason why we call such fields **prime fields** (or, in many cases, one calls them **finite fields**).

So finally, with all the definitions, we can define the finite field.

> **Definition 1.16.** A **finite field** (or *prime field*) is a set with prime number p of elements $\{0, 1, \dots, p-2, p-1\}$, in which operations are defined "modulo p" (see details above).

Typically, finite fields are denoted as $\mathbb{F}_p$ or $\mathrm{GF}(p)$.

Finite fields is the core object in cryptography. Instead of real numbers or pure integers, we will almost always use finite fields.

Remark. In many cases, one might encounter both $\mathbb{F}_p$ and $\mathbb{Z}_p$ notations. The difference is the following: when one refers to $\mathbb{Z}_p$, it is typically assumed that the operations are performed in the ring[a] of integers modulo p (meaning, we need only addition, subtraction, and multiplication in the protocol), while division is of little interest. When one refers to $\mathbb{F}_p$, it is typically assumed that we need full arithmetic (including division) for the procool.

[a]We have not defined as of now what ring is, but, roughly speaking, this is a field without multiplicative inverses

Example. Consider $9, 14 \in \mathbb{F}_{17}$. Some examples of calculations:

1. $9 + 14 = 6$.
2. $9 - 14 = 12$.
3. $9 \times 14 = 7$.
4. $14^{-1} = 11$ since $14 \cdot 11 = 154 \equiv 1 \pmod{17}$.

1.4 Polynomials

1.4.1 Basic Definition

Polynomials are intensively used in almost all areas of cryptography. In our particular case, polynomials will encode the information about statements we will need to prove. That being said, let us define what polynomial is.

Definition 1.17. A **polynomial** $f(x)$ is a function of the form

$$p(x) = c_0 + c_1x + c_2x^2 + \cdots + c_nx^n = \sum_{k=0}^{n} c_kx^k,$$

where $c_0, c_1, \dots, c_n$ are coefficients of the polynomial.

Notice that for now we did not specify what are c_i's. We are interested in the case where $c_i \in \mathbb{F}$, where $\mathbb{F}$ is a field.

Definition 1.18. A set of polynomials depending on x with coefficients in a field $\mathbb{F}$ is denoted as $\mathbb{F}[x]$, that is

$$\mathbb{F}[x] = \left\{ p(x) = \sum_{k=0}^{n} c_k x^k : c_k \in \mathbb{F},\ k = 0, \dots, n \right\}.$$

Definition 1.19. Evaluation of a polynomial $p(x) \in \mathbb{F}[x]$ at point $x_0 \in \mathbb{F}$ is simply finding the value of $p(x_0) \in \mathbb{F}$.

Example. Consider the finite field $\mathbb{F}_3$. Then, some examples of polynomials from $\mathbb{F}_3[x]$ are listed below:

1. $p(x) = 1 + x + 2x^2$.
2. $q(x) = 1 + x^2 + x^3$.
3. $r(x) = 2x^3$.

If we were to evaluate these polynomials at $1 \in \mathbb{F}_3$, we would get:

1. $p(1) = 1 + 1 + 2 \cdot 1 \bmod 3 = 1$.
2. $q(1) = 1 + 1 + 1 \bmod 3 = 0$.
3. $r(1) = 2 \cdot 1 = 2$.

Definition 1.20. The **degree** of a polynomial $p(x) = c_0 + c_1x + c_2x^2 + \dots$ is the largest $k \in \mathbb{Z}_{\geq 0}$ such that $c_k \neq 0$. We denote the degree of a polynomial as $\deg p$. We also denote by $\mathbb{F}^{(\leq m)}[x]$ a set of polynomials of degree at most m.

Example. The degree of the polynomial $p(x) = 1 + 2x + 3x^2$ is 2, so $p(x) \in \mathbb{F}_3^{(\leq 2)}[x]$.

Theorem 1.21. For any two polynomials $p, q \in \mathbb{F}[x]$ and $n = \deg p, m = \deg q$, the following two statements are true:

1. $\deg(pq) = n + m$.
2. $\deg(p + q) = \max\{n, m\}$ if $n \neq m$ and $\deg(p + q) \leq m$ for $m = n$.

1.4.2 Roots and divisibility

Definition 1.22. Let $p(x) \in \mathbb{F}[x]$ be a polynomial of degree $\deg p \geq 1$. A field element $x_0 \in \mathbb{F}$ is called a root of $p(x)$ if $p(x_0) = 0$.

Example. Consider the polynomial $p(x) = 1 + x + x^2 \in \mathbb{F}_3[x]$. Then, $x_0 = 1$ is a root of $p(x)$ since $p(x_0) = 1 + 1 + 1 \bmod 3 = 0$.

One of the fundamental theorems of polynomials is following.

Theorem 1.23. Let $p(x) \in \mathbb{F}[x], \deg p \geq 1$. Then, $x_0 \in \mathbb{F}$ is a root of $p(x)$ if and only if there exists a polynomial $q(x)$ (with $\deg q = n - 1$) such that

$$p(x) = (x - x_0)q(x)$$

Example. Note that $x_0 = 1$ is a root of $p(x) = x^2 + 2$. Indeed, we can write $p(x) = (x - 1)(x - 2)$, so here $q(x) = x - 2$.

Also, this might not be obvious, but we can also divide polynomials in the same way as we divide integers. The result of division is not always a polynomial, so we also get a remainder.

Theorem 1.24. Given $f, g \in \mathbb{F}[x]$ with $g \neq 0$, there are unique polynomials $p, q \in \mathbb{F}[x]$ such that

$$f = q \cdot g + r,\ 0 \leq \deg r < \deg g$$

Example. Consider $f(x) = x^3 + 2$ and $g(x) = x + 1$ over $\mathbb{R}$. Then, we can write $f(x) = (x^2 - x + 1)g(x) + 1$, so the remainder of the division is 1. Typically, we denote this as:

$$f \operatorname{div} g = x^2 - x + 1, \quad f \bmod g = 1.$$

The notation is pretty similar to one used in integer division.

Similarly, one can define gcd, lcm, and other number field theory operations for polynomials. However, we will not go into details here, besides mentioning the divisibility.

Definition 1.25. A polynomial $f(x) \in \mathbb{F}[x]$ is called **divisible** by $g(x) \in \mathbb{F}[x]$ (or, g **divides** f, written as $g \mid f$) if there exists a polynomial $h(x) \in \mathbb{F}[x]$ such that $f = gh$.

Theorem 1.26. If $x_0 \in \mathbb{F}$ is a root of $p(x) \in \mathbb{F}[x]$, then $(x - x_0) \mid p(x)$.

Definition 1.27. A polynomial $f(x) \in \mathbb{F}[x]$ is said to be **irreducible** in $\mathbb{F}$ if there are no polynomials $g, h \in \mathbb{F}[x]$ both of degree more than 1 such that $f = gh$.

Example. A polynomial $f(x) = x^2 + 16$ is irreducible in $\mathbb{R}$. In turn, $f(x) = x^2 - 2$ is not irreducible since $f(x) = (x - \sqrt{2})(x + \sqrt{2})$.

Example. There are no polynomials over complex numbers $\mathbb{C}$ with degree more than 2 that are irreducible. This follows from the *fundamental theorem of algebra*.

1.4.3 Interpolation

Now, let us ask the question: what defines the polynomial? Well, given expression $p(x) = \sum_{k=0}^{n} c_k x^k$ one can easily say: "hey, I need to know the coefficients $\{c_k\}_{k=0}^{n}$".

Indeed, each polynomial of degree n is uniquely determined by the vector of its coefficients $(c_0, c_1, \dots, c_n) \in \mathbb{F}^n$. However, that is not the only way to define a polynomial.

Suppose I tell you that $p(x) = ax + b$ – just a simple linear function over $\mathbb{R}$. Suppose I tell you that $p(x)$ intercepts $(0, 0)$ and $(1, 2)$. Then, you can easily say that $p(x) = 2x$.

The more general question is: suppose $\deg p = n$, how many points do I need to define the polynomial $p(x)$ uniquely? The answer is $n+1$ distinct points. This is the idea behind the interpolation: the polynomial is uniquely defined by $n+1$ distinct points on the plane. An example is depicted in Figure 1.3. Now, let us see how we can interpolate the polynomial practically.

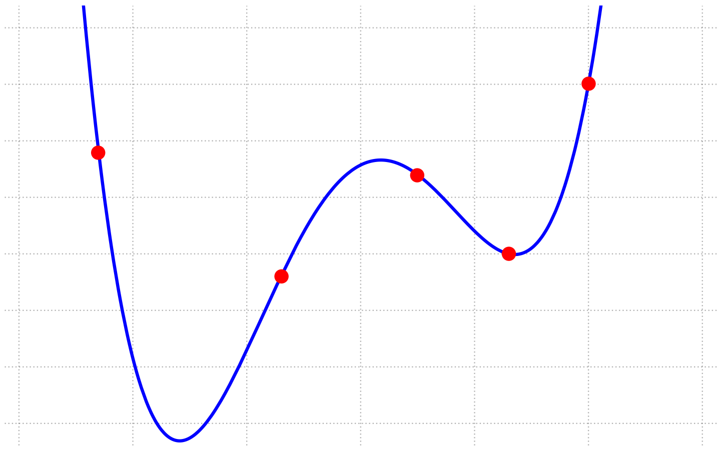


Figure 1.3: 5 points on the plane uniquely define the polynomial of degree 4.

Theorem 1.28. Given a set of points $\{(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)\} \subset \mathbb{F} \times \mathbb{F}$, there is a unique polynomial $L(x)$ of degree n such that $L(x_i) = y_i$ for all $i = 0, \dots, n$. This polynomial is called the **Lagrange interpolation polynomial** and can be found through the following formula:

$$L(x) = \sum_{i=0}^{n} y_i \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j \neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

Lemma 1.29. The polynomials $\{\ell_i\}_{i=1}^{n}$, in fact, have quite an interesting property:

$$\ell_i(x_j) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases},$$

where δ_{ij} is the Kronecker delta. Moreover, $\{\ell_i\}_{i=1}^{n}$ form a basis of $\mathbb{F}^{(\leq n)}[x]$: for any polynomial $p(x) \in \mathbb{F}^{(\leq n)}[x]$ there exist unique coefficients $\alpha_0, \dots, \alpha_n \in \mathbb{F}$ such that

$$p(x) = \sum_{i=0}^{n} \alpha_i \ell_i(x).$$

Example. Suppose we have points $(0, 1)$ and $(1, 2)$. Then, the Lagrange interpolation polynomial is

$$L(x) = 1 \cdot \frac{x-1}{0-1} + 2 \cdot \frac{x-0}{1-0} = (-1) \cdot (x-1) + 2 \cdot x = x + 1$$

1.4.4 Some Fun: Shamir's Secret Sharing

Shamir's Secret Sharing, also known as (t, n)-threshold scheme, is one of the protocols exploiting Lagrange Interpolation.

But first, let us define what secret sharing is. Suppose we have a secret data α, which is represented as an element from some finite set F. We divide this secret into n pieces $\alpha_1, \dots, \alpha_n \in F$ in such a way:

1. Knowledge of any t shares can reconstruct the secret α.
2. Knowledge of any number of shares below t cannot be used to reconstruct the secret α.

Now, let us define the sharing scheme.

Definition 1.30. Secret Sharing scheme is a pair of efficient algorithms $(\mathsf{Gen}, \mathsf{Comb})$ which work as follows:

- $\mathsf{Gen}(\alpha, t, n)$: probabilistic sharing algorithm that yields n shards $(\alpha_1, \dots, \alpha_t)$ for which t shards are needed to reconstruct the secret α.
- $\mathsf{Comb}(\mathcal{I}, \{\alpha_i\}_{i \in \mathcal{I}})$: deterministic reconstruction algorithm that reconstructs the secret α from the shards $\mathcal{I} \subset \{1, \dots, n\}$ of size t.

Here, we require the **correctness**: for every $\alpha \in F$, for every possible output $(\alpha_1, \dots, \alpha_n) \leftarrow \mathsf{Gen}(\alpha, t, n)$, and any t-size subset $\mathcal{I}$ of $\{1, \dots, n\}$ we have

$$\mathsf{Comb}(\mathcal{I}, \{\alpha_i\}_{i \in \mathcal{I}}) = \alpha.$$

Now, Shamir's protocol is one of the most famous secret sharing schemes. It works as follows: our finite set is $\mathbb{F}_q$ for some large prime q. Then, algorithms in the protocol are defined as follows:

- $\mathsf{Gen}(\alpha, k, n)$: choose random $k_1, \dots, k_{t-1} \xleftarrow{R} \mathbb{F}_q$ and define the polynomial

$$\omega(x) := \alpha + k_1 x + k_2 x^2 + \cdots + k_{t-1} x^{t-1} \in \mathbb{F}_q^{\leq (t-1)}[x],$$

  and then compute $\alpha_i \leftarrow \omega(i) \in \mathbb{F}_q$, $i = 1, \dots, n$. Return $(\alpha_1, \dots, \alpha_n)$.
- $\mathsf{Comb}(\mathcal{I}, \{\alpha_i\}_{i \in \mathcal{I}})$: reconstruct the polynomial $\omega(x)$ using Lagrange interpolation and return $\omega(0) = \alpha$.

The combination function is possible since, having t points $\{i, \alpha_i\}_{i \in \mathcal{I}}$ with $\omega(i) = \alpha_i$, we can fully reconstruct the polynomial $\omega(x)$ and then evaluate it at

0 to get α.

Instead, suppose we have only $t-1$ (or less) pairs $\{i, \alpha_i\}_{i \in \mathcal{I}}$. Then, there are many polynomials $\omega(x)$ that pass through these points (in fact, if we were in the field of real numbers, this number would be infinite), and thus the secret α is not uniquely determined.

The intuition behind the Shamir's protocol is illustrated in Figure 1.4.

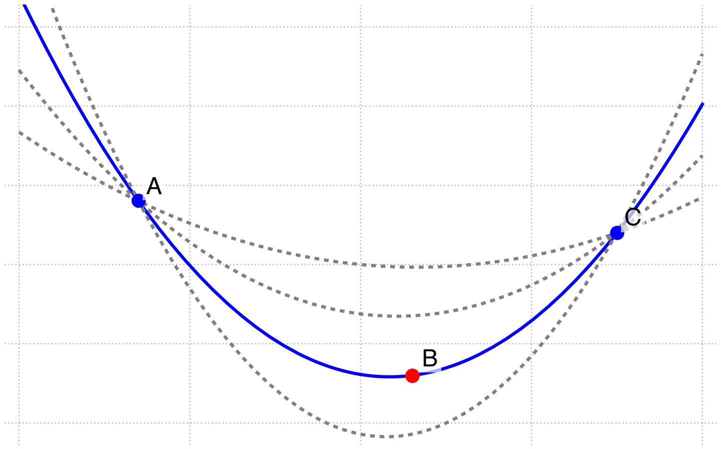


Figure 1.4: Suppose we have $t = 3$. Having only 2 points means knowing two blue points without knowing the red one. There are infinitely many quadratic polynomials passing through these two points (gray dashed lines). However, knowing the third red point allows us to uniquely determine the polynomial and thus get its value at 0. Note that this is illustrated over $\mathbb{R}$, but for $\mathbb{F}_q$ the logic is similar.

1.4.5 Some Fun: Group Implementation in Rust

In programming, we can think of a group as an interface, having a single binary operation defined, that obeys the rules of closure, associativity, identity element, and inverse element.

For that reason, we might even code a group in Rust! We will also write a simple test to check whether the group is valid and whether the group is abelian.

Trait for Group. First, we define a trait for a group. We will define a group as a trait with the following methods:

```rust
/// Trait that represents a group.
pub trait Group: Sized {
    /// Checks whether the two elements are equal.
    fn eq(&self, other: &Self) -> bool;
    /// Returns the identity element of the group.
    fn identity() -> Self;
    /// Adds two elements of the group.
    fn add(&self, a: &Self) -> Self;
    /// Returns the negative of the element.
    fn negate(&self) -> Self;
    /// Subtracts two elements of the group.
    fn sub(&self, a: &Self) -> Self {
        self.add(&a.negate())
    }
}
```

Checking group validity. Now observer the following: we get closure for free, since the compiler will check whether the return type of the operation is the same as the type of the group. However, there is no guarantee that associativity holds, and our identity element is at all valid. For that reason, we need to somehow additionally check the validity of implementation.

We propose to do the following: we will randomly sample three elements from the group $a, b, c \xleftarrow{R} \mathbb{G}$ and check our three properties:

1. $a \oplus (b \oplus c) \overset{?}{=} (a \oplus b) \oplus c$.
2. $a \oplus e \overset{?}{=} e \oplus a \overset{?}{=} a$.
3. $a \oplus (\ominus a) \overset{?}{=} (\ominus a) \oplus a \overset{?}{=} e$.

Additionally, if we want to verify whether the group is abelian, we can check whether $a \oplus b \overset{?}{=} b \oplus a$.

For that reason, for the check, we require the group to be samplable (i.e. we can randomly sample elements from the group):

```
/// Trait for sampling a random element from a group.
pub trait Samplable {
    /// Returns a random element from the group.
    fn sample() -> Self;
}
```

And now, our test looks as follows:

```
/// Number of tests to check the group properties.
const TESTS_NUMBER: usize = 100;

/// Asserts that the given group G is valid.
/// A group is valid if the following properties hold:
/// 1. Associativity: (a + b) + c = a + (b + c)
/// 2. Identity: a + e = a = e + a
/// 3. Inverse: a + (-a) = e = (-a) + a
```

```
pub fn assert_group_valid<G>()
where
    G: Group + Samplable,
{
    for _ in 0..TESTS_NUMBER {
        // Take random three elements
        let a = G::sample();
        let b = G::sample();
        let c = G::sample();

        // Check whether associativity holds
        let ab_c = a.add(&b).add(&c);
        let a_bc = a.add(&b.add(&c));
        let associativity_holds = ab_c.eq(&a_bc);
        assert!(associativity_holds, "Associativity does
         ↪ not hold for the given group");

        // Check whether identity element is valid
        let e = G::identity();
        let ae = a.add(&e);
        let ea = e.add(&a);
        let identity_holds = ae.eq(&a) && ea.eq(&a);
        assert!(identity_holds, "Identity element does not
         ↪ hold for the given group");

        // Check whether inverse element is valid
        let a_neg = a.negate();
        let a_neg_add_a = a_neg.add(&a);
        let a_add_a_neg = a.add(&a_neg);
        let inverse_holds = a_neg_add_a.eq(&e) &&
         ↪ a_add_a_neg.eq(&e);
        assert!(inverse_holds, "Inverse element does not
         ↪ hold for the given group");
    }
}

/// Asserts that the given group G is abelian.
/// A group is an abelian group if the following property
 ↪ holds:
/// a + b = b + a for all a, b in G (commutativity)
pub fn assert_group_abelian<G>()
where
    G: Group + Samplable,
{
    for _ in 0..TESTS_NUMBER {
        assert_group_valid::<G>();

        // Take two random elements
        let a = G::sample();
        let b = G::sample();

        // Check whether commutativity holds
        let ab = a.add(&b);
        let ba = b.add(&a);
        assert!(ab.eq(&ba), "Commutativity does not hold
         ↪ for the given group");
    }
}
```

Testing the group $(\mathbb{Z}, +)$**.** And now, we can define a group for integers and check whether it is valid and abelian:

```
use crate::group::{Group, Samplable};
use rand::Rng;

/// Implementing group for Rotation3<f32>
impl Group for i64 {
    fn eq(&self, other: &Self) -> bool {
        self == other
    }

    fn identity() -> Self {
        0i64
    }

    fn add(&self, a: &Self) -> Self {
        self + a
    }

    fn negate(&self) -> Self {
        -self
    }
}

impl Samplable for i64 {
    fn sample() -> Self {
        let mut gen = rand::thread_rng();

        // To prevent overflow, we choose a smaller range
         ↪ for i64
```

```rust
        let min = i64::MIN / 3;
        let max = i64::MAX / 3;
        gen.gen_range(min..max)
    }
}
```

Just a small note: since we cannot generate infinite integers, we restrict the range of integers to prevent overflow. So, for the sake of simplicity, we divide the range of integers by 3, in which overflow never occurs.

And now, the moment of truth! Let us define some tests and run them:

```rust
#[cfg(test)]
mod tests {
    use super::*;
    use group::*;

    #[test]
    fn test_integers_are_group() {
        assert_group_valid::<i64>()
    }

    #[test]
    fn test_integers_are_abelian() {
        assert_group_abelian::<i64>();
    }
}
```

Both tests pass! Now let us consider something a bit trickier.

Testing the group SO(3). We can define a group for 3×3 rotation matrices. Of course, composition of two rotation is not commutative, so we expect the abelian test to fail. However, the group is still valid! For example, there is an identity rotation matrix E, and for each rotation matrix $A \in \mathrm{SO}(3)$, there exists a rotation matrix $A^{-1} \in \mathrm{SO}(3)$ such that $AA^{-1} = A^{-1}A = E$. Finally, the associativity holds as well.

We will use the `nalgebra` library for this purpose, which contains the implementation of rotation matrices. So our implementation can look as follows:

```rust
/// A threshold below which two floating point numbers are
 ↪ considered equal.
const EPSILON: f32 = 1e-6;

/// Implementing group for Rotation3<f32>
impl Group for Rotation3<f32> {
    fn eq(&self, other: &Self) -> bool {
        // Checking whether the norm of a difference is
         ↪ small
        let difference = self.matrix() - other.matrix();
        difference.norm_squared() < EPSILON
    }

    fn identity() -> Self {
        Rotation3::identity()
    }

    fn add(&self, a: &Self) -> Self {
        self * a
    }

    fn negate(&self) -> Self {
        self.inverse()
    }
}

impl Samplable for Rotation3<f32> {
    fn sample() -> Self {
        let mut gen = rand::thread_rng();

        // Pick three random angles
        let roll = gen.gen_range(0.0..1.0);
        let pitch = gen.gen_range(0.0..1.0);
        let yaw = gen.gen_range(0.0..1.0);

        Rotation3::from_euler_angles(roll, pitch, yaw)
    }
}
```

Here, there are two tricky moments:

1. We cannot compare floating point numbers directly, since they might differ by a small amount. For that reason, we define a small threshold ε. We say that two matrices are equal iff the norm[3] of their difference is less than ε.
2. To generate a random rotation matrix, we generate three random angles and create a rotation matrix from these angles.

[3]one can think of norm as being the measure of "distance" between two objects. Similarly, we can define norm not only on matrices, but on vectors as well.

1.5 Exercises

Exercise 1. Which of the following statements is **false**?

1. $(\forall a, b \in \mathbb{Q}, a \neq b)\,(\exists q \in \mathbb{R}) : \{a < q < b\}$.
2. $(\forall \varepsilon > 0)\,(\exists n_\varepsilon \in \mathbb{N})\,(\forall n \geq n_\varepsilon) : \{1/n < \varepsilon\}$.
3. $(\forall k \in \mathbb{Z})\,(\exists n \in \mathbb{N}) : \{n < k\}$.
4. $(\forall x \in \mathbb{Z} \setminus \{-1\})\,(\exists! y \in \mathbb{Q}) : \{(x+1)y = 2\}$.

Exercise 2. Denote $X := \{(x, y) \in \mathbb{Q}^2 : xy = 1\}$. Oleksandr claims the following:

1. $X \cap \mathbb{N}^2 = \{(1, 1)\}$.
2. $|X \cap \mathbb{Z}^2| = 2|X \cap \mathbb{N}^2|$.
3. X is a group under the operation $(x_1, y_1) \oplus (x_2, y_2) = (x_1x_2, y_1y_2)$.

Which statements are **true**?

a) Only 1.
b) Only 1 and 2.
c) Only 1 and 3.
d) Only 2 and 3.
e) All statements are correct.

Exercise 3. Does a tuple $(\mathbb{Z}, \oplus)$ with operation $a \oplus b = a + b - 1$ define a group?

a) Yes, and this group is abelian.
b) Yes, but this group is not abelian.
c) No, since the associativity property does not hold.
d) No, since there is no identity element in this group.
e) No, since there is no inverse element in this group.

Exercise 4. Consider the Cartesian plane $\mathbb{R}^2$, where two coordinates are real numbers. For two points A, B define the operation $\oplus$ as follows: $A \oplus B$ is the midpoint on segment AB. Does $(\mathbb{R}^2, \oplus)$ define a group?

a) Yes, and this group is abelian.
b) Yes, but this group is not abelian.
c) No, since the associativity property does not hold and there is no identity element in this group.
d) No, since the associativity property does not hold, but we might define an identity element nonetheless.

Exercise 5. Find the inverse of 4 in $\mathbb{F}_{11}$.

a) 8
b) 5
c) 3
d) 7

Exercise 6. Suppose for three polynomials $p, q, r \in \mathbb{F}[x]$ we have $\deg p = 3, \deg q = 4, \deg r = 5$. Which of the following is true for $n := \deg\{(p-q)r\}$?

a) $n = 9$.
b) n might be less than 9.
c) $n = 20$.
d) n is less than $\deg\{qr\}$.

Exercise 7. Define the polynomial over $\mathbb{F}_5$: $f(x) := 4x^2 + 7$. Which of the following is the root of $f(x)$?

a) 2
b) 3
c) 4
d) No roots.

Exercise 8. Quadratic polynomial $p(x) = ax^2 + bx + c \in \mathbb{R}[x]$ has zeros at 1 and 2 and $p(0) = 2$. Find the value of $a + b + c$.

a) 0
b) -1
c) 1
d) Not enough information.

Exercise 9. Which of the following is a **valid** endomorphism $f : X \to X$?

a) $X = [0, 1]$, $f : x \mapsto x^2$.
b) $X = [0, 1]$, $f : x \mapsto x + 1$.
c) $X = \mathbb{R}_{>0}$, $f : x \mapsto (x-1)^3$.
d) $X = \mathbb{Q}_{>0}$, $f : x \mapsto \sqrt{x}$.

Exercise 10*. Denote by $\mathrm{GL}(2, \mathbb{R})$ a set of 2×2 invertable matrices with real entries. Define two functions $\varphi : \mathrm{GL}(2, \mathbb{R}) \to \mathbb{R}$:

$$\varphi_1\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc, \ \varphi_2\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = a + d$$

Den claims the following:

1. φ_1 is a group homomorphism between multiplicative groups $(\mathrm{GL}(2, \mathbb{R}), \times)$ and $(\mathbb{R}, \times)$.
2. φ_2 is a group homomorphism between additive groups $(\mathrm{GL}(2, \mathbb{R}), +)$ and $(\mathbb{R}, +)$.

Which of the following is **true**?

a) Only statement 1 is correct.
b) Only statement 2 is correct.
c) Both statements 1 and 2 are correct.
d) None of the statements is correct.

2 Basics of Security Analysis

2.1 Basics of Security Analysis

In many cases, technical papers include the analysis on the key question: "How secure is this cryptographic algorithm?" or rather "Why this cryptographic algorithm is secure?". In this section, we will shortly describe the notation and typical construction for justifying the security of cryptographic algorithms.

Typically, the cryptographic security is defined in a form of a game between the adversary (who we call $\mathcal{A}$) and the challenger (who we call $\mathcal{C}h$). The adversary is trying to break the security of the cryptographic algorithm using arbitrary (but still efficient) protocol, while the challenger is following a simple, fixed protocol. The game is played in a form of a challenge, where the adversary is given some information and is asked to perform some task. The security of the cryptographic algorithm is defined based on the probability of the adversary to win the game.

2.1.1 Cipher Semantic Security

Let us get into specifics. Suppose that we want to specify that the encryption scheme is secure. Recall that cipher $\mathcal{E} = (E, D)$ over the space $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ (here, $\mathcal{K}$ is the space containing all possible keys, $\mathcal{M}$ — all possible messages and $\mathcal{C}$ — all possible ciphers) consists of two efficiently computable methods:

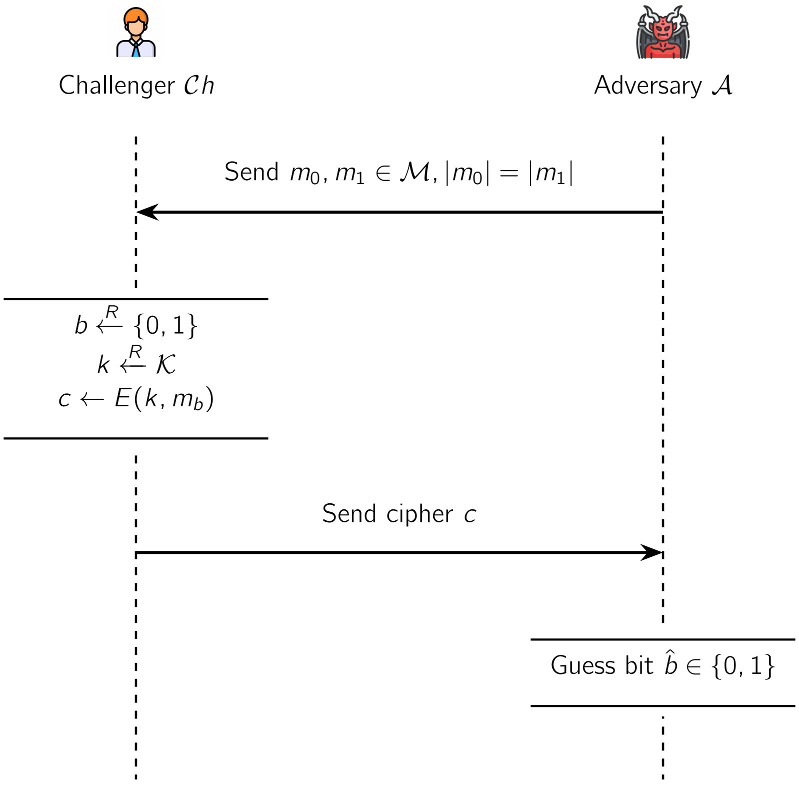
- $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ — encryption method, that based on the provided message $m \in \mathcal{M}$ and key $k \in \mathcal{K}$ outputs the cipher $c = E(k, m) \in \mathcal{C}$.
- $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ — decryption method, that based on the provided cipher $c \in \mathcal{C}$ and key $k \in \mathcal{K}$ outputs the message $m = D(k, c) \in \mathcal{M}$.

Of course, we require the **correctness**:

$$(\forall k \in \mathcal{K})\,(\forall m \in \mathcal{M}) : \{D(k, E(k, m)) = m\}$$

Now let us play the following game between adversary $\mathcal{A}$ and challenger $\mathcal{C}h$:

1. $\mathcal{A}$ picks any two messages $m_0, m_1 \in \mathcal{M}$ on his choice.
2. $\mathcal{C}h$ picks a random key $k \xleftarrow{R} \mathcal{K}$ and random bit $b \xleftarrow{R} \{0, 1\}$ and sends the cipher $c = E(k, m_b)$ to $\mathcal{A}$.
3. $\mathcal{A}$ is trying to guess the bit b by using the cipher c.
4. $\mathcal{A}$ outputs the guess $\hat{b}$.



Challenger $\mathcal{C}h$
Adversary $\mathcal{A}$
Send $m_0, m_1 \in \mathcal{M}, |m_0| = |m_1|$
$b \xleftarrow{R} \{0,1\}$
$k \xleftarrow{R} \mathcal{K}$
$c \leftarrow E(k, m_b)$
Send cipher c
Guess bit $\hat{b} \in \{0,1\}$

Figure 2.1: The game between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}h$ for defining the semantic security.

Now, what should happen if our encryption scheme is secure? The adversary should not be able to guess the bit b with a probability significantly higher than $1/2$ (a random guess). Formally, define the **advantage** of the adversary $\mathcal{A}$ as:

$$\mathrm{SSAdv}[\mathcal{E}, \mathcal{A}] := \left| \Pr\left[\hat{b} = b\right] - \frac{1}{2} \right|$$

We say that the encryption scheme is **semantically secure**[4] if for any efficient adversary $\mathcal{A}$ the advantage $\mathrm{SSAdv}[\mathcal{A}]$ is negligible. In other words, the adversary cannot guess the bit b with a probability significantly higher than $1/2$.

Now, what negligible means? Let us give the formal definition!

[4]This version of definition is called a **bit-guessing** version.

Definition 2.1. A function $f : \mathbb{N} \to \mathbb{R}$ is called **negligible** if for all $c \in \mathbb{R}_{>0}$ there exists $n_c \in \mathbb{N}$ such that for any $n \geq n_c$ we have $|f(n)| < 1/n^c$.

The alternative definition, which is problably easier to interpret, is the following.

Theorem 2.2. A function $f : \mathbb{N} \to \mathbb{R}$ is **negligible** if and only if for any $c \in \mathbb{R}_{>0}$, we have

$$\lim_{n \to \infty} f(n)n^c = 0$$

Example. The function $f(n) = 2^{-n}$ is negligible since for any $c \in \mathbb{R}_{>0}$ we have

$$\lim_{n \to \infty} 2^{-n}n^c = 0$$

The function $g(n) = \frac{1}{n!}$ is also negligible for similar reasons.

Example. The function $h(n) = \frac{1}{n}$ is not negligible since for $c = 1$ we have

$$\lim_{n \to \infty} \frac{1}{n} \times n = 1 \neq 0$$

Well, that is weird. For some reason we are considering a function the depends on some natural number n, but what is this number?

Typically, when defining the security of the cryptographic algorithm, we are considering the security parameter λ (e.g., the length of the key). The function is negligible if the probability of the adversary to break the security of the cryptographic algorithm is decreasing with the increasing of the security parameter λ. Moreover, we require that the probability of the adversary to break the security of the cryptographic algorithm is decreasing faster than any polynomial function of the security parameter λ.

So all in all, we can define the semantic security as follows.

Definition 2.3. The encryption scheme $\mathcal{E}$ with a security paramter $\lambda \in \mathbb{N}$ is **semantically secure** if for any efficient adversary $\mathcal{A}$ we have:

$$\left| \Pr\left[b = \hat{b} \;\middle|\; \begin{array}{c} m_0, m_1 \leftarrow \mathcal{A},\; k \xleftarrow{R} \mathcal{K},\; b \xleftarrow{R} \{0,1\} \\ c \leftarrow E(k, m_b) \\ \hat{b} \leftarrow \mathcal{A}(c) \end{array} \right] - \frac{1}{2} \right| < \mathsf{negl}(\lambda)$$

Do not be afraid of such complex notation, it is quite simple. Notation $\Pr[A \mid B]$ means "the probability of A, given that B occurred". So our inner probability is read as "the probability that the guessed bit $\hat{b}$ equals b given the setup on the right". Then, on the right we define the setup: first we generate two messages $m_0, m_1 \in \mathcal{M}$, then we choose a random bit b and a key k, cipher the message m_b, send it to the adversary and the adversary, based on provided cipher, gives $\hat{b}$ as an output. We then claim that the probability of the adversary to guess the bit b is close to $1/2$.

Let us see some more examples of how to define the security of certain crypographic objects.

2.1.2 Message recovery attacks

Essentially, message recovery attacks are types of attacks that, from given ciphertext, recover the message with significantly better probability than random guessing, which is obviously $1/|\mathcal{M}|$. Of course, any reasonable notion of security should rule out such an attack, and indeed, semantic security does.

Although this might seem intuitively obvious, we provide a formal proof to clarify the reasoning. One of the reasons for doing this is to thoroughly demonstrate the concept of a *security reduction*, which is the primary method used to assess the security of systems.

Let us play the following attack game message recovery between adversary $\mathcal{A}$ and challenger $\mathcal{C}h$. For a given cipher $\mathcal{E} = (E, D)$, defined over $\mathcal{K}, \mathcal{M}, \mathcal{C}$, and for a given adversary $\mathcal{A}$, the attack game proceeds as follows:

- The challenger computes $m \overset{R}{\leftarrow} \mathcal{M}$, $k \overset{R}{\leftarrow} \mathcal{K}$, $c \overset{R}{\leftarrow} E(k, m)$ and sends c to the adversary.
- the adversary output a message $\hat{m} \in \mathcal{M}$.

We say that $\mathcal{A}$ wins the game in this case, and we define $\mathcal{A}$'s message recovery advantage with respect to $\mathcal{E}$ as:

$$\text{MRadv}[\mathcal{E}, \mathcal{A}] := \left| \Pr[\hat{m} = m] - \frac{1}{|\mathcal{M}|} \right|.$$

Definition 2.4 (Security against message recovery)**.** A cipher $\mathcal{E}$ is secure against message recovery if for all efficient adversaries $\mathcal{A}$, the value $\text{MRadv}[\mathcal{E}, \mathcal{A}]$ is negligible.

Theorem 2.5. Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. If $\mathcal{E}$ is semantically secure then $\mathcal{E}$ is secure against message recovery.

▶ Let us prove by assuming the opposite statement. In other words, if $\mathcal{E}$ is not secure against message recovery, then $\mathcal{E}$ is not semantically secure.

Assume that $\mathcal{E}$ is not secure against message recovery. We have an efficient adversary $\mathcal{A}$ who has a significant advantage in the message recovery game and an efficient adversary $\mathcal{B}$ who is trying to compromise $\mathcal{E}$. The proof idea is to use an efficient adversary $\mathcal{A}$ as a "black box" (or oracle machine). We construct $\mathcal{B}$ as follows:

1. Adversary $\mathcal{B}$ generates $m_0, m_1 \in \mathcal{M}$ and sends them to the semantic security challenger.
2. The semantic security challenger returns ciphertext c to adversary $\mathcal{B}$. Since $\mathcal{B}$ can use the message recovery oracle, it sends c to $\mathcal{A}$.
3. $\mathcal{A}$ returns $\hat{m}$. If $\hat{m} = m_0$, then $b = 0$ otherwise, $b = 1$.
4. Sends b to the semantic security challenger.

Intuitively, this implies that SSadv$[\mathcal{E}, \mathcal{B}]$ is exactly equal to MRadv$[\mathcal{E}, \mathcal{A}]$. ◀

This means that if an adversary is capable of successfully winning the message recovery game, then they can also effectively win the semantic security game. In essence, security against message recovery is a stronger property than semantic security.

Our motivation for demonstrating proof is to illustrate in detail the notion of a security reduction, which is the main technique used to reason about the security of systems.

Security reduction is typically applied to certain computational problems. To be more precise, we will now explain how security reduction operates in general.

Definition 2.6 (Security Reduction)**.**

1. Let $\mathcal{A}$ be an efficient adversary who is trying to compromise the cryptosystem $\mathcal{E}$.
2. We construct an efficient algorithm $\mathcal{A}'$, which we call the **reduction**, that will solve the computationally hard problem $\mathcal{X}$ while using an oracle access to $\mathcal{A}$. For any input x of the problem $\mathcal{X}$, the algorithm $\mathcal{A}'$ generates the input of the cryptosystem $\mathcal{E}$ for $\mathcal{A}$. Once it wins the corresponding experiment, $\mathcal{A}'$ solves $\mathcal{X}$ for input x with a certain non-negligible probability $\mu(n)$. Typically, such probability turns out to be bounded below by $1/n^c, c \in \mathbb{N}$.
3. Thus, the algorithm $\mathcal{A}'$ solves $\mathcal{X}$ with the non-negligible probability $\varepsilon(n)\mu(n)$ (or typically $\varepsilon(n)/n^c$). This quantity is non-negligible if $\varepsilon(n)$ is assumed to be non-negligible.
4. Since the problem $\mathcal{X}$ is computationally hard, we get a contradic-

tion. It follows that no attacker $\mathcal{A}$ can effectively crack $\mathcal{E}$, i.e., this cryptosystem is computationally strong.

2.1.3 Discrete Logarithm Assumption

Now, let us define the fundamental assumption used in cryptography formally: the **Discrete Logarithm Assumption** (DL).

Definition 2.7. Assume that $\mathbb{G}$ is a cyclic group of prime order r generated by $g \in \mathbb{G}$. Define the following game:

1. Both challenger $\mathcal{C}h$ and adversary $\mathcal{A}$ take a description $\mathbb{G}$ as an input: order r and generator $g \in \mathbb{G}$.
2. $\mathcal{C}h$ computes $\alpha \xleftarrow{R} \mathbb{Z}_r$, $u \leftarrow g^\alpha$ and sends $u \in \mathbb{G}$ to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ outputs $\hat{\alpha} \in \mathbb{Z}_r$.

We define $\mathcal{A}$'s **advantage in solving the discrete logarithm problem in** $\mathbb{G}$, denoted as $\mathsf{DLadv}[\mathcal{A}, \mathbb{G}]$, as the probability that $\hat{\alpha} = \alpha$.

Definition 2.8. The **Discrete Logarithm Assumption** holds in the group $\mathbb{G}$ if for any efficient adversary $\mathcal{A}$ the advantage $\mathsf{DLadv}[\mathcal{A}, \mathbb{G}]$ is negligible.

Informally, this assumption means that given u, it is very hard to find α such that $u = g^\alpha$. But now we can write down this formally!

2.1.4 Computational Diffie-Hellman

Another fundamental problem in cryptography is the **Computational Diffie-Hellman** (CDH) problem. It states that given g^α, g^β it is hard to find $g^{\alpha\beta}$. This property is frequently used in the construction of cryptographic protocols such as the Diffie-Hellman key exchange.

Let us define this problem formally.

Definition 2.9. Let $\mathbb{G}$ be a cyclic group of prime order r generated by $g \in \mathbb{G}$. Define the following game:

1. Both challenger $\mathcal{C}h$ and adversary $\mathcal{A}$ take a description $\mathbb{G}$ as an input: order r and generator $g \in \mathbb{G}$.
2. $\mathcal{C}h$ computes $\alpha, \beta \xleftarrow{R} \mathbb{Z}_r$, $u \leftarrow g^\alpha$, $v \leftarrow g^\beta$, $w \leftarrow g^{\alpha\beta}$ and sends $u, v \in \mathbb{G}$ to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ outputs $\hat{w} \in \mathbb{G}$.

We define $\mathcal{A}$'s **advantage in solving the computational Diffie-Hellman**

problem in $\mathbb{G}$, denoted as $\text{CDHadv}[\mathcal{A}, \mathbb{G}]$, as the probability that $\hat{w} = w$.

Definition 2.10. The **Computational Diffie-Hellman Assumption** holds in the group $\mathbb{G}$ if for any efficient adversary $\mathcal{A}$ the advantage $\text{CDHadv}[\mathcal{A}, \mathbb{G}]$ is negligible.

2.1.5 Decisional Diffie-Hellman

Now, we loosen the requirements a bit. The **Decisional Diffie-Hellman** (DDH) problem states that given $g^{\alpha}, g^{\beta}, g^{\alpha\beta}$ it is "hard" to distinguish $g^{\alpha\beta}$ from a random element in $\mathbb{G}$. Formally, we define this problem as follows.

Definition 2.11. Let $\mathbb{G}$ be a cyclic group of prime order r generated by $g \in \mathbb{G}$. Define the following game:

1. Both challenger $\mathcal{C}h$ and adversary $\mathcal{A}$ take a description $\mathbb{G}$ as an input: order r and generator $g \in \mathbb{G}$.
2. $\mathcal{C}h$ computes $\alpha, \beta, \gamma \xleftarrow{R} \mathbb{Z}_r, u \leftarrow g^{\alpha}, v \leftarrow g^{\beta}, w_0 \leftarrow g^{\alpha\beta}, w_1 \leftarrow g^{\gamma}$. Then, $\mathcal{C}h$ flips a coin $b \xleftarrow{R} \{0, 1\}$ and sends u, v, w_b to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ outputs the predicted bit $\hat{b} \in \{0, 1\}$.

We define $\mathcal{A}$'s **advantage in solving the Decisional Diffie-Hellman problem in** $\mathbb{G}$, denoted as $\text{DDHadv}[\mathcal{A}, \mathbb{G}]$, as

$$\text{DDHadv}[\mathcal{A}, \mathbb{G}] := \left| \Pr\left[b = \hat{b}\right] - \frac{1}{2} \right|$$

Now, let us break this assumption for some quite generic group! Consider the following example.

Theorem 2.12. Suppose that $\mathbb{G}$ is a cyclic group of an even order. Then, the Decision Diffie-Hellman Assumption does not hold in $\mathbb{G}$. In fact, there is an efficient adversary $\mathcal{A}$ that can distinguish $g^{\alpha\beta}$ from a random element in $\mathbb{G}$ with an advantage $1/4$.

Proof. If $|\mathbb{G}| = 2n$ for $n \in \mathbb{N}$, it means that we can split the group into two subgroups of order n, say, $\mathbb{G}_1$ and $\mathbb{G}_2$. The first subgroup consists of elements in a form g^{2k}, while the second subgroup consists of elements in a form g^{2k+1}.

Now, if we could efficiently determine, based on group element $g \in \mathbb{G}$, whether $g \in \mathbb{G}_1$ or $g \in \mathbb{G}_2$, we essentially could solve the problem. Fortunately, there is such a method! Consider the following lemma.

Lemma 2.13. Suppose $u = g^{\alpha}$. Then, α is even if and only if $u^n = 1$.

Proof. If α is even, then $\alpha = 2\alpha'$ and thus

$$u^n = (g^{2\alpha'})^n = g^{2n\alpha'} = (g^{2n})^{\alpha'} = 1^{\alpha'} = 1$$

Conversely, if $u^n = 1$ then $u^{\alpha n} = 1$, meaning that $2n \mid \alpha n$, implying that α is even. Lemma is proven.

Now, we can construct our adversary $\mathcal{A}$ as follows. Suppose $\mathcal{A}$ is given (u, v, w). Then,

1. Based on u, get the parity of α, say $p_{\alpha} \in \{\text{even}, \text{odd}\}$.
2. Based on v, get the parity of β, say $p_{\beta} \in \{\text{even}, \text{odd}\}$.
3. Based on w, get the parity of γ, say $p_{\gamma} \in \{\text{even}, \text{odd}\}$.
4. Calculate $p'_{\gamma} \in \{\text{even}, \text{odd}\}$ — parity of $\alpha\beta$.
5. Return $\hat{b} = 0$ if $p'_{\gamma} = p_{\gamma}$, and $\hat{b} = 1$, otherwise.

Suppose γ is indeed $\alpha \times \beta$. Then, condition $p'_{\gamma} = p_{\gamma}$ will always hold. If γ is a random element, then the probability that $p'_{\gamma} = p_{\gamma}$ is $1/2$. Therefore, the probability that $\mathcal{A}$ will guess the bit b correctly is $3/4$, and the advantage is $1/4$ therefore. ■

Why is this necessary? Typically, it is impossible to prove the predicate "for every efficient adversary $\mathcal{A}$ this probability is negligible" and therefore we need to make assumptions, such as the Discrete Logarithm Assumption or the Computational Diffie-Hellman Assumption. In turn, proving the statement "if X is secure then Y is also secure" is manageable and does not require solving any fundamental problems. So, for example, knowing that the probability of the adversary to break the Diffie-Hellman assumption is negligible, we can prove that the Diffie-Hellman key exchange is secure.

2.2 Basic Number Theory

As mentioned earlier, the cryptography must work with integer-based formats. One of the earliest and most fundamental branches of mathematics is number theory, which deals with the properties of the integer set $\mathbb{Z}$. Although, as we will see later, we will primarily need the notion of *prime/finite field* further, the basic concepts of number theory will still be used extensively nonetheless.

2.2.1 Introduction to number theory

We start with the most basic definition of number theory — *divisibility*.

Definition 2.14. An integer a is divisible by a non-zero integer b, denoted $b \mid a$ (or *b is a divisor of a*), if and only if there exists an integer $k \in \mathbb{Z}$ such that $a = k \cdot b$.

Let us consider some basic properties of this relation.

Lemma 2.15 (Divisibility properties)**.** For all $a, b, c \in \mathbb{Z}$:

1. $1 \mid a$ (any number is divisible by 1)
2. If $a \neq 0$, then $a \mid a$ (any non-zero integer divides itself).
3. If $a \neq 0$, then $a \mid 0$ (any non-zero integer divides 0).
4. If $b \mid a$ and $c \mid b$, then $c \mid a$ (formally called *transitivity*).
5. $b \mid a \Leftrightarrow b \cdot c \mid a \cdot c$ for any $c \neq 0$.
6. If $c \mid a$ and $c \mid b$, then $c \mid (\alpha \cdot a \pm \beta \cdot b)$, for any $\alpha, \beta \in \mathbb{Z}$

But what happens if a number a is not divisible by the given integer b, meaning there is no integer k that satisfies the condition $a = b \cdot k$? In such cases, a new theorem comes into play.

Theorem 2.16 (Division theorem)**.**

$$\forall a, b \in \mathbb{Z}\ \exists! q, r \in \mathbb{Z} \text{ such that } a = b \cdot q + r \text{ with } 0 \leq r < |b|$$

To prove this theorem, all we need to prove is the existence and uniqueness of such a decomposition. To not make the book too long, we will not prove this theorem here, but you can find the proof in any number theory textbook.

This theorem allows us to define two new operations. Suppose a, b, q, r are given as in the Theorem 2.16. Then,

- **Floor Operation** ($\lfloor a/b \rfloor$ or $a \text{ div } b$) is defined as q. This operation is a standard `div` opeartion commonly used in programming languages.
- **Mod Operation** ($a \bmod b$) is defined as r. This operation is a standard `mod` operation commonly used in programming languages.

In division operations, it is common to check for any shared factors between two numbers. This is where the concept of the **greatest common divisor** (or **gcd** for short) comes into play.

Definition 2.17 (GCD)**.** For any $a, b \in \mathbb{Z}$, the **greatest common divisor** $\gcd(a, b)$ is defined as an integer $d \in \mathbb{N}$ such that:

1. $d \mid a$ and $d \mid b$.

2. d is a maximal integer that satisfies the first condition.

One might right the above definition more concisely:

$$\gcd(a,b) = \max\{d \in \mathbb{N} : d \mid a \text{ and } d \mid b\}.$$

Definition 2.18. Two numbers a and b are **coprime** if and only if $\gcd(a,b) = 1$.

As with any previous concept, let us check some basic properties of the GCD operation.

Lemma 2.19 (Greatest common divisor properties)**.**

1. $\gcd(a,b) = b \Leftrightarrow b \mid a$
2. If $a \neq 0$, then $\gcd(a,0) = a$
3. If there exists $\delta \in \mathbb{Z}$ such that $\delta \mid a$ and $\delta \mid b$, then $\delta \mid \gcd(a,b)$
4. If $c > 0$, then $\gcd(ac,bc) = c \cdot \gcd(a,b)$
5. Suppose $d = \gcd(a,b)$. Then, $\gcd(a/d, b/d) = 1$

Lemma 2.20. For any $a, b \in \mathbb{Z}$, we have $\gcd(a,b) = \gcd(b, a-b)$.

The aforementioned lemma will be further extensively used for the Euclidean algorithm.

Corollary 2.21. As a corollary, for any $a, b \in \mathbb{Z}$ we also have $\gcd(a, b) = \gcd(b, a \bmod b)$.

All these properties are interesting and useful from the theory standpoint, but you may wonder how to practically find $\gcd(a,b)$ (say, if you were to implement this function in the programming language). *Euclidean algorithm* is an efficient method for computing the greatest common divisor. The main idea of Euclidean algorithm is to recursively apply the Corollary 2.21. The concrete implementation in Python is specified below.

```python
def gcd(a: int, b: int) -> int:
    return gcd(b, a % b) if b != 0 else a
```

Notice that the algorithm can be easily implemented in a single line.

While the greatest common divisor (gcd) focuses on finding the largest shared factor between two numbers, the least common multiple (lcm) deals with finding the smallest multiple that both numbers have in common. The LCM is particularly useful when we need to synchronize cycles or work with fractions.

Definition 2.22 (LCM)**.** For any $a, b \in \mathbb{Z}$, the **least common multiple** $\operatorname{lcm}(a, b)$ is defined as an integer $m \in \mathbb{N}$ such that:

1. $a \mid m$ and $b \mid m$
2. m is a minimal integer that satisfies the first condition

One might right the above definition more concisely:

$$\operatorname{lcm}(a, b) = \min\{m \in \mathbb{N} : a \mid m \text{ and } b \mid m\}.$$

Lemma 2.23 (Least Common Multiple Properties)**.**

1. We assume that $\operatorname{lcm}(a, 0)$ is undefined.
2. $\operatorname{lcm}(a, b) = a \Leftrightarrow b \mid a$.
3. If a and b are coprime, then $\operatorname{lcm}(a, b) = a \cdot b$.
4. Any common divisor δ of a and b satisfies $\delta \mid \operatorname{lcm}(a, b)$.
5. For any $c > 0$, we have $\operatorname{lcm}(a \cdot c, b \cdot b) = c \cdot \operatorname{lcm}(a, b)$.
6. Integers $\operatorname{lcm}(a, b)/a$ and $\operatorname{lcm}(a, b)/b$ are coprime.

Theorem 2.24. For any $a, b \in \mathbb{N}$, we have $\gcd(a, b) \cdot \operatorname{lcm}(a, b) = ab$.

One interpretation of the above theorem is that no additional algorithm is required for $\operatorname{lcm}(a, b)$ if we already have an algorithm for $\gcd(a, b)$. Indeed, we can simply use the formula $\operatorname{lcm}(a, b) = ab/\gcd(a, b)$ with the previously computed $\gcd(a, b)$.

The reasonable question is how to generalize the gcd and lcm operations to more than two arguments. For that reason, we provide the following algorithm:

Definition 2.25 (GCD and LCM for multiple arguments)**.** We define the **greatest common divisor** $\gcd(a_1, a_2, \ldots, a_n)$ and the **least common multiple** $\operatorname{lcm}(a_1, a_2, \ldots, a_n)$ for any set of integers $a_1, a_2, \ldots, a_n \in \mathbb{Z}$ as follows:

$$\gcd(a_1, a_2, \ldots, a_n) = \max\{d \in \mathbb{N} : d \mid a_1, d \mid a_2, \ldots, d \mid a_n\}.$$
$$\operatorname{lcm}(a_1, a_2, \ldots, a_n) = \min\{m \in \mathbb{N} : a_1 \mid m, a_2 \mid m, \ldots, a_n \mid m\}.$$

Theorem 2.26 (Computational Properties Of GCD and LCM For Multiple Arguments.)**.** The following two statements are true:

- $\forall a, b \in \mathbb{N} : \gcd(a, b, c) = \gcd(\gcd(a, b), c) = \gcd(a, \gcd(a, b))$.
- $\forall a, b \in \mathbb{N} : \operatorname{lcm}(a, b, c) = \operatorname{lcm}(\operatorname{lcm}(a, b), c) = \operatorname{lcm}(a, \operatorname{lcm}(a, b))$.

In conclusion, from these two theorems there is no necessity for specific algorithms for gcd and lcm when dealing with many arguments. There are more specialized algorithms for each when considering a specific number of arguments, but unfortunately, such topics are beyond the scope of this book.

2.2.2 Extended Euclidean algorithm

In this section, we will introduce the Extended Euclidean Algorithm and an important lemma related to the GCD. You might have reasonable question: why do we even need an extended version? One of the primary reasons is that this algorithm will help in finding inverse modular elements, introduced in the subsequent sections.

Lemma 2.27 (Bezout identity)**.** For any two given integers $a, b \in \mathbb{N}$ with $d = \gcd(a, b)$ there exists such $u, v \in \mathbb{Z}$ that $d = au + bv$.

Corollary 2.28 (From Bezout identity)**.**

1. Integers u and v are of different signs (excluding the case when either $u = 0$ or $v = 0$).
2. *Generalization for multiple integers:* Suppose $d = \gcd(a_1, a_2, \dots, a_n)$, then there exist such integers $u_1, u_2, \dots, u_n \in \mathbb{Z}$ that $d = u_1a_1 + u_2a_2 + \dots + u_na_n$.

The integers u and v are called **Bezout coefficients**. The first corollary can be understood intuitively: if all coefficients are non-negative, the result will be much larger than necessary. Similarly, if they are non-positive, the result must be negative, but the GCD was defined to be positive. The second consequence follows from the fact that we can decompose gcd, thus sequentially derive this sequence. Also note that we can find Bezout coefficients on each Euclidean algorithm step.

Now we introduce the **extended Euclidean algorithm.** The extended Euclidean algorithm finds the Bezout coefficients together with the GCD efficiently.

Algorithm 1: Extended Euclidean algorithm

```
Input   : $a, b \in \mathbb{N}$. Without loss of generality, $a \geq b$
Output : $(\gcd(a, b), u, v)$
$r_0 \leftarrow a$; $r_1 \leftarrow b$
$u_0 \leftarrow 1$; $u_1 \leftarrow 0$
$v_0 \leftarrow 0$; $v_1 \leftarrow 1$
while $r_{i+1} \neq 0,\ i = 1, 2, \ldots$ do
    $q_i \leftarrow r_{i-1}$ div $r_i$
    $u_{i+1} \leftarrow u_{i-1} - u_i q_i$
    $v_{i+1} \leftarrow v_{i-1} - v_i q_i$
    $r_{i+1} \leftarrow a u_{i+1} + b v_i$
end
return $(r_i, u_i, v_i)$
```

The time complexity of the Extended Euclidean algorithm is $O(\log a \log b)$ bit operations, which is very efficient. Although we already know how the Extended Euclidean Algorithm works, this method is still not very human-friendly. For that reason, let us consider an example of an easy way to find the GCD.

Example (Extended Euclidean algorithm example)**.**
First, we you need to find $d = \gcd(a, b)$. Then, knowing the sequence of expansions, find the Bezout coefficients. Note that this can be done simultaneously.

1. $125 = 93 \cdot 1 + 32$
2. $93 = 32 \cdot 2 + 29$
3. $32 = 29 \cdot 1 + 3$
4. $29 = 3 \cdot 9 + 2$
5. $3 = 2 \cdot 1 + 1$
6. $2 = 1 \cdot 2 + 0$

i	0	1	2	3	4	5	6
q_i	×	1	2	1	9	1	2
u_i	1	0	1	−2	3	−29	32
v_i	0	1	−1	3	−4	**?**	**?**

Here, each corresponding cell is calculated with the following formula:

$$u_{i-1} - q_i u_i = u_{i+1}$$
$$v_{i-1} - q_i v_i = v_{i+1}$$

Knowing that, try to finish this example by filling in the missed cells marked by **?**. After finding v_6, be sure to check yourself.

2.2.3 Prime numbers

Prime numbers are fundamental in mathematics due to their role of the building blocks of all natural numbers. Every integer greater than 1 can be uniquely factored into primes, a concept known as the *Fundamental Theorem of Arithmetic*, which we introduce in the next section. This property makes primes central to number theory, and they play a crucial role in various mathematical proofs and structures.

Definition 2.29. Number $n \in \mathbb{N}$ is **prime** if and only if its only two divisors are 1 and n.

Definition 2.30. Number $n \in \mathbb{N}$ is **composite** iff there exists an integer $a \in \mathbb{N}$, which is not 1 or n, for which $a \mid n$. In other words, it is not prime.

Remark. One might ask: what to do with the number 1? We consider it to be neither prime nor composite.

Lemma 2.31. For all $n \in \mathbb{N}$, we have $\gcd(n, n+1) = 1$

Theorem 2.32 (Euclidean theorem)**.** If $P = \{p_1, p_2, p_3, \dots p_k\} \subset \mathbb{N}$ is a finite set, consisting of prime numbers, then there exists a prime number p such that $p \notin P$.

Corollary 2.33. The set of prime numbers has an infinite cardinality.

Lemma 2.34. For all $n \in \mathbb{N}$, where n is composite, if there exists a minimal divisor $d > 1$ of n, then d is a prime number.

Let's say we have a large number and we need to find out if it is prime, how do we do it? In other words, are there any methods for checking a number for prime? Yes, there are, starting with logical reasoning, you can check numbers with brute force, although this method is not practically applicable. There are probabilistic prime tests that will fail with some error. In 2003, an effective deterministic test of simplicity was found, which moved the problem to the class P.

It is worth to be mentioned, there are also different forms of primes that find their application in some fields. For example, Mersenne primes, factorial primes, Euclidean primes, Fibonacci primes, and many other types of primes. We will consider one of the most famous types of primes — Mersenne primes.

Definition 2.35 (Mersenne primes)**.** The prime number of form $2^p - 1$ is called **Mersenne prime**, where p is a prime number.

Mersenne primes, are important in both number theory and cryptography. They have unique mathematical properties that make them useful for testing primality and generating large prime numbers. Mersenne primes are also crucial in the construction of efficient algorithms for error correction in coding theory and for generating random numbers in cryptographic applications, etc. Beside, next theorem importan says that we can know the form of a prime numbers.

Theorem 2.36 (Dirichlet's theorem)**.** For any $a, b \in \mathbb{N}$ if $\gcd(a, b) = 1$, then infinite primes number form of $am + b$ exists, where $m \in \mathbb{N}$.

In other words, every infinite arithmetic progression whose first term and difference are positive integers contains an infinite number of primes.

2.2.4 Fundamental Theorem of Arithmetic

The Fundamental Theorem of Arithmetic states that every integer greater than 1 can be uniquely factored into primes, which is crucial for understanding the structure of numbers. It plays a key role in areas like number theory, cryptography, and simplifying calculations involving divisibility. But before formal description of the theorem, for better understanding it is good to know the following Lemma 2.37.

Lemma 2.37 (Euclidean)**.** If p is a prime and $p \mid ab$, then $p \mid a$ or $p \mid b$.

Proof. ▶ Let $p \mid ab$, but $p \nmid a$, then $\gcd(a, p) = 1$ because there are no common divisors. In which case, by Bezout identity Lemma 2.27, there exist such $u, v \in \mathbb{Z}$ that $au + pv = 1$. Let's multiply the left and right sides by b: $abu + pbv = b$, but $p \mid ab$ and $p \mid pb$, therefore their sum is also divisible by $p \mid abu + pbv$. ◀

Now, we are ready to introduce the central Number Theory theorem — Fundamental Theorem of Arithmetic.

Theorem 2.38 (Fundamental theorem of arithmetic)**.** Any integer $n > 1$ can be decomposed in the unique way into a product of prime numbers:

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_t^{\alpha_t} = \prod_{j=1}^{t} p_j^{\alpha_j},$$

where $p_1, \dots, p_t$ are prime numbers and $\alpha_1, \dots, \alpha_t \in \mathbb{N}$.

Proof. ▶ The theorem state equation then need to proof equation existence and uniqueness. Since the existence property is easy to prove, let us make the small exception and show it. For other proves (in particular, for the uniqueness case, see literature)

Existence. Suppose the theorem statement is false and there exist n that do not have such a representation. Let n_0 be the smallest of them. If n_0 is prime, it can be represented as $p_1 = n_0$, $\alpha_1 = 1$, which satisfies the representation. Therefore, n_0 must be composite, which means $\exists a, b \in \mathbb{N}, 1 < a, b < n_0$ such that $n = ab$. Since n_0 is the smallest non-decomposable number and $a, b < n_0$, then a and b can be represented as

$$a = p_1^{\alpha_1} \dots p_t^{\alpha_t}$$
$$b = q_1^{\beta_1} \dots q_k^{\beta_k}.$$

Therefore, $n_0 = ab = p_1^{\alpha_1} \dots p_t^{\alpha_t} q_1^{\beta_1} \dots q_k^{\beta_k}$. Thus, the contradiction. ◀

Corollary 2.39. Suppose n is decomposed as in Theorem 2.38. Then,

1. If $d \mid n$, then $d = p_1^{\beta_1} p_2^{\beta_2} \dots p_t^{\beta_t}$, where $0 \le \beta_i \le \alpha_i$.
2. Suppose $a = p_1^{\alpha_1} \dots p_t^{\alpha_t}$ and $b = p_1^{\beta_1} \dots p_t^{\beta_t}$. Then, the GCD can be evaluated as $\gcd(a, b) = \prod_{i=0}^{t} p^{\min\{\alpha_i, \beta_i\}}$.
3. Suppose $a = p_1^{\alpha_1} \dots p_t^{\alpha_t}$ and $b = p_1^{\beta_1} \dots p_t^{\beta_t}$. Then, the LCM can be evaluated as $\operatorname{lcm}(a, b) = \prod_{i=0}^{t} p^{\max\{\alpha_i, \beta_i\}}$.
4. If $b \mid a$ and $c \mid a$ and $\gcd(b, c) = 1$, then $bc \mid a$.

The implications and applications of the Fundamental Theorem of Arithmetic are very large and important, and a number of "obvious" ones are listed above. However, you should also be aware that the problem of factorization, i.e., knowing the decomposition of a number into prime factors, is in a NP class (complexity) and is the basis of some cryptosystems, although it is gradually being abandoned, including due to the potential of quantum computers.

2.2.5 Modular arithmetic

Now we are ready for practical applications of number theory, starting with modulo congruence. In this section, we will review the idea of congruence, learn how to use it, and explore its key properties. Understanding these properties will help simplify calculations and solve problems more efficiently in fields such as cryptography, algorithms, and number theory.

Definition 2.40. Two integers $a, b \in \mathbb{Z}$ are said to be **congruent** modulo $n \in \mathbb{N}$ (congruence modulo n is denoted as $a \equiv b \pmod{n}$), if one of the following conditions is met:

1. There exists such $t \in \mathbb{Z}$ that $a = b + nt$
2. $a \bmod n = b \bmod n$
3. $n \mid (a - b)$

It is fairly easy to see that all conditions are equivalent to each other. Next, as always, it is necessary to formally describe the properties, although some of them may seem intuitive, they need to be formally defined.

Lemma 2.41 (Reflexivity, Symmetry, and Transitivity)**.**

1. $a \equiv a \pmod{n}$
2. If $a \equiv b \pmod{n}$, then $b \equiv a \pmod{n}$
3. If $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$, then $a \equiv c \pmod{n}$

The lemma states three basic properties of congruence modulo n: reflexivity, symmetry, and transitivity. These properties confirm that congruence modulo n is an equivalence relation (for general equivalence relation definition, see Section 5.1).

Lemma 2.42. Suppose we have $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then we have

1. $a \pm c \equiv b \pm d \pmod{n}$
2. $ac \equiv bd \pmod{n}$

In turn, this lemma states that we can perform addition, subtraction, and multiplication on the congruent numbers modulo n similarly to the usual arithmetic.

Lemma 2.43. If $ca \equiv cb \pmod{n}$ and $\gcd(c, n) = 1$, then $a \equiv b \pmod{n}$.

This means that we can cancel out the same left and right parts respectively, but with a certain requirement. You may ask why this is necessary, the answer is in the following example.

Example. Let us try to simplify the following equation: $6 \equiv 2 \pmod 4$. Suppose we divide both sides by 2, then we have the false statement $3 \equiv 1 \pmod 4$. That being said, the requirement $\gcd(c, n) = 1$ is mandatory.

Remark. In particular, the example above shows why the division operation is fundamentally different from the regular real/rational numbers. We will see that, in fact, the arithmetic modulo n is not always what mathematicians call a **field**. You will see more details in Section 4.

Lemma 2.44. Modulo congruence can be scaled in both directions:

1. If $k \neq 0, a \equiv b \pmod n$, then $ak \equiv bk \pmod{nk}$
2. If $d = \gcd(a, b, n)$ and $a = a_1 d, b = b_1 d, n = n_1 d, a \equiv b \pmod n$, then $a_1 = b_1 \pmod{n_1}$

Lemma 2.45. If $d \mid n$ and $a \equiv b \pmod n$, then $a \equiv b \pmod d$.

This property is very convenient when it comes to large calculations, if you know its decomposition. It allows you to significantly reduce them, and then restore the result by a large modulus.

Lemma 2.46. Suppose $a \equiv b \pmod{n_1}$, $a \equiv b \pmod{n_2}$, ..., $a \equiv b \pmod{n_k}$. Then, the following statement is true:

$$a \equiv b \pmod{\operatorname{lcm}(n_1, n_2, \ldots, n_k)}.$$

Lemma 2.47. If $a \equiv b \pmod n$, then $\gcd(a, n) = \gcd(b, n)$

Definition 2.48. The congruence class or residues of k modulo n is defined as a set $k + n\mathbb{Z} = \{k + nt \mid t \in \mathbb{Z}\}$

Definition 2.49. The **complete residue system modulo** (or residue ring) n is a set of integers, where every integer is congruent to a unique member of the set modulo n, usually denoted as $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$.

Remark. Sometimes, one might also encounter the notation $\mathbb{Z}/n\mathbb{Z}$, which is more frequently used in Abstract Algebra.

The aforementioned lemmas and definitions describe the properties of addition, subtraction, and multiplication. But what about division? To define the division (if such operation is valid at all), we need to consider the so-called *modular multiplicative inverse*, which is usually denoted as $a^{-1} \pmod{n}$, similarly to real/rational numbers.

Definition 2.50. A modular multiplicative inverse of an integer $a \in \mathbb{Z}$ modulo $n \in \mathbb{N}$ is such an integer a^{-1} that satisfies $a \cdot a^{-1} \equiv a^{-1}a \equiv 1 \pmod{n}$.

Remark. (**Caution**): not all numbers have an inverse number!

Remark. The inverse (if exists) behaves similarly to the usual inverse over rations/reals. For example, $(a^2)^{-1} = (a^{-1})^2$, which means, we can first find the inverse, then the squared value, and vice versa.

The question then is when does the number have the inverse? Consider the following theorem.

Theorem 2.51. Modular inverse $a^{-1} \pmod{n}$ exists if and only if $\gcd(a, n) = 1$.

Based on Extended Euclidean Algorithm 1 and Theorem 2.51, we build the algorithm that can verify an existence and find the inverse value.

Algorithm 2: Modular multiplicative inverse algorithm

```
Input  : $a, n$
Output : $a^{-1}$
1 $(d, u, v) \leftarrow$ ExtendedEuclideanAlgorithm$(a, n)$ /* See Algorithm 1 */
2 if $d \neq 1$ then
3     return inverse does not exist.
4 end
5 return $u$
```

Remark. Note that the complexity of this algorithm is the same as for Algorithm 1, which is $O(\log a \log n)$ bit operations.

Definition 2.52. The **multiplicative group of integers** modulo n, denoted as $\mathbb{Z}_n^\times$, is a set of natural numbers that are coprime to n and less than n. In other words, $\mathbb{Z}_n^\times = \{a \in \mathbb{N} : \gcd(a, n) = 1\}$.

The $\mathbb{Z}_n^\times$ is a fundamental object in number theory and plays a crucial role in cryptography, forming the basis almost for every cryptographic primitive.

Definition 2.53. Euler's Totient Function $\varphi(n)$ is the cardinality of the multiplication group of integers $\mathbb{Z}_n^\times$. In other words, $\varphi(n) = |\mathbb{Z}_n^\times|$.

Remark. The alternative Euler's totient function intuition is following: $\varphi(n)$ counts all coprimes with n in range $[1, n]$.

Lemma 2.54 (Euler's totient function properties)**.**

1. $\varphi(1) = 1$.
2. $\varphi(p) = p - 1$, p where is prime.
3. $\varphi(pq) = \varphi(p) \cdot \varphi(q)$, where p, q are primes.
4. $\varphi(p^\alpha) = p^\alpha - p^{\alpha-1}$, where p is prime.

Corollary 2.55. For any number $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_t^{\alpha_t}$ general formula for Euler's totient function is:

$$\varphi(n) = \prod_{i=1}^{t} \left(p_i^{\alpha_i} - p_i^{\alpha_i - 1}\right) = n \prod_{i=1}^{t} \left(1 - \frac{1}{p_i}\right).$$

Example.

- $\varphi(107) = 106$, because 107 is prime.
- $\varphi(123) = \varphi(3 \cdot 41) = \varphi(3)\varphi(41) = 2 \cdot 40 = 80$
- $\varphi(729) = \varphi(3^6) = 3^6 - 3^5 = 486$

2.2.6 Chinese Remainder Theorem

The Chinese Remainder Theorem is a fundamental result in number theory that provides an efficient method for solving congruence systems. It allows to decompose a complex problem with large integers into several simpler problems with smaller modules. This decomposition reduces the required computational complexity, especially when working with large numbers, and is widely used in areas such as modular arithmetic, cryptography, and algorithmic number theory.

Theorem 2.56 (Chinese Remainder Theorem)**.** Suppose there is the following system

$$\begin{cases} x \equiv x_1 \pmod{n_1} \\ x \equiv x_2 \pmod{n_2} \\ x \equiv x_3 \pmod{n_3} \\ \cdots \\ x \equiv x_t \pmod{n_t} \end{cases}$$

where the n_i are pairwise coprime and $x_1, x_2, \cdots x_t$ are fixed numbers. Then there is unique solution in modulo $N = n_1 n_2 \cdots n_t$.

In fact, we can easily find such solution. Let $N_i = N/n_i$ since n_i are pairwise coprime one, for all i has $\gcd(N_i, n_i) = 1$, and by the Extended Euclidean Algorithm, there exists an integer a_i such that $a_i N_i \equiv 1 \pmod{n_i}$. Thus, the solution is given by

$$x = a_1 N_1 x_1 + a_2 N_2 x_2 + \cdots + a_t N_t x_t.$$

This expression provides the unique solution modulo N, where $N = n_1 n_2 \cdots n_t$. Beside, using this idea, we can write an efficient algorithm to compute x iteratively.

2.2.7 Euler's Theorem

Theorem 2.57 (Euler's)**.** For all $n \in \mathbb{N}$ and $a \in \mathbb{Z}_n^\times$ holds $a^{\varphi n} \equiv 1 \pmod{n}$.

▶ To prove this fact, we prove the following auxiliary lemma.

Lemma 2.58. Suppose $a \in \mathbb{Z}_n^\times$. Denote by $a\mathbb{Z}_n^\times = \{ax : x \in \mathbb{Z}_n^\times\}$. Then, $\mathbb{Z}_n^\times = a\mathbb{Z}_n^\times$. In other words, elementwise multiplication by a permutes the elements of $\mathbb{Z}_n^\times$.

Proof. Our statement is equivalent to claiming that the function $f : \mathbb{Z}_n^\times \to a\mathbb{Z}_n^\times$ defined as $f(x) = ax \pmod{n}$ is a **bijection**. To show this, we need to prove the following three staments:

1. **Correctness** is obvious since ax is in $a\mathbb{Z}_n^\times$ if x is in $\mathbb{Z}_n^\times$.
2. **Injectivity**: we need to prove that if $f(x_1) = f(x_2)$ for two $x_1 \neq x_2 \in \mathbb{Z}_n^\times$, then $x_1 = x_2$. From the function definition, we have $ax_1 = ax_2 \pmod{n}$. Following the Lemma 2.44, we can cancel a modulo n as a is coprime to n. Thus, $x_1 = x_2$.
3. **Surjectivity**: We need to prove that every element $y \in a\mathbb{Z}_n^\times$ has at least

one pre-image $f^{-1}(y)$. Indeed, in this case we have $ax \equiv y \pmod{n}$. Then, set $x := a^{-1}y \pmod{n}$. This expression is well-defined as a is coprime to n and has an inverse.

Therefore, f is bijective and thus $\mathbb{Z}_n^\times = a\mathbb{Z}_n^\times$.

So we know $a\mathbb{Z}_n^\times = \mathbb{Z}_n^\times$, how do we proceed? Notice that from the set equality follows the equality of the products of all elements. Let us write this down:

$$\prod_{i=1}^{\varphi(n)} x_i \equiv \prod_{i=1}^{\varphi(n)} ax_i \pmod{n} \implies \prod_{i=1}^{\varphi(n)} x_i \equiv a^{\varphi(n)} \prod_{i=1}^{\varphi(n)} x_i \pmod{n}$$

Since all x_i are coprime to n by definition, we can cancel out the product of all x_i from both sides, leading to the conclusion of Euler's theorem. ◀

Corollary 2.59. If p is a prime and a is not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$.

This result is commonly known as Fermat's Little Theorem. However, to maintain generality, we state the theorem in a more general form below.

Theorem 2.60 (Fermat's Little Theorem.)**.** Let p is prime number. For any integer a, the following holds $a^p \equiv a \pmod{p}$.

Note that Euler's theorem is a generalization of Fermat's Little Theorem.

2.2.8 Schwartz-Zippel Lemma

Lemma 2.61. Let $\mathbb{F}$ be a field. Let $f(x_1, x_2, ..., x_n)$ be a polynomial of total degree d. Suppose that f is not the zero polynomial. Let S be a finite subset of $\mathbb{F}$. Let $r_1, r_2, ...r_n$ be chosen at random uniformly and independently from S. Then the probability that $f(r_1, r_2, ..., r_n) = 0$ is $\leq \frac{d}{|S|}$.

Example. Let $F = \mathbb{F}_3$, $f(x) = x^2 - 5x + 6$, $S = F$, $r \xleftarrow{R} \mathbb{F}_3$.
Schwartz-Zippel lemma says that the probability that $f(r) = 0$ is $\leq \frac{2}{3}$.

Given two polynomials P, Q with degree d in a field $\mathbb{F}_p$, for $r \xleftarrow{R} \mathbb{F}_3$: $\Pr[P(r) == Q(r)] \leq \frac{d}{p}$. For large fields, where $\frac{d}{p}$ is negligible, this property allows to succinctly check the equality of polynomials. Let $H(x) := P(x) - Q(x)$. Than for each $P(x) = Q(x) \to H(x) = 0$. Applying Schwartz-Zippel lemma, the probability of $H(x) = 0$ for $x \xleftarrow{R} \mathbb{F}$ is $\leq \frac{d}{|S|}$.

2.3 Exercises

Exercise 1. Suppose that for the given cipher with a security parameter λ, the adversary $\mathcal{A}$ can deduce the least significant bit of the plaintext from the ciphertext. Recall that the advantage of a bit-guessing game is defined as $\mathrm{SSAdv}[\mathcal{A}] = \left|\Pr\left[b = \hat{b}\right] - \frac{1}{2}\right|$, where b is the randomly chosen bit of a challenger, while $\hat{b}$ is the adversary's guess. What is the maximal advantage of $\mathcal{A}$ in this case?

Hint: The adversary can choose which messages to send to challenger to further distinguish the plaintexts.

a) 1
b) $\frac{1}{2}$
c) $\frac{1}{4}$
d) 0
e) Negligible value (negl(λ)).

Exercise 2. Consider the cipher $\mathcal{E} = (E, D)$ with encryption function $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ over the message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and key space $\mathcal{K}$. We want to define the security that, based on the cipher, the adversary $\mathcal{A}$ cannot restore the message (*security against message recovery*). For that reason, we define the following game:

1. Challenger chooses random $m \overset{R}{\leftarrow} \mathcal{M}, k \overset{R}{\leftarrow} \mathcal{K}$.
2. Challenger computes the ciphertext $c \leftarrow E(k, m)$ and sends to $\mathcal{A}$.
3. Adversary outputs $\hat{m}$, and wins if $\hat{m} = m$.

We say that the cipher $\mathcal{E}$ is secure against message recovery if the **message recovery advantage**, denoted as $\mathrm{MRadv}[\mathcal{A}, \mathcal{E}]$ is negligible. Which of the following statements is a valid interpretation of the message recovery advantage?

a) $\mathrm{MRadv}[\mathcal{A}, \mathcal{E}] := \left|\Pr[m = \hat{m}] - \frac{1}{2}\right|$
b) $\mathrm{MRadv}[\mathcal{A}, \mathcal{E}] := |\Pr[m = \hat{m}] - 1|$.
c) $\mathrm{MRadv}[\mathcal{A}, \mathcal{E}] := \Pr[m = \hat{m}]$
d) $\mathrm{MRadv}[\mathcal{A}, \mathcal{E}] := \left|\Pr[m = \hat{m}] - \frac{1}{|\mathcal{M}|}\right|$

Exercise 3. Suppose that f and g are negligible functions. Which of the following functions is not neccessarily negligible?

a) $f + g$
b) $f \times g$
c) $f - g$
d) f/g
e) $h(\lambda) := \begin{cases} 1/f(\lambda) & \text{if } 0 < \lambda < 100000 \\ g(\lambda) & \text{if } \lambda \geq 100000 \end{cases}$

Exercise 4. Suppose that $f \in \mathbb{F}_p[x]$ is a d-degree polynomial with d **distinct** roots in $\mathbb{F}_p$. What is the probability that, when evaluating f at n random points, the polynomial will be zero at all of them?

a) Exactly $(d/p)^n$.

b) Strictly less that $(d/p)^n$.

c) Exactly nd/p.

d) Exactly d/np.

Exercise 5-6. To demonstrate the idea of Reed-Solomon codes, consider the toy construction. Suppose that our message is a tuple of two elements $a, b \in \mathbb{F}_{13}$. Consider function $f : \mathbb{F}_{13} \to \mathbb{F}_{13}$, defined as $f(x) = ax + b$, and define the encoding of the message (a, b) as $(a, b) \mapsto (f(0), f(1), f(2), f(3))$.

Question 5. Suppose that you received the encoded message $(3, 5, 6, 9)$. Which number from the encoded message is corrupted?

a) First element (3).

b) Second element (5).

c) Third element (6).

d) Fourth element (9).

e) The message is not corrupted.

Question 6. Consider the previous question. Suppose that the original message was (a, b). Find the value of $a \times b$ (in $\mathbb{F}_{13}$).

a) 4 b) 6 c) 12 d) 2 e) 1

3 Basics of Linear Algebra

3.1 Linear Algebra Basics

Although linear algebra is out of the main scope of this course, familiarity with its basic definitions and concepts is still essential (especially when considering Linear PCPs further from **??**). In this section, we provide a brief review of the key principles that are used throughout the course. You can skip this section without any doubts if you are already familiare with that, and come back to it whenever you need to refresh your memory.

3.1.1 Vector Space

Similarly to group theory working with *groups*, the linear algebra also has a special designated primitive — **vector space**. If previously we were working with the (finite) field $\mathbb{F}$, now we will work with the vector space V over this field. In many practical applications, vector space is formed by **vectors** consisting of a finite fixed collection of elements from the field $\mathbb{F}$. For example, the vector space might be simply $\mathbb{F}^n$: the set of all n-tuples $(x_1, x_2, \dots, x_n)$ of elements from $\mathbb{F}$. Yet, let us give a bit more general definition.

Definition 3.1. A **vector space** V over the field $\mathbb{F}$ is an abelian group for addition $+$ together with a scalar multiplication operation $\cdot$ from $\mathbb{F} \times V$ to V, sending $(\lambda, x) \mapsto \lambda x$ and such that for any $\mathbf{v}, \mathbf{u} \in V$ and $\lambda, \mu \in \mathbb{F}$ we have:

- $\lambda(\mathbf{u} + \mathbf{v}) = \lambda\mathbf{u} + \lambda\mathbf{v}$
- $(\lambda + \mu)\mathbf{v} = \lambda\mathbf{v} + \mu\mathbf{v}$
- $(\lambda\mu)\mathbf{v} = \lambda(\mu\mathbf{v})$
- $1\mathbf{v} = \mathbf{v}$

Any element $\mathbf{v} \in V$ is called a **vector**, and any element $\lambda \in \mathbb{F}$ is called a **scalar**. We also mark vector elements in boldface.

Example. For example, $V = \mathbb{F}^n$ with operations defined as:

$$\lambda \cdot (x_1, x_2, \dots, x_n) = (\lambda x_1, \lambda x_2, \dots, \lambda x_n)$$
$$(x_1, x_2, \dots, x_n) + (y_1, y_2, \dots, y_n) = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$$

is a vector space. Similarly, the following three sets V_1, V_2, V_3 with operations defined above are also valid vector spaces:

$$V_1 = \{(x_1, x_2, \dots, x_n) \in \mathbb{F}^n : x_1 = 0\}$$
$$V_2 = \{(x_1, x_2, \dots, x_n) \in \mathbb{F}^n : x_1 - x_3 = 0\}$$
$$V_3 = \{(x_1, x_2, \dots, x_n) \in \mathbb{F}^n : x_1 + x_2 + \dots + x_n = 0\}$$

Reasoning. Let us see why, for example, V_3 is a valid vector space. Since all operations are inherited from $\mathbb{F}^n$, we only need to check that V_3 is closed under addition and scalar multiplication. Suppose $\mathbf{v}_1 = (x_1, \dots, x_n), \mathbf{v}_2 = (y_1, \dots, y_n) \in V_3$. Then:

$$\mathbf{v}_1 + \mathbf{v}_2 = (x_1 + x_2, \dots, x_n + y_n)$$

Note that $\sum_{i=1}^n x_i + y_i = \left(\sum_{i=1}^n x_i\right) + \left(\sum_{i=1}^n y_i\right) = 0 + 0 = 0$, so the sum is indeed closed.

We will also need to define one very important notion: the linear dependence and independence of vectors.

Definition 3.2. A set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\}$ in a vector space V over a field $\mathbb{F}$ is said to be **linearly independent** if the only scalars $\lambda_1, \lambda_2, \dots, \lambda_k \in \mathbb{F}$ that satisfy the equation

$$\lambda_1\mathbf{v}_1 + \lambda_2\mathbf{v}_2 + \cdots + \lambda_k\mathbf{v}_k = \mathbf{0}$$

are $\lambda_1 = \lambda_2 = \cdots = \lambda_k = 0$. If there exist scalars, not all zero, that satisfy this equation, then the vectors are said to be **linearly dependent**.

Example. Consider the vectors $\mathbf{v}_1 = (1, 0, 0)$, $\mathbf{v}_2 = (0, 1, 0)$, and $\mathbf{v}_3 = (0, 0, 1)$ in $\mathbb{R}^3$. These vectors are linearly independent because the only solution to

$$\lambda_1\mathbf{v}_1 + \lambda_2\mathbf{v}_2 + \lambda_3\mathbf{v}_3 = \mathbf{0}$$

is $\lambda_1 = \lambda_2 = \lambda_3 = 0$.

Example. Consider the vectors $\mathbf{u}_1 = (1, 2, 3)$, $\mathbf{u}_2 = (2, 4, 6)$ in $\mathbb{R}^3$. These vectors are linearly dependent because $\mathbf{u}_2 = 2\mathbf{u}_1$, so the equation

$$\lambda_1\mathbf{u}_1 + \lambda_2\mathbf{u}_2 = \mathbf{0}$$

has non-trivial solutions, such as $\lambda_1 = 2$ and $\lambda_2 = -1$.

3.1.2 Matrix

Besides vectors, frequently we are working with **matrices**. In the most basic sense, the matrix is a rectangular array of numbers, symbols, or expressions, arranged in rows and columns. For example, the matrix A with m rows and n columns, consisting of elements from the finite field $\mathbb{F}$ is denoted as $A \in \mathbb{F}^{m \times n}$.

Additionally, we use notation $A = \{a_{i,j}\}_{i,j=1}^{m\times n}$ to denote the square matrix A of size $m \times n$ with elements $a_{i,j}$. Now, let us define operations on matrices.

Definition 3.3. Let A, B be two matrices over the field $\mathbb{F}$. The following operations are defined:

- **Matrix addition/subtraction**: $A \pm B = \{a_{i,j} \pm b_{i,j}\}_{i,j=1}^{m\times n}$. The matrices A and B must have the same size $m \times n$.
- **Scalar multiplication**: $\lambda A = \{\lambda a_{i,j}\}_{1\le i,j \le n}$ for any $\lambda \in \mathbb{F}$.
- **Matrix multiplication**: $C = AB$ is a matrix $C \in \mathbb{F}^{m\times p}$ with elements $c_{i,j} = \sum_{\ell=1}^{n} a_{i,\ell} b_{\ell,j}$. The number of columns in A must be equal to the number of rows in B, that is $A \in \mathbb{F}^{m\times n}$ and $B \in \mathbb{F}^{n\times p}$.

Example. Suppose $\mathbb{F} = \mathbb{R}$. Then, consider

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2\times 3}, \quad B = \begin{bmatrix} 2 & 1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 2}$$

We cannot add A and B since they have different sizes. However, we can multiply them:

$$AB = \begin{bmatrix} 5 & 6 \\ 7 & 9 \end{bmatrix}, \quad BA = \begin{bmatrix} 4 & 4 & 5 \\ 7 & 7 & 5 \\ 3 & 3 & 3 \end{bmatrix}$$

To see why, for example, the upper left element of AB is 5, we can calculate it as $\sum_{\ell=1}^{3} a_{1,\ell} b_{\ell,1} = 1 \times 2 + 1 \times 1 + 2 \times 1 = 5$.

Remark. Now, we add a very important remark. It just so happens that when working with vectors, we usually assume that they are **column vectors**. This means that the vector $v = (v_1, v_2, \dots, v_n)$ is represented as a matrix:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

This is a common convention in linear algebra, and we will use it in the following sections.

Sometimes we may derive some rows or columns of the matrix from the others. There is a very important notion related to this.

Definition 3.4. The **rank** of a matrix $A \in \mathbb{F}^{m \times n}$ is the maximum number of linearly independent rows or columns in A. This is also known as the **row rank** or **column rank** of the matrix.

Remark. The row rank and column rank of a matrix are always equal.

Example. Consider the matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. To determine the rank of A, we can perform certain linear rows (or columns) permutations to show that the rows (or columns) are linearly independent vectors. For example, we can perform the following operations:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \xrightarrow{R_2 \leftarrow R_2 - 3R_1} \begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix} \xrightarrow{R_2 \leftarrow -\frac{1}{2}R_2} \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \xrightarrow{R_1 \leftarrow R_1 - 2R_2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Since we can clearly see that the rows (and columns) are linearly independent, the rank of A is 2.

Example. Consider the matrix $B = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$. Let's perform the following operations:

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \xrightarrow{R_2 \leftarrow R_2 - 2R_1} \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}$$

Here we can see that the second row is a linear combination of the first row, so the rank of B is 1.

One important operation we will be frequently working with is the **transpose** of the matrix. The transpose of a matrix is an operator that flips a matrix over its diagonal, that is, it switches the row and column indices of the matrix by producing another matrix denoted as $A^\top$.

Definition 3.5 (Transposition)**.** Given a matrix $A \in \mathbb{F}^{m \times n}$, the **transpose** of A is a matrix $A^\top \in \mathbb{F}^{n \times m}$ with elements $A^\top_{ij} = A_{ji}$.

Example. For example, consider the square matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. Then, the transpose of A is $A^\top = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$.

Finally, is just happens that we can construct matrix from the vectors. Therefore, let us introduce the corresponding notation.

Definition 3.6 (Composing Matrix from vectors)**.** Suppose we are given n vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n \in \mathbb{F}^m$. Then, we might define matrix A as a matrix with columns $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ as follows:

$$A = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_n \end{bmatrix} = \begin{bmatrix} v_{1,1} & v_{2,1} & \dots & v_{n,1} \\ v_{1,2} & v_{2,2} & \dots & v_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ v_{1,m} & v_{2,m} & \dots & v_{n,m} \end{bmatrix}$$

Alternatively, vectors might be represented as rows, and the matrix A might be defined as a matrix with rows $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$:

$$A = \begin{bmatrix} \mathbf{v}_1^\top \\ \mathbf{v}_2^\top \\ \vdots \\ \mathbf{v}_n^\top \end{bmatrix} = \begin{bmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,m} \\ v_{2,1} & v_{2,2} & \dots & v_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n,1} & v_{n,2} & \dots & v_{n,m} \end{bmatrix}$$

Example. For example, consider the vectors $\mathbf{v}_1 = (1, 2, 3)$ and $\mathbf{v}_2 = (4, 5, 6)$. Then, the matrix A with columns $\mathbf{v}_1$ and $\mathbf{v}_2$ is:

$$A = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Similarly, the matrix B with rows $\mathbf{v}_1$ and $\mathbf{v}_2$ is:

$$B = \begin{bmatrix} \mathbf{v}_1^\top \\ \mathbf{v}_2^\top \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

3.1.3 Inner Product

Definition 3.7. Consider the vector space $\mathbb{F}^n$. The **inner product** is a function $\langle \cdot, \cdot \rangle : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$ satisfying the following conditions for all $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{F}^n$:

- $\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle + \langle \mathbf{v}, \mathbf{w} \rangle$.
- $\langle \mathbf{u}, \mathbf{v} + \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{v} \rangle + \langle \mathbf{u}, \mathbf{w} \rangle$.

- $\langle \mathbf{u}, \mathbf{v} \rangle = 0$ for all $\mathbf{u} \in \mathbb{F}^n$ iff $\mathbf{v} = \mathbf{0}$.
- $\langle \mathbf{u}, \mathbf{v} \rangle = 0$ for all $\mathbf{v} \in \mathbb{F}^n$ iff $\mathbf{u} = \mathbf{0}$.

Plenty of functions can be built that satisfy the inner product definition, we will use the one that is usually called **dot product**.

Definition 3.8. Consider the vector space $\mathbb{F}^n$. The **dot product** on $\mathbb{F}^n$ is a function $\langle \cdot, \cdot \rangle : \mathbb{V} \times \mathbb{V} \to \mathbb{F}$, defined for every $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$ as follows:

$$\langle \mathbf{u}, \mathbf{v} \rangle := \mathbf{u}^\top \mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

Alternatively, the dot product can also be denoted using the dot notation as $\mathbf{u} \cdot \mathbf{v}$. That is why it is called the “dot“ product.

Example. Let $\mathbf{u}, \mathbf{v}$ are vectors over the real number $\mathbb{R}$, where

$$\mathbf{u} = (1, 2, 3), \quad \mathbf{v} = (2, 4, 3)$$

Then:

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{3} u_i v_i = 2 \cdot 1 + 2 \cdot 4 + 3 \cdot 3 = 2 + 8 + 9 = 19$$

3.1.4 Hadamard Product

Yet another product we are going to use is the **Hadamard product**. Let us see how it works.

Definition 3.9. Suppose $A, B \in \mathbb{F}^{m \times n}$. The **Hadamard product** $A \odot B$ gives a matrix C such that $C_{i,j} = A_{i,j} B_{i,j}$. Essentially, we multiply elements elementwise.

Example. Consider $A = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 0 & 3 \end{bmatrix}$, $B = \begin{bmatrix} 3 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix}$. Then, the Hadamard product is:

$$A \odot B = \begin{bmatrix} 1 \cdot 3 & 1 \cdot 2 & 2 \cdot 1 \\ 3 \cdot 0 & 0 \cdot 2 & 3 \cdot 1 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 2 \\ 0 & 0 & 3 \end{bmatrix}$$

3.1.5 Outer Product

The final product we want to introduce is the **outer product** and some of its properties.

Definition 3.10. Given two vectors $\mathbf{u} \in \mathbb{F}^n$, $\mathbf{v} \in \mathbb{F}^m$ the **outer product** is a matrix whose entries are all products of an element in the first vector with an element in the second vector:

$$\mathbf{u} \otimes \mathbf{v} := \mathbf{u}\mathbf{v}^\top = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \cdots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \cdots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \cdots & u_m v_n \end{bmatrix}$$

Lemma 3.11 (Properties of outer product)**.** For any scalar $c \in \mathbb{F}$ and $(\mathbf{u}, \mathbf{v}, \mathbf{w}) \in \mathbb{F}^n \times \mathbb{F}^m \times \mathbb{F}^p$:

- Transpose: $(\mathbf{u} \otimes \mathbf{v}) = (\mathbf{v} \otimes \mathbf{u})^\top$
- Distributivity: $\mathbf{u} \otimes (\mathbf{v} + \mathbf{w}) = \mathbf{u} \otimes \mathbf{v} + \mathbf{u} \otimes \mathbf{w}$
- Scalar Multiplication: $c(\mathbf{v} \otimes \mathbf{u}) = (c\mathbf{v}) \otimes \mathbf{u} = \mathbf{v} \otimes (c\mathbf{u})$
- Rank: the outer product $\mathbf{u} \otimes \mathbf{v}$ is a rank-1 matrix if $\mathbf{u}$ and $\mathbf{v}$ are non-zero vectors

Example. Let $\mathbf{u}, \mathbf{v}$ are vectors over the real number $\mathbb{R}$, where

$$\mathbf{u} = (1, 2, 3), \quad \mathbf{v} = (2, 4, 3)$$

Then:

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^\top = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \end{bmatrix} = \begin{bmatrix} 1 \cdot 2 & 1 \cdot 4 & 1 \cdot 3 \\ 2 \cdot 2 & 2 \cdot 4 & 2 \cdot 3 \\ 3 \cdot 2 & 3 \cdot 4 & 3 \cdot 3 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 3 \\ 4 & 8 & 6 \\ 6 & 12 & 9 \end{bmatrix}$$

Additionally, as we can see the rows number 2 and 3 in the result matrix can be represented as a linear combination of the first row, specifically by multiplying it by 2 and 3, respectively. The same property applies to the columns. This demonstrates the property of the outer product, that the resulting matrix has a rank of 1.

4 Field Extensions ands Elliptic Curves

4.1 Finite Field Extensions

4.1.1 General Definition

Previously, our discussion resolved around the finite field $\mathbb{F}_p$ for a prime p. However, many protocols need more than just a prime field. For example, elliptic curve pairings and certain STARK constructions require extending $\mathbb{F}_p$ to, in a sense, the analogous of complex numbers.

From school and, possibly, university, you might remember how complex numbers $\mathbb{C}$ are constructed. You take two real numbers, say, $x, y \in \mathbb{R}$, introduce a new symbol i satisfying $i^2 = -1$, and define the complex number as $z = x + iy$. In certain cases, one might encounter a bit more rigorous and abstract definition of complex numbers as the set of pairs $(x, y) \in \mathbb{R}^2$ where addition in naturally defined as $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$, and the multiplication is:

$$(x_1, y_1) \cdot (x_2, y_2) = (x_1x_2 - y_1y_2, x_1y_2 + x_2y_1)^5.$$

In spite of what interpretation you have seen, the complex number is just a tuple of two real numbers that satisfy a bit different rules of multiplication (since addition is typically defined in the same way). What is even more important to us, is that $\mathbb{C}$ is our first example of the so-called **field extension** of $\mathbb{R}$.

Formally, definition of the field extension is very straightforward:

Definition 4.1. Let $\mathbb{F}$ be a field and $\mathbb{K}$ be another field. We say that $\mathbb{K}$ is an **extension** of $\mathbb{F}$ if $\mathbb{F} \subset \mathbb{K}$ and we denote it as $\mathbb{K}/\mathbb{F}$.

Despite just a simplicity of the definition, the field extensions are a very powerful tool in mathematics. But first, let us consider a few non-trivial examples of field extensions.

Example. Denote by $\mathbb{Q}(\sqrt{2}) = \{x + y\sqrt{2} : x, y \in \mathbb{Q}\}$. This is a field extension of $\mathbb{Q}$. It is obvious that $\mathbb{Q} \subset \mathbb{Q}(\sqrt{2})$, but why is $\mathbb{Q}(\sqrt{2})$ a field? Addition and multiplication operations are obviously closed:

$$(x_1 + y_1\sqrt{2}) + (x_2 + y_2\sqrt{2}) = (x_1 + x_2) + (y_1 + y_2)\sqrt{2},$$
$$(x_1 + y_1\sqrt{2}) \cdot (x_2 + y_2\sqrt{2}) = (x_1x_2 + 2y_1y_2) + (x_1y_2 + x_2y_1)\sqrt{2}.$$

[5] Notice that $(x_1 + iy_1)(x_2 + iy_2) = x_1x_2 + iy_2x_1 + iy_1x_2 + i^2y_1y_2 = (x_1x_2 - y_1y_2) + (x_1y_2 + x_2y_1)i$.

But what about the inverse element? Well, here is the trick:

$$\frac{1}{x+y\sqrt{2}} = \frac{x-y\sqrt{2}}{(x+y\sqrt{2})(x-y\sqrt{2})} = \frac{x-y\sqrt{2}}{x^2-2y^2} =$$
$$= \frac{x}{x^2-2y^2} - \frac{y}{x^2-2y^2}\sqrt{2} \in \mathbb{Q}(\sqrt{2}).$$

Example. Consider $\mathbb{Q}(\sqrt{2}, i) = \{a+bi : a, b \in \mathbb{Q}(\sqrt{2})\}$ where $i^2 = -1$. This is a field extension of $\mathbb{Q}(\sqrt{2})$ and, consequently, of $\mathbb{Q}$. The representation of the element is:

$$(a + b\sqrt{2}) + (c + d\sqrt{2})i = a + b\sqrt{2} + ci + d\sqrt{2}i$$

Showing that this is a field is a bit more tedious, but still straightforward. Suppose we take $\alpha + \beta i \in \mathbb{Q}(\sqrt{2}, i)$ with $\alpha, \beta \in \mathbb{Q}(\sqrt{2})$. Then:

$$\frac{1}{\alpha+\beta i} = \frac{\alpha-\beta i}{\alpha^2+\beta^2} = \frac{\alpha}{\alpha^2+\beta^2} - \frac{\beta}{\alpha^2+\beta^2}i$$

Since $\mathbb{Q}(\sqrt{2})$ is a field, both $\frac{\alpha}{\alpha^2+\beta^2}$ and $\frac{\beta}{\alpha^2+\beta^2}$ are in $\mathbb{Q}(\sqrt{2})$, and, consequently, $\mathbb{Q}(\sqrt{2}, i)$ is a field as well.

Remark. Notice that basically, $\mathbb{Q}(\sqrt{2}, i)$ is just a linear combination of $\{1, \sqrt{2}, i, \sqrt{2}i\}$. This has a very important implication: $\mathbb{Q}(\sqrt{2}, i)$ is a four-dimensional vector space over $\mathbb{Q}$, where elements $\{1, \sqrt{2}, i, \sqrt{2}i\}$ naturally form **basis**. We are not going to use it implicitly, but this observation might make further discussion a bit more intuitive.

Remark. One might have defined $\mathbb{Q}(\sqrt{2}, i) = \{x + \sqrt{2}y : x, y \in \mathbb{Q}(i)\}$ instead. Indeed, $\mathbb{Q}(\sqrt{2})(i) = \mathbb{Q}(i)(\sqrt{2}) = \mathbb{Q}(\sqrt{2}, i)$.

4.1.2 Polynomial Quotient Ring

Now, we present a more general way to construct field extensions. Notice that when constructing $\mathbb{C}$, we used the magical element i that satisfies $i^2 = -1$. But here is another way how to think of it.

Consider the set of polynomials $\mathbb{R}[x]$, then I pick $p(x) := x^2 + 1 \in \mathbb{R}[x]$ and ask you to find roots of $p(x)$. Of course, you would claim "hey, this equation has no solutions over $\mathbb{R}$" and that is totally true. That is why mathematicians introduced a new element i that we formally called the root of $x^2 + 1$. Note however, that i is not a number in the traditional sense, but rather a fictional symbol that we artifically introduced to satisfy the equation.

Now, could we have picked another polynomial, say, $q(x) = x^2 + 4$? Sure! As long as its roots cannot be found in $\mathbb{R}$, we are good to go.

Example. Suppose β is the root of $q(x) := x^2 + 4$. Then we could have defined complex numbers as a set of $x + y\beta$ for $x, y \in \mathbb{R}$. In this case, multiplication, for example, would be defined a bit differently than in the case of $\mathbb{C}$:

$$(x_1 + y_1\beta) \cdot (x_2 + y_2\beta) = (x_1x_2 - 4y_1y_2) + (x_1y_2 + x_2y_1)\beta.$$

We shifted to the polynomial consideration for a reason: now, instead of considering the complex number $\mathbb{C}$ as "some" tuple of real numbers (c_0, c_1), now let us view it as a polynomial[6] $c_0 + c_1X$ modulo polynomial $X^2 + 1$.

Example. Indeed, take, for example, $p_1(X) := 1+2X$ and $p_2(X) := 2+3X$. Addition is performed as we are used to:

$$p_1 + p_2 = (1 + 2X) + (2 + 3X) = 3 + 5X,$$

but multiplication is a bit different:

$$p_1p_2 = (1 + 2X) \cdot (2 + 3X) = 2 + 3X + 4X + 6X^2 = 6X^2 + 7X + 2.$$

Well, and what next? Recall that we are doing arithmetic modulo $X^2 + 1$ and for that reason, we divide the polynomial by $X^2 + 1$:

$$6X^2+7X+2 = 6(X^2+1)+7X-4 \implies (6X^2+7X+2) \bmod (X^2+1) = 7X-4,$$

meaning that $p_1p_2 = 7X - 4$. Oh wow, hold on! Let us come back to our regular complex number representation and multiply $(1 + 2i)(2 + 3i)$. We get $2+3i+4i+6i^2 = -4+7i$. That is exactly the same result if we change X to i above! In fact, what we have observed is the fact that our polynomial quotient ring $\mathbb{R}[X]/(X^2 + 1)$ is isomorphic to $\mathbb{C}$.

So, let us generalize this observation to any field $\mathbb{F}$ and any irreducible polynomial $\mu(x) \in \mathbb{F}[x]$.

Theorem 4.2. Let $\mathbb{F}$ be a field and $\mu(x)$ — irreducible polynomial over $\mathbb{F}$ (sometimes called a **reduction polynomial**). Consider a set of polynomials

[6]Here, we use X to represent the polynomial variable to avoid confusion with the notation $x + yi$.

over $\mathbb{F}[x]$ modulo $\mu(x)$, formally denoted as $\mathbb{F}[x]/(\mu(x))$. Then, $\mathbb{F}[x]/(\mu(x))$ is a field.

Example. As we considered above, let $\mathbb{F} = \mathbb{R}$, $\mu(x) = x^2 + 1$, then $\mathbb{R}[X]/(X^2 + 1)$ (a set of polynomials modulo $X^2 + 1$) is a field.

Example. Suppose $\mathbb{F} = \mathbb{Q}$ and $\mu(x) := x^2 - 2$. Then, $\mathbb{Q}[X]/(X^2 - 2)$ is a field isomorphic to $\mathbb{Q}(\sqrt{2})$, considered above.

Example. Suppose $\mathbb{F} = \mathbb{Q}$ and $\mu(x) := (x^2 + 1)(x^2 - 2) = x^4 - x^2 - 2$. Then, $\mathbb{Q}[X]/(x^4 - x^2 - 2)$ is a field isomorphic to $\mathbb{Q}(\sqrt{2}, i)$.

Remark. Although we have not defined the isomorphism between two rings/-fields, it is defined similarly to group isomorphism. Suppose we have fields $(\mathbb{F}, +, \times)$ and $(\mathbb{K}, \oplus, \otimes)$. Bijective function $\phi : \mathbb{F} \to \mathbb{K}$ is called an isomorphism if it preserves additive and multiplicative structures, that is for all $a, b \in \mathbb{F}$:

$$\phi(a + b) = \phi(a) \oplus \phi(b),$$
$$\phi(a \times b) = \phi(a) \otimes \phi(b).$$

This theorem (aka definition) corresponds to viewing complex numbers as a polynomial quotient ring $\mathbb{R}[X]/(X^2 + 1)$. But, we can give a theorem (aka definition) for our classical representation via magical root i of $x^2 + 1$.

Theorem 4.3. Let $\mathbb{F}$ be a field and $\mu \in \mathbb{F}[X]$ is an irreducible polynomial of degree n and let $\mathbb{K} := \mathbb{F}[X]/(\mu(X))$. Let $\theta \in \mathbb{K}$ be the root of μ over $\mathbb{K}$. Then,

$$\mathbb{K} = \{c_0 + c_1\theta + \cdots + c_{n-1}\theta^{n-1} : c_0, \dots, c_{n-1} \in \mathbb{F}\}$$

Although this definition is quite useful, we will mostly rely on the polynomial quotient ring definition. Let us define the **prime field extension**.

Definition 4.4. Suppose p is prime and $m \geq 2$. Let $\mu \in \mathbb{F}_p[X]$ be an irreducible polynomial of degree m. Then, elements of $\mathbb{F}_{p^m}$ are polynomials in $\mathbb{F}_p^{(\leq m)}[X]$. In other words,

$$\mathbb{F}_{p^m} = \{c_0 + c_1X + \cdots + c_{m-1}X^{m-1} : c_0, \dots, c_{m-1} \in \mathbb{F}_p\},$$

where all operations are performed modulo $\mu(X)$.

Again, let us consider a few examples.

Example. Consider the $\mathbb{F}_{2^4}$. Then, there are 16 elements in this set:

$$\begin{aligned}
&0, 1, X, X+1,\\
&X^2, X^2+1, X^2+X, X^2+X+1,\\
&X^3, X^3+1, X^3+X, X^3+X+1,\\
&X^3+X^2, X^3+X^2+1, X^3+X^2+X, X^3+X^2+X+1.
\end{aligned}$$

One might choose the following reduction polynomial: $\mu(X) = X^4 + X + 1$ (of degree 4). Then, operations are performed in the following manner:

- Addition: $(X^3 + X^2 + 1) + (X^2 + X + 1) = X^3 + X$.
- Subtraction: $(X^3 + X^2 + 1) - (X^2 + X + 1) = X^3 + X$.
- Multiplication: $(X^3 + X^2 + 1) \cdot (X^2 + X + 1) = X^2 + 1$ since:

$$(X^3+X^2+1)\cdot(X^2+X+1) = X^5+X+1 \bmod (X^4+X+1) = X^2+1$$

- Inversion: $(X^3 + X^2 + 1)^{-1} = X^2$ since $(X^3 + X^2 + 1) \cdot X^2 \bmod (X^4 + X + 1) = 1$.

Now, in the subsequent sections, we would need to extend $\mathbb{F}_p$ at least to $\mathbb{F}_{p^2}$. A convenient choice, similarly to the complex numbers, is to take $\mu(X) = X^2+1$. However, in contrast to $\mathbb{R}$, equation $X^2 = -1 \pmod p$ might have solutions over certain prime numbers p. Thus, we consider proposition below.

Proposition 4.5. Let p be an odd prime. Then $X^2 + 1$ is irreducible in $\mathbb{F}_p[X]$ if and only if $p \equiv 3 \pmod 4$.

Corollary 4.6. $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 + 1)$ is a valid prime field extension for odd primes p satisfying $p \equiv 3 \pmod 4$. In this case, extended elements are of the form $c_0 + c_1 u$ where $c_0, c_1 \in \mathbb{F}_p$ and $u^2 = -1$.

4.1.3 Multiplicative Group of a Finite Field

The non-zero elements of $\mathbb{F}_p$, denoted as $\mathbb{F}_p^\times$, form a multiplicative cyclic group. In other words, there exist elements $g \in \mathbb{F}_p^\times$, called *generators*, such that

$$\mathbb{F}_p^\times = \{g^k : 0 \leq k \leq p-2\}$$

The order of $x \in \mathbb{F}_p^\times$ is the smallest positive integer r such that $x^r = 1$. It is also not difficult to show that $r \mid (p-1)$.

Definition 4.7. $\omega \in \mathbb{F}$ is the *primitive root* in the finite field $\mathbb{F}$ if $\langle \omega \rangle = \mathbb{F}^{\times}$.

Example. $\omega = 3$ is the primitive root of $\mathbb{F}_7$. Indeed,

$$3^1 = 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5, 3^6 = 1.$$

So clearly $\langle \omega \rangle = 7$.

In STARKs (and in optimizing operations) for DFT (Discrete Fourier Transform) we would need the so-called nth primitive roots of unity.

Example. For those who studied complex numbers a bit (it is totally OK if you did not, so you might skip this example), recall an equation $\zeta^n = 1$ over $\mathbb{C}$. The solutions are $\zeta_k = \cos\left(\frac{2\pi k}{n}\right) + i \sin\left(\frac{2\pi k}{n}\right)$ for $k \in \{0, 1, \dots, n-1\}$, so one has exactly n solutions (in contrast to $x^n = 1$ over $\mathbb{R}$ where there are at most 2 solutions[a]). For any solution ζ_k, it is true that $\zeta_k^n = 1$, but if one were to consider the subgroup generated by ζ_k (that is, $\{1, \zeta_k, \zeta_k^2, \dots\}$), then not neccecerily $\langle \zeta_k \rangle$ would enumerate all the roots of unity $\{\zeta_j\}_{j=0}^{n-1}$. For that reason, we call ζ_k the nth primitive root of unity if $\langle \zeta_k \rangle$ enumerates all roots of unity. One can show that this is the case if and only if $\gcd(k, n) = 1$. This is always the case for $k = 1$, so commonly mathematicians use ζ_n to denote an expression $\cos \frac{2\pi}{n} + i \sin \frac{2\pi}{n} = e^{2\pi i/n}$.

[a]Think why.

Yet, let us give the broader definition, including the finite fields case.

Definition 4.8. ω is the nth primitive root of unity if $\omega^n = 1$ and $\omega^k \neq 1$ for all $1 \leq k < n$.

Note that such ω exists if and only if $n \mid (p-1)$.

4.1.4 Algebraic Closure

Consider the following interesting question: suppose we have a field $\mathbb{F}$. Is there an extension $\mathbb{K}/\mathbb{F}$ such that $\mathbb{K}$ contains all roots of any polynomial in $\mathbb{F}[X]$? The answer is yes, and such a field is called the **algebraic closure** of $\mathbb{F}$, although not always this algebraic closure has a nice form. But first, let us define what it means for field $\mathbb{F}$ to be algebraically closed.

Definition 4.9. A field $\mathbb{F}$ is called **algebraically closed** if every non-constant polynomial $p(x) \in \mathbb{F}[X]$ has a root in $\mathbb{F}$.

Example. $\mathbb{R}$ is not algebraically closed since $X^2 + 1$ has no roots in $\mathbb{R}$. However, $\mathbb{C}$ is algebraically closed, which follows from the fundamental theorem of algebra. Since $\mathbb{C}$ is a field extension of $\mathbb{R}$, it is also an algebraic closure of $\mathbb{R}$. This is commonly denoted as $\overline{\mathbb{R}} = \mathbb{C}$.

Definition 4.10. A field $\mathbb{K}$ is called an **algebraic closure** of $\mathbb{F}$ if $\mathbb{K}/\mathbb{F}$ is algebraically closed. This is denoted as $\overline{\mathbb{F}} = \mathbb{K}$.

Since we are doing cryptography and not mathematics, we are interested in the algebraic closure of $\mathbb{F}_p$. Well, I have two news for you (as always, one is good and one is bad). The good news is that any finite field $\mathbb{F}_{p^m}$ has an algebraic closure. The bad news is that it does not have a form $\mathbb{F}_{p^k}$ for $k > m$ and there are infinitely many elements in it (so in other words, the algebraic closure of a finite field is not finite). This is due to the following theorem.

Theorem 4.11. No finite field $\mathbb{F}$ is algebraically closed.

Proof. Suppose $f_1, f_2, \dots, f_n \in \mathbb{F}$ are all elements of $\mathbb{F}$. Consider the following polynomial:

$$p(x) = \prod_{i=1}^{n}(x - f_i) + 1 = (x - f_1)(x - f_2)\cdots(x - f_n) + 1.$$

Clearly, $p(x)$ is a non-constant polynomial and has no roots in $\mathbb{F}$, since for any $f \in \mathbb{F}$, one has $p(f) = 1$. ■

But what form does the $\overline{\mathbb{F}}_p$ have? Well, it is a union of all $\mathbb{F}_{p^k}$ for $k \geq 1$. This is formally written as:

$$\overline{\mathbb{F}}_p = \bigcup_{k \in \mathbb{N}} \mathbb{F}_{p^k}.$$

Remark. But this definition is super counter-intuitive! So here how we usually interpret it. Suppose I tell you that polynomial $q(x)$ has a root in $\overline{\mathbb{F}}_p$. What that means is that there exists some extension $\mathbb{F}_{p^m}$ such that for some $\alpha \in \mathbb{F}_{p^m}$, $q(\alpha) = 0$. We do not know how large this m is, but we know that it exists. For that reason, $\overline{\mathbb{F}}_p$ is defined as an infinite union of all possible field extensions.

4.2 Elliptic Curves

4.2.1 Classical Definition

Probably, there is no need to explain the importance of elliptic curves. Essentially, the main group being used for cryptographic protocols is the group of points on an elliptic curve. If elliptic curve is "good enough", then the discrete logarithm problem assumption, Diffie-Hellman assumption and other core cryptographic assumptions hold. Moreover, this group does not require a large field size, which is a huge advantage for many cryptographic protocols.

So, let us formally define what an elliptic curve is. Further assume that, when speaking of the finite field $\mathbb{F}_p$, the underlying prime number is greater than 3.[7]. The definition is the following.

Definition 4.12. Suppose that $\mathbb{K}$ is a field. An **elliptic curve** E over $\mathbb{K}$ is defined as a set of points $(x, y) \in \mathbb{K}^2$:

$$y^2 = x^3 + ax + b,$$

called a **Short Weierstrass equation**, where $a, b \in \mathbb{K}$ and $4a^3 + 27b^2 \neq 0$. We denote $E/\mathbb{K}$ to denote the elliptic curve over field $\mathbb{K}$.

Remark. One might wonder why $4a^3 + 27b^2 \neq 0$. This is due to the fact that the curve $y^2 = x^3 + ax + b$ might have certain degeneracies and special points, which are not desirable for us. So we require this condition to make $E/\mathbb{K}$ "good".

Definition 4.13. We say that $P = (x_P, y_P) \in \mathbb{A}^2(\mathbb{K})$ is the **affine representation** of the point on the elliptic curve $E/\mathbb{K}$ if it satisfies the equation $y_P^2 = x_P^3 + ax_P + b$.

Example. Consider the curve $E/\mathbb{Q} : y^2 = x^3 - x + 9$. This is an elliptic curve. Consider $P = (0, 3), Q = (-1, -3) \in \mathbb{A}^2(\mathbb{Q})$: both are valid affine points on the curve. See Figure 4.1.

Typically, our elliptic curve is defined over a finite field $\mathbb{F}_p$, so we are interested in this paricular case.

Remark. Although, in many cases one might encounter the definition where an elliptic curve E is defined over the algebraic closure of $\mathbb{F}_p$, that is $E/\overline{\mathbb{F}}_p$. This is typically important when considering elliptic curve pairings. However, for the sake of simplicity, we will consider elliptic curves over $\mathbb{F}_p$ and corresponding finite extensions $\mathbb{F}_{p^m}$ as of now.

[7]Note that, for example, for $\mathbb{F}_{2^n}$ equation of elliptic curve is very different, but usually we do not deal with binary field elements.

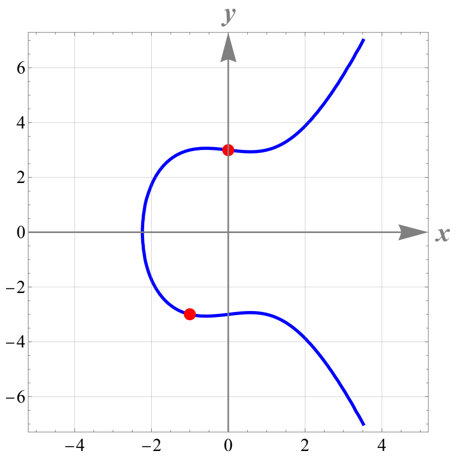


Figure 4.1: Elliptic curve $E/\mathbb{Q} : y^2 = x^3 - x + 9$ with points $P = (0, 3), Q = (-1, -3)$ depicted on it.

Remark. It is easy to see that if $(x, y) \in E/\mathbb{K}$, then $(x, -y) \in E/\mathbb{K}$. We will use this fact intensively further.

Now, elliptic curves are useless without any operation defined on top of them. But as will be seen later, it is quite unclear how to define the identity element. For that reason, we introduce a bit different definition of a set of points on the curve.

Definition 4.14. The set of points on the curve, denoted as $E_{a,b}(\mathbb{K})$, is defined as:

$$E_{a,b}(\mathbb{K}) = \{(x, y) \in \mathbb{A}^2(\mathbb{K}) : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\},$$

where $\mathcal{O}$ is the so-called **point at infinity**.

Remark. The difference between $E(\mathbb{K})$ and $E/\mathbb{K}$ is that the former includes the point at infinity, while the latter does not. We also omit the index a, b, so instead of $E_{a,b}(\mathbb{K})$ we write simply $E(\mathbb{K})$.

Now, the reason we introduced the point at infinity $\mathcal{O}$ is because it allows us to define the group binary operation $\oplus$ on the elliptic curve. The operation is sometimes called the **chord-tangent law**. Let us define it.

Definition 4.15. Consider the curve $E(\mathbb{F}_{p^m})$. We define $\mathcal{O}$ as the identity element of the group. That is, for all points P, we set $P \oplus \mathcal{O} = \mathcal{O} \oplus P = P$.

For any other non-identity elements $P = (x_P, y_P), Q = (x_Q, y_Q) \in E(\mathbb{F}_{p^m})$, define the $P \oplus Q = (x_R, y_R)$ as follows:

1. If $x_P \neq x_Q$, use the **chord method**. Define $\lambda := \frac{y_P - y_Q}{x_P - x_Q}$ — the slope between P and Q. Set the resultant coordinates as:
$$x_R := \lambda^2 - x_P - x_Q, \quad y_R := \lambda(x_P - x_R) - y_P.$$
2. If $x_P = x_Q \wedge y_P = y_Q$ (that is, $P = Q$), use the **tangent method**. Define the slope of the tangent at P as $\lambda := \frac{3x_P^2 + a}{2y_P}$ and set
$$x_R := \lambda^2 - 2x_P, \quad y_R := \lambda(x_P - x_R) - y_P.$$
3. Otherwise, define $P \oplus Q := \mathcal{O}$.

The aforementioned definition is illustrated in the Figure below[8].

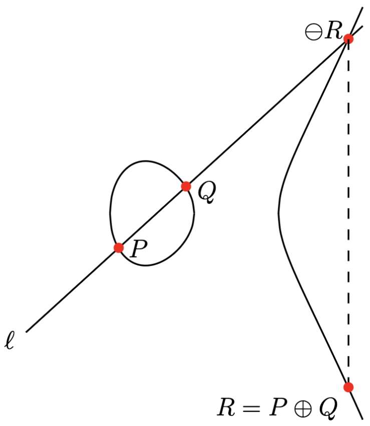


Figure 2.5: Elliptic curve addition.

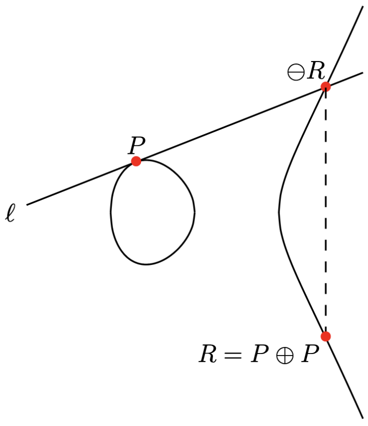


Figure 2.6: Elliptic curve doubling.

Example. Consider $E/\mathbb{R} : y^2 = x^3 - 2x$. The points $(-1,-1), (0,0), (2,2)$ are all on E and also on the line $\ell : y = x$. Therefore, $(-1,1) \oplus (0,0) = (2,-2)$ or, similarly, $(2,2) \oplus (-1,-1) = (0,0)$.
Now, let us compute $[2](-1,-1)$. Calculate the tangent slope as $\lambda := \frac{3\cdot(-1)^2 - 2}{2\cdot(-1)} = -\frac{1}{2}$. Thus, the tangent line has an equation $\ell' : y = -\frac{1}{2}x + c$. Substituting $(-1,-1)$ into the equation, we get $c = -\frac{3}{2}$. Therefore, the

[8]Illustration taken from *"Pairing for Beginners"*

equation of the tangent line is $y = -\frac{1}{2}x - \frac{3}{2}$. The intersection of the curve and the line is $\left(\frac{9}{4}, -\frac{21}{8}\right)$, yielding $[2](-1,-1) = \left(\frac{9}{4}, -\frac{21}{8}\right)$.
The whole illustration is depicted in Figure 4.2.

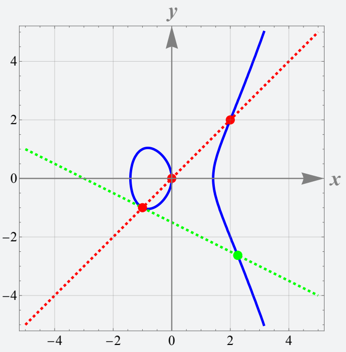


Figure 4.2: Illustration of the group law on the elliptic curve $E/\mathbb{R} : y^2 = x^3 - 2x$. In red we marked points lying on the line $\ell : y = x$. In dashed red, we marked the line ℓ, while in dashed green — the tangent line ℓ' at $(-1,-1)$, which is used to calculate $[2](-1,1)$.

Theorem 4.16. $(E(\mathbb{F}_{p^m}), \oplus)$ forms an abelian group.

Proof Sketch. The identity element is $\mathcal{O}$. Every point $\mathcal{O} \neq P = (x_P, y_P) \in E(\mathbb{F}_{p^m})$ has an additive inverse: indeed, $\ominus P := (x_P, -y_P)$. Finally, a bit of algebra might show that the operation is associative. It is also clearly commutative: even geometrically it is evident, that the result of $P \oplus Q$ does not depend on the order of P and Q ("drawing a line between P and Q" and "drawing a line between Q and P" are equivalent statements). ■

Now, let us talk a bit about the group order. The group order is the number of elements in the group. For elliptic curves, the group order is typically denoted as r or n, but we are going to use r. Also, the following theorem is quite important.

Theorem 4.17. Define $r := |E(\mathbb{F}_{p^m})|$. Then, $r = p^m + 1 - t$ for some integer $|t| \leq 2\sqrt{p^m}$. A bit more intuitive explanation: the number of points on the curve is close to $p^m + 1$. This theorem is commonly called the **Hasse's theorem on elliptic curves**, and the value t is called the **trace of Frobenius**.

Remark. In fact, $r = |E(\mathbb{F}_{p^m})|$ can be computed in $O(\log(p^m))$, so the number of points can be computed efficiently even for fairly large primes p.

Finally, let us define the scalar multiplication operation.

Definition 4.18. Let $P \in E(\mathbb{F}_{p^m})$ and $\alpha \in \mathbb{Z}_r$. Define the scalar multiplication $[\alpha]P$ as:

$$[\alpha]P = \underbrace{P \oplus P \oplus \cdots \oplus P}_{\alpha \text{ times}}.$$

Question. Why do we restrict α to $\mathbb{Z}_r$ and not to $\mathbb{Z}$?

4.2.2 Discrete Logarithm Problem on Elliptic Curves

Finally, as defined in the previous section, the **discrete logarithm** problem on the elliptic curve is the following: typically, $E(\mathbb{F}_p)$ is cyclic, meaning there exist some point $G \in E(\mathbb{F}_p)$, called the **generator**, such that $\langle G \rangle = E(\mathbb{F}_p)$. Given $P \in E(\mathbb{F}_p)$, the problem consists in finding such a scalar $\alpha \in \mathbb{Z}_r$ such that $[\alpha]G = P$.

Now, if the curve is "good", then the discrete logarithm problem is hard. In fact, the best-known algorithms have a complexity $O(\sqrt{r})$. However, there are certain cases when the discrete log problem is much easier.

1. If r is composite, and all its prime factors are less than some bound r_{max}, then the discrete log problem can be solved in $O(\sqrt{r_{\text{max}}})$. For this very reason, typically r is prime.
2. If $|E(\mathbb{F}_p)| = p$, then the discrete logarithm can be solved in polynomial time. These curves are called **anomalous curves**.
3. Suppose that there is some small integer $\tau > 0$ such that $r \mid (p^\tau - 1)$. The discrete log in that case reduces to the discrete log in the finite field $\mathbb{F}_{p^\tau}$, which is typically not hard for small enough τ.

4.3 Exercises

Exercises 4-9. Tower of Extensions

You are given the passage explaining the topic of tower of extensions. The text has gaps that you need to fill in with the correct statement among the provided choices.

This question demonstrates the concept of the so-called **tower of extensions**. Suppose we want to build an extension field $\mathbb{F}_{p^4}$. Of course, we can find some irreducible polynomial $p(X)$ of degree 4 over $\mathbb{F}_p$ and build $\mathbb{F}_{p^4}$ as $\mathbb{F}_p[X]/(p(X))$. However, this method is very inconvenient since implementing the full 4-degree polynomial arithmetic is inconvenient. Moreover, if we were to implement arithmetic over, say, $\mathbb{F}_{p^{24}}$, that would make the matters worse. For this reason, we will build $\mathbb{F}_{p^4}$ as $\mathbb{F}_{p^2}[j]/(q(j))$ where $q(j)$ is an irreducible polynomial of degree 2 over $\mathbb{F}_{p^2}$, which itself is represented as $\mathbb{F}_p[i]/(r(i))$ for some suitable irreducible quadratic polynomial $r(i)$. This way, we can first implement $\mathbb{F}_{p^2}$, then $\mathbb{F}_{p^4}$, relying on the implementation of $\mathbb{F}_{p^2}$ and so on.

For illustration purposes, let us pick $p := 5$. As noted above, we want to build $\mathbb{F}_{5^2}$ first. A valid way to represent $\mathbb{F}_{5^2}$ would be to set $\mathbb{F}_{5^2} := \boxed{\mathbf{4}}$. Given this representation, the zero of a linear polynomial $f(x) = ix - (i + 3)$, defined over $\mathbb{F}_{5^2}$, is $\boxed{\mathbf{5}}$.

Now, assume that we represent $\mathbb{F}_{5^4}$ as $\mathbb{F}_{5^2}[j]/(j^2 - \xi)$ for $\xi = i + 1$. Given such representation, the value of j^4 is $\boxed{\mathbf{6}}$. Finally, given $c_0 + c_1 j \in \mathbb{F}_{5^4}$ we call $c_0 \in \mathbb{F}_{5^2}$ a **real part**, while $c_1 \in \mathbb{F}_{5^2}$ an **imaginary part**. For example, the imaginary part of number $j^3 + 2i^2\xi$ is $\boxed{\mathbf{7}}$, while the real part of $(a_0 + a_1 j)b_1 j$ is $\boxed{\mathbf{8}}$. Similarly to complex numbers, it motivates us to define the number's **conjugate**: for $z = c_0 + c_1 j$, define the conjugate as $\overline{z} := c_0 - c_1 j$. The expression $z\overline{z}$ is then $\boxed{\mathbf{9}}$.

Warmup (Oleksandr in search of perfect field extension)

Exercise 1. Oleksandr decided to build $\mathbb{F}_{49}$ as $\mathbb{F}_7[i]/(i^2+1)$. Compute $(3+i)(4+i)$.

a) $6+i$. b) 6. c) $4+i$. d) 4. e) $2+4i$.

Exercise 2. Oleksandr came up with yet another extension $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2+2)$. He asked interns to calculate $2/i$. Based on five answers given below, help Oleksandr to find the correct one.

a) 1. b) $p-2$. c) $(p-3)i$. d) $(p-1)i$. e) $p-1$.

Exercise 3*. After endless tries, Oleksandr has finally found the perfect field extension: $\mathbb{F}_{p^2} := \mathbb{F}_p[v]/(v^2+v+1)$. However, Oleksandr became very frustrated since not for any p this would be a valid field extension. For which of the following values p such construction would **not** be a valid field extension? Use the fact that equation $\omega^3 = 1$ over $\mathbb{F}_p$ has non-trivial solutions (meaning, two others except for $\omega = 1$) if $p \equiv 1 \pmod 3$. You can assume that listed numbers are primes.

a) 8431. b) 9173. c) 9419. d) 6947.

Exercise 4.

a) $\mathbb{F}_5[i]/(i^2+1)$
b) $\mathbb{F}_5[i]/(i^2+2)$
c) $\mathbb{F}_5[i]/(i^2+4)$
d) $\mathbb{F}_5[i]/(i^2+2i+1)$
e) $\mathbb{F}_5[i]/(i^2+4i+4)$

Exercise 5.

a) $1+i$
b) $1+2i$
c) $1+4i$
d) $2+3i$
e) $3+i$

Exercise 6.

a) $4+2i$
b) $4i$
c) 1
d) $1+2i$
e) $2+4i$

Exercise 7.

a) equal to zero.
b) equal to one.
c) equal to the real part.
d) $2(1+i)$
e) -4

Exercise 8.

a) a_1b_1
b) $a_1b_1\xi$
c) a_0b_1
d) $a_0b_1\xi$
e) a_0a_1

Exercise 9.

a) $c_0^2+c_1^2$
b) $c_0^2-c_1^2\xi$
c) $c_0^2+c_1^2\xi^2$
d) $(c_0^2+c_1^2\xi)j$
e) $(c_0^2-c_1^2)j$

Elliptic Curves

Exercise 10. Suppose that elliptic curve is defined as $E/\mathbb{F}_7 : y^2 = x^3 + b$. Suppose $(2, 3)$ lies on the curve. What is the value of b?

Exercise 11. Sum of which of the following pairs of points on the elliptic curve $E/\mathbb{F}_{11}$ is equal to the point at infinity $\mathcal{O}$ for any valid curve equation?

a) $P = (2, 3), Q = (2, 8)$.
b) $P = (9, 2), Q = (2, 8)$.
c) $P = (9, 9), Q = (5, 7)$.
d) $P = \mathcal{O}, Q = (2, 3)$.
e) $P = [10]G, Q = G$ where G is a generator.

Exercise 12. Consider an elliptic curve E over $\mathbb{F}_{167^2}$. Denote by r the order of the group of points on E (that is, $r = |E|$). Which of the following **can** be the value of r?

a) $167^2 - 5$
b) $167^2 - 1000$
c) $167^2 + 5 \cdot 167$
d) 170^2
e) 160^2

Exercise 13. Suppose that for some elliptic curve E the order is $|E| = qr$ where both q and r are prime numbers. Among listed, what is the most optimal complexity of algorithm to solve the discrete logarithm problem on E?

a) $O(qr)$
b) $O(\sqrt{qr})$
c) $O(\sqrt{\max\{q, r\}})$
d) $O(\sqrt{\min\{q, r\}})$
e) $O(\max\{q, r\})$

5 Projective Coordinates and Pairing

5.1 Relations

Before delving into the projective coordinates and further zero-knowledge topics, let us first discuss the concept of relations, which will be intensively used from now on. Now, what is a relation? The definition is incredibly concise.

Definition 5.1. Let $\mathcal{X}, \mathcal{Y}$ be some sets. Then, $\mathcal{R}$ is a **relation** if

$$\mathcal{R} \subset \mathcal{X} \times \mathcal{Y} = \{(x, y) : x \in \mathcal{X}, y \in \mathcal{Y}\}$$

Interpretation is approximately the following: suppose we have sets $\mathcal{X}$ and $\mathcal{Y}$. Then, relation $\mathcal{R}$ gives a set of pairs (x, y), telling that $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ are *related*.

Example. Let $\mathcal{X} = \{\text{Oleksandr}, \text{Phat}, \text{Anton}\}$ and $\mathcal{Y} = \{\text{Backend}, \text{Frontend}, \text{Research}\}$. Define the following relation of "person x works in field y":

$$\mathcal{R} = \{(\text{Oleksandr}, \text{Research}), (\text{Phat}, \text{Frontend}), (\text{Anton}, \text{Backend})\}$$

Obviously, $\mathcal{R} \subset \mathcal{X} \times \mathcal{Y}$, so $\mathcal{R}$ is a relation.

Remark. There are many ways to express that $(x, y) \in \mathcal{R}$. Most common are $x\mathcal{R}y$ and $x \sim y$. Also, sometimes, one might encounter relation definition as a boolean function $\mathcal{R} : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$, where $\mathcal{R}(x, y)$ is 1 if (x, y) is in the relation, and 0 otherwise.
Further, we will use notation $x \sim y$ to denote that $(x, y) \in \mathcal{R}$.

Example. Let E be a cyclic group of points on the Elliptic Curve of order $r \geq 2$ with a generator $\langle G \rangle = E$. Let $\mathcal{X} = \mathbb{Z}_r$ and $\mathcal{Y} = E$. Define a relation $\mathcal{R} \subset \mathcal{X} \times \mathcal{Y}$ by:

$$\mathcal{R} = \{(\alpha, P) \in \mathbb{Z}_r \times E : [\alpha]G = P\}$$

Essentially, such a relation is a set of secret keys α and corresponding public keys P. In this case, for example, $0\mathcal{R}\mathcal{O}$ and $1\mathcal{R}G$ or $0 \sim \mathcal{O}$ and $1 \sim G$.

Remark. When we say that $\sim$ is a relation on a set $\mathcal{X}$, we mean that $\sim$ is a relation $\mathcal{R}$ on the following Cartesian product: $\mathcal{R} \subset \mathcal{X} \times \mathcal{X}$.

Remark. The provided example is relevant in most cases (ecdsa, eddsa, schnorr signatures etc.). But for some algorithms, the relation between secret key α and public key P can be defined as:

$$\mathcal{R} = \{(\alpha, P) \in \mathbb{Z}_r \times E : \ominus[\alpha]G = P\}$$

for DSTU 4145 standard or even:

$$\mathcal{R} = \{(\alpha, P) \in \mathbb{Z}_r \times E : [\alpha^{-1}]G = P\}$$

for twisted ElGamal algorithm.

Now, let us formally define the term **equivalence relation**.

Definition 5.2. Let $\mathcal{X}$ be a set. A relation $\sim$ on $\mathcal{X}$ is called an **equivalence relation** if it satisfies the following properties:

1. **Reflexivity:** $x \sim x$ for all $x \in \mathcal{X}$.
2. **Symmetry:** If $x \sim y$, then $y \sim x$ for all $x, y \in \mathcal{X}$.
3. **Transitivity:** If $x \sim y$ and $y \sim z$, then $x \sim z$ for all $x, y, z \in \mathcal{X}$.

Example. Let $\mathcal{X}$ be the set of all people. Define a relation $\sim$ on $\mathcal{X}$ by $x \sim y$ if $x, y \in \mathcal{X}$ have the same birthday. Then $\sim$ is an equivalence relation on $\mathcal{X}$. Let us demonstrate that:

1. **Reflexivity:** $x \sim x$ since x has the same birthday as x.
2. **Symmetry:** If $x \sim y$, then $y \sim x$ since x has the same birthday as y.
3. **Transitivity:** If $x \sim y$ and $y \sim z$, then $x \sim z$ since x has the same birthday as y and y has the same birthday as z.

Example. Suppose $\mathcal{X} = \mathbb{Z}$ and n is some fixed integer. Let $a \sim b$ mean that $a \equiv b \pmod{n}$. It is easy to verify that $\sim$ is an equivalence relation:

1. **Reflexivity:** $a \equiv a \pmod{n}$, so $a \sim a$.
2. **Symmetry:** If $a \equiv b \pmod{n}$, then $b \equiv a \pmod{n}$, so $b \sim a$.
3. **Transitivity:** If $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$, then $a \equiv c \pmod{n}$. It is not that obvious, so we can prove it: from the first equality we have $\exists q \in \mathbb{Z} : a - b = nq$. From the second, $\exists r \in \mathbb{Z} : b - c = nr$. Adding both we get $(a - b) + (b - c) = n(r + q)$ or, equivalently, $a - c = n(r + q)$, meaning $a \equiv c \pmod{n}$.

The example below is less obvious with a bit more difficult proof, which we will skip. Yet, it is quite curious, so here it is.

Example. Let $\mathcal{G}$ be the set of all possible groups. Define a relation $\sim$ on $\mathcal{G}$ by $\mathbb{G} \sim \mathbb{H}$ if $\mathbb{G} \cong \mathbb{H}$ (in other words, $\mathbb{G}$ and $\mathbb{H}$ are isomorphic). Then $\sim$ is an equivalence relation.

Now, suppose I give you a set $\mathcal{X}$ with some equivalence relation $\sim$ (say, $\mathcal{X} = \mathbb{Z}$ and $a \equiv b \pmod{n}$). Notice that you can find some subset $\mathcal{X}' \subset \mathcal{X}$ in which all elements are equivalent (and any other element from $\mathcal{X} \setminus \mathcal{X}'$ is not). In the case of modulo relation above, $\mathcal{X}'$ could be the set of all integers that are congruent to 1 modulo n, so $\mathcal{X}' = \{\dots, -n+1, 1, n+1, 2n+1, \dots\}$. This way, we can partition the set $\mathcal{X}$ into disjoint subsets, where all elements in each subset are equivalent. Such subsets are called **equivalence classes**. Now, let us give a formal definition.

Definition 5.3. Let $\mathcal{X}$ be a set and $\sim$ be an equivalence relation on $\mathcal{X}$. For any $x \in \mathcal{X}$, the **equivalence class** of x is the set

$$[x] = \{y \in \mathcal{X} : x \sim y\}$$

The **set of all equivalence classes** is denoted by $\mathcal{X}/\!\sim$ (or, if the relation $\mathcal{R}$ is given explicitly, then $\mathcal{X}/\mathcal{R}$), which is read as "$\mathcal{X}$ modulo relation $\sim$".

Example. Let $\mathcal{X} = \mathbb{Z}$ and n be some fixed integer. Define $\sim$ on $\mathcal{X}$ by $x \sim y$ if $x \equiv y \pmod{n}$. Then the equivalence class of x is the set

$$[x] = \{y \in \mathbb{Z} : x \equiv y \pmod{n}\}$$

For example, $[0] = \{\dots, -2n, -n, 0, n, 2n, \dots\}$ while $[1] = \{\dots, -2n + 1, -n+1, 1, n+1, 2n+1, \dots\}$.

Now, as we have said before, a set of all equivalence classes form a partition of the set $\mathcal{X}$. This means that any element $x \in \mathcal{X}$ belongs to exactly one equivalence class. This is a very important property, which we will use in the next section. Formally, we have the following lemma.

Lemma 5.4. Let $\mathcal{X}$ be a set and $\sim$ be an equivalence relation on $\mathcal{X}$. Then,

1. For each $x \in \mathcal{X}$, $x \in [x]$ (quite obvious, follows from reflexivity).
2. For each $x, y \in \mathcal{X}$, $x \sim y$ if and only if $[x] = [y]$.
3. For each $x, y \in \mathcal{X}$, either $[x] = [y]$ or $[x] \cap [y] = \emptyset$.

Example. Let $n \in \mathbb{N}$ and, again, $\mathcal{X} = \mathbb{Z}$ with a "modulo n" equivalence relation $\mathcal{R}_n$. Define the equivalence class of x by $[x]_n = \{y \in \mathbb{Z} : x \equiv y \pmod{n}\}$. Then,

$$\mathbb{Z}/\mathcal{R}_n = \{[0]_n, [1]_n, [2]_n, \dots, [n-2]_n, [n-1]_n\}$$

forms a partition of $\mathbb{Z}$, that is

$$\bigcup_{i=0}^{n-1} [i]_n = \mathbb{Z},$$

and for all $i, j \in \{0, 1, \dots, n-1\}$, if $i \neq j$, then $[i]_n \cap [j]_n = \emptyset$. Commonly, we denote the set of all equivalence classes as $\mathbb{Z}/n\mathbb{Z}$ or, as we got used to, $\mathbb{Z}_n$. Moreover, we can naturally define the addition as:

$$[x]_n + [y]_n = [x+y]_n$$

Then, the set $(\mathbb{Z}/n\mathbb{Z}, +)$ with the defined addition is a group.

The primary reason we considered equivalence relations is that we will define the projective space as a set of equivalence classes. Besides this, when defining proofs of knowledge, argument of knowledge and zero-knowledge protocols, we will use the concept of relations and equivalence relations intensively.

5.2 Elliptic Curve in Projective Coordinates

5.2.1 Projective Space

Recall that we defined the elliptic curve as

$$E(\overline{\mathbb{F}}_p) := \{(x, y) \in \mathbb{A}^2(\overline{\mathbb{F}}_p) : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$$

The above definition is the definition of the elliptic curve in *the affine space*. However, notice that in this case we need to append a somewhat artificial point $\mathcal{O}$ to the curve. This is done to make the curve a group since without this point it is unclear how to define addition of two, say, negative points on the curve (since the resultant vertical line does not intersect the curve at any other point). The way to unify all the points $E/\overline{\mathbb{F}}_p$ with this magical point at infinity $\mathcal{O}$ is to use the **projective space**.

Essentially, instead of working with points in affine n-space (in our case, with two-dimensional points $\mathbb{A}^2(\mathbb{K})$), we work with lines that pass through the origin in $(n+1)$-dimensional space (in our case, 3-dimensional space $\mathbb{A}^3(\mathbb{K})$). We say that two points from this $(n+1)$-dimensional space are **equivalent** if they lie on the same line that passes through the origin (we will show the illustration a bit later).

It seems strange that we need to work with 3-dimensional space to describe 2-dimensional points, but this is the way to unify all the points on the curve. Because, in this case, the point at infinity is represented by a set of points on the line that passes through the origin and is parallel to the y-axis. We will get to understanding how to interpret that. Moreover, by defining operations on the projective space, we can make the operations on the curve more efficient.

Now, to the formal definition.

Definition 5.5. Projective coordinate, denoted as $\mathbb{P}^2(\mathbb{K})$ (or sometimes simply $\mathbb{KP}^2$) is a triple of elements $(X : Y : Z)$ from $\mathbb{A}^3(\overline{\mathbb{K}}) \setminus \{0\}$ modulo the equivalence relation[a]:

$$(X_1 : Y_1 : Z_1) \sim (X_2 : Y_2 : Z_2) \text{ if and only if exists such } \lambda \in \overline{\mathbb{K}} \text{ that}$$
$$\text{the following holds } (X_1 : Y_1 : Z_1) = (\lambda X_2 : \lambda Y_2 : \lambda Z_2)$$

[a]Although we specify the definition for $n = 2$, the definition can be generalized to any $\mathbb{P}^n(\overline{\mathbb{K}})$.

This definition on itself might be a bit too abstract, so let us consider the concrete example for projective space $\mathbb{P}^2(\mathbb{R})$.

Example. Consider the projective space $\mathbb{P}^2(\mathbb{R})$. Then, two points $(x_1, y_1, z_1), (x_2, y_2, z_2) \in \mathbb{R}^3$ are equivalent if there exists $\lambda \in \mathbb{R}$ such that $(x_1, y_1, z_1) = (\lambda x_2, \lambda y_2, \lambda z_2)$. For example, $(1, 2, 3) \sim (2, 4, 6)$ since $(1, 2, 3) = 0.5(2, 4, 6)$.

Example. Now, how to geometrically interpret $\mathbb{P}^2(\mathbb{R})$? Consider the Figure below.

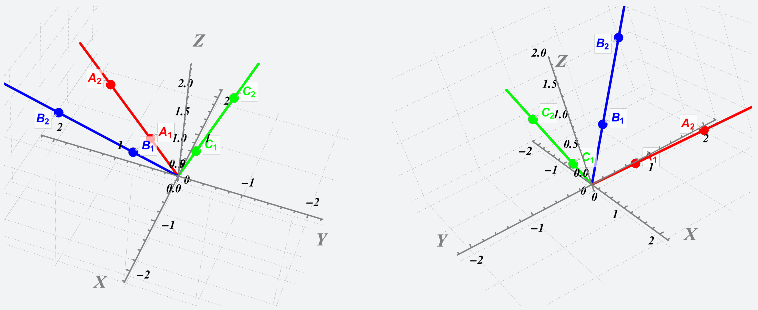


Illustration: Geometric interpretation of $\mathbb{P}^2(\mathbb{R})$, the same scene from different perspectives. The **red** line is represented by equation $(2t, 3t, t)$, **blue** line by $(-2t, 3t, 3t)$, and **green** line is represented by $(t, -2t, 5t)$ for parameter $t \in \mathbb{R}$.

Here, the figure demonstrates three equivalence classes, being a set of points on the **red**, **blue**, and **green** lines (except for the origin).
The reason why geometrically the set of equivalence classes lie on the same line that passes through the origin is following: suppose we have a point $\vec{v}_0 = (x_0, y_0, z_0) \in \mathbb{R}^3$, represented as a vector. Then, the set of all points that are equivalent to (x_0, y_0, z_0) is the set of all points $(\lambda x_0, \lambda y_0, \lambda z_0) = \lambda \vec{v}_0$ for $\lambda \in \mathbb{R} \setminus \{0\}$. So $\vec{v}_0$ is the representative of equivalence class $[\vec{v}_0] = \{\lambda \vec{v}_0 : \lambda \in \mathbb{R}, \lambda \neq 0\}$. Now notice, that this is a parametric equation of a line that passes through the origin and the point $\vec{v}_0$: notice that for $\lambda = 0$ (if we assume that expression is also defined for zero λ) we have the origin $\vec{0}$, while for $\lambda = 1$ we have the point $\vec{v}_0$. Then, any other values of λ in-between $[0, 1]$ or outside define the set of points lying on the same line.

Now, projective coordinates are not that useful unless we can come back to the affine space. This is done by defining the map $\phi : \mathbb{P}^2(\overline{\mathbb{K}}) \to \mathbb{A}^2(\overline{\mathbb{K}})$ as follows: $\phi : (X : Y : Z) \mapsto (X/Z, Y/Z)$. If, in turn, we want to go from the affine space to the projective space, we can define the map $\psi : \mathbb{A}^2(\overline{\mathbb{K}}) \to \mathbb{P}^2(\overline{\mathbb{K}})$ as follows: $\psi : (x, y) \mapsto (x : y : 1)$. Geometrically, map ϕ means that we take a point $(X : Y : Z)$ and project it onto the plane $Z = 1$.

Example. Again, consider three lines from the previous example. Now, we additionally draw a plane $\pi : z = 1$ in our 3-dimensional space (see Illustration below).

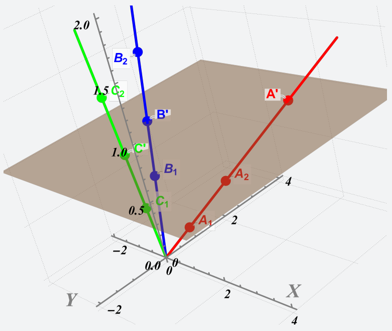


Illustration: Geometric interpretation of converting projective to affine form.

By using the map $(X : Y : Z) \mapsto (X/Z, Y/Z)$, all points on the line get mapped to the itersection of the line with the plane $\pi : z = 1$. This way, for example, points on the **red line** ℓ_{red} get mapped to the point $A' = (2, 3, 1)$, corresponding to $(2, 3)$ in affine coordinates. So, for example, point $(6, 9, 3) \in \ell_{\text{red}}$, lying on the same line, gets mapped to $(6/3, 9/3) = (2, 3)$. Similarly, all **blue line** points get mapped to the point $B' = (-2/3, 1, 1)$, while all **green line** points get mapped to the point $C' = (0.2, -0.4, 1)$[a].

[a]One can verify that based on the equations provided from the previous example

5.2.2 Elliptic Curve Equation in Projective Form

Now, quite an interesting question is following: how to represent (basically, rewrite) the "affine" elliptic curve equation[9]

$$E_{\mathbb{A}}(\overline{\mathbb{F}}_p) : y^2 = x^3 + ax + b, \; a, b \in \overline{\mathbb{F}}_p$$

in the projective form? Since currently, we defined the curve as the 2D curve, but now we are working in 3D space! The answer is following: recall that if $(X : Y : Z) \in \mathbb{P}^2(\overline{\mathbb{F}}_p)$ lies on the curve, so does the point $(X/Z, Y/Z)$. The condition on the latter point to lie on $E_{\mathbb{A}}(\overline{\mathbb{F}}_p)$ is following:

$$\left(\frac{Y}{Z}\right)^2 = \left(\frac{X}{Z}\right)^3 + a \cdot \frac{X}{Z} + b$$

But now multiply both sides by Z^3 to get rid of the fractions:

$$E_{\mathbb{P}}(\overline{\mathbb{F}}_p) : Y^2Z = X^3 + aXZ^2 + bZ^3$$

This is an equation of the elliptic curve in **projective form**.

Now, one of the motivations to work with the projective form was to unify affine points $E_{\mathbb{A}}/\overline{\mathbb{F}}_p$ and the point at infinity $\mathcal{O}$, which acted as an identity element in the group $E_{\mathbb{A}}(\overline{\mathbb{F}}_p)$. So how do we encode the point at infinity in the projective form?

Well, notice the following observation: all points $(0 : \lambda : 0)$ always lie on the curve $E_{\mathbb{P}}(\overline{\mathbb{F}}_p)$. Moreover, the map from the projective form to the affine form is ill-defined for such points, since we would need to divide by zero. So, we can naturally make the points $(0 : \lambda : 0)$ to be the set of points at infinity. This way, we can define the point at infinity as $\mathcal{O} = (0 : 1 : 0)$.

Finally, let us summarize what we have observed so far.

[9]Further, we will use notation $E_{\mathbb{A}}$ to represent the elliptic curve equation in the affine form, and $E_{\mathbb{P}}$ to represent the elliptic curve in the projective form.

Definition 5.6. The **homogenous projective form of the elliptic curve** $E_{\mathbb{P}}(\overline{\mathbb{F}}_p)$ is defined as the set of all points $(X : Y : Z) \in \mathbb{P}^2(\overline{\mathbb{F}}_p)$ in the projective space that satisfy the equation

$$E_{\mathbb{P}}(\overline{\mathbb{F}}_p) : Y^2Z = X^3 + aXZ^2 + bZ^3, \quad a, b \in \overline{\mathbb{F}}_p,$$

where the point at infinity is encoded as $\mathcal{O} = (0 : 1 : 0)$.

Example. Consider the BN254 curve $y^2 = x^3 + 3$ over reals $\mathbb{R}$. Its projective form is given by the equation $Y^2Z = X^3 + 3Z^3$, which gives a surface, depicted below.

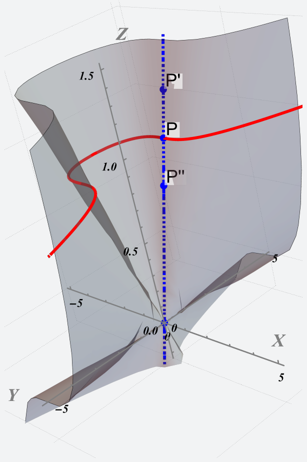


Illustration: BN254 Curve Elliptic Curve in Projective Form over $\mathbb{R}$. In **gray** is the surface, while **red** points are the points on the affine curve (lying on the plane $\pi : z = 1$).

Points $P' \approx (0 : 2.165 : 1.25)$ and $P'' \approx (0 : 1.3 : 0.75)$ in projective form both lie on the curve and get mapped to the same point $P \approx (0, 1.732)$ in affine coordinates.

5.2.3 General Projective Coordinates

Hold on, but why did we use the term *homogenous*? The reason why is because we defined equivalence as follows: $(X : Y : Z) \sim (\lambda X : \lambda Y : \lambda Z)$ for some $\lambda \in \overline{\mathbb{K}}$, called **homogenous coordinates**. However, this is not the only way to define equivalence. Consider a more general form of equivalence relation:

$$(X : Y : Z) \sim (X' : Y' : Z') \text{ iff } \exists \lambda \in \overline{\mathbb{K}} : (X, Y, Z) = (\lambda^n X', \lambda^m Y', \lambda Z')$$

In this case, to come back to the affine form, we need to use the map $\phi : (X : Y : Z) \mapsto (X/Z^n, Y/Z^m)$.

Example. The case $n = 2, m = 3$ is called the **Jacobian Projective Coordinates**. An Elliptic Curve equation might be then rewritten as:

$$Y^2 = X^3 + aXZ^4 + bZ^6$$

The reason why we might want to use such coordinates is that they can be more efficient in some operations, such as point addition. However, we will not delve into this topic much further.

Example. Consider the BN254 curve $y^2 = x^3 + 3$ over reals $\mathbb{R}$, again. Its *Jacobian projective form* is given by the equation $Y^2 = X^3 + 3Z^6$, which gives a surface, depicted below.

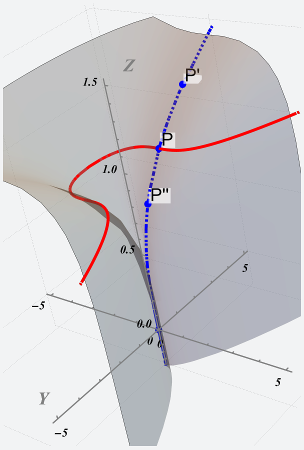


Illustration: BN254 Curve Elliptic Curve in Jacobian Projective Form over $\mathbb{R}$. In **gray** is the surface, while **red** points are the points on the affine curve (lying on the plane $\pi : z = 1$).

Notice that now, under the map $(X : Y : Z) \mapsto (X/Z^2, Y/Z^3)$, points in the same equivalence class (in $\mathbb{R}^3$) do not lie on the same line, but rather on the same *curve*. Namely, equivalence class has a form $[(x_0, y_0, z_0)] = \{t^2 x_0, t^3 y_0, t z_0 : t \in \mathbb{R} \setminus \{0\}\}$.

5.2.4 Fast Addition

Let us come back to the affine case and assume that the underlying field is the prime field $\mathbb{F}_p$. Recall that for adding two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ to get $R = (x_R, y_R) \leftarrow P \oplus Q$ one used the following formulas (there is no need to understand the derivation fully, just take it as a fact):

$$x_R \leftarrow \left(\frac{y_Q - y_P}{x_Q - x_P}\right)^2 - x_P - x_Q, \quad y_R \leftarrow \left(\frac{y_Q - y_P}{x_Q - x_P}\right)(x_P - x_R) - y_P$$

Denote by M the cost of multiplication, by S the cost of squaring, and by I the cost of inverse operation in $\mathbb{F}_p$. Note that we do not count addition/inverse costs as they are significantly lower than operations listed. Then, the cost of additing two points using above formula is 2M+1S+1I. Indeed, our computation can proceed as follows:

1. Calculate $t_1 \leftarrow (x_Q - x_P)^{-1}$, costing 1I.
2. Calculate $\lambda \leftarrow (y_Q - y_P)t_1$, costing 1M.
3. Calculate $t_2 \leftarrow \lambda^2$, costing 1S.
4. Calculate $x_R \leftarrow t_2 - x_P - x_Q$, costing almost nothing.
5. Calculate $y_R \leftarrow \lambda(x_P - x_R) - y_P$, costing 1M.

Well, there are just 4 operations in total, so what can go wrong? The problem is that we need to calculate the inverse of $(x_Q - x_P)$, which is a very, very costly operation. In fact, typically 1I $\gg$ 20M or even worse, the ratio might reach 80 in certain cases.

Now imagine we want to add 4 points, say $P_1 \oplus P_2 \oplus P_3 \oplus P_4$: this costs 6M + 3S + 3I. Now we have 3 inverses, which is a lot. Finally, if we are to add much larger number of points (for example, when finding the scalar product), this gets even worse.

Projective coordinates is a way to solve this problem. The idea is to represent points in the projective form $(X : Y : Z)$, so when adding two numbers in projective form, you still get a point in a form $(X : Y : Z)$. Then, after conducting a series of additions, you can convert the point back to the affine form.

But why adding two points, say, $(X_P : Y_P : Z_P)$ and $(X_Q : Y_Q : Z_Q)$, in the projective form is more efficient? We will not derive the formulas, but trust us that they have the following form:

$$\begin{aligned}
X_R &= (X_PZ_Q - X_QZ_P)(Z_PZ_Q(Y_PZ_Q - Y_QZ_P)^2 - (X_PZ_Q - X_QZ_P)^2(X_PZ_Q + X_QZ_P));\\
Y_R &= Z_PZ_Q(X_QY_P - X_PY_Q)(X_PZ_Q - X_QZ_P)^2 -\\
&\quad - (Y_PZ_Q - Y_QZ_P)((Y_PZ_Q - Y_QZ_P)^2Z_PZ_Q - (X_PZ_Q + X_QZ_P)(X_PZ_Q - X_QZ_P)^2);\\
Z_R &= Z_PZ_Q(X_PZ_Q - X_QZ_P)^3.
\end{aligned}$$

Do not be afraid, you do not need to understand how this formula is derived. But notice that despite the very scary look, there is no inversions involved! Moreover, this formula can be calculated in only $12\mathsf{M} + 2\mathsf{S}$! So all in all, this is much more effective than $2\mathsf{M} + 1\mathsf{S} + 1\mathsf{I}$.

The only inversion which is unavoidable in the projective form is the inversion of Z since after all additions (and doublings) have been made, we need to use map $(X : Y : Z) \mapsto (X/Z, Y/Z)$ to return back to the affine form. However, this inversion is done only once at the end of the computation, so it is not that costly.

Proposition 5.7. To conclude, typically, when working with elliptic curves, one uses the following strategy:

1. Convert affine points to projective form using the map $(x, y) \mapsto (x, y, 1)$.
2. Perform all operations in the projective form, which do not involve inversions.
3. Convert the result back to the affine form using the map $(X : Y : Z) \mapsto (X/Z, Y/Z)$.

This is illustrated in the Figure below.

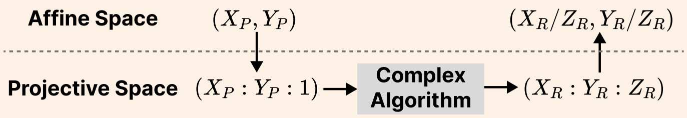


Illustration: General strategy when performing operations over Elliptic Curves.

5.2.5 Scalar Multiplication Basic Implementation

Now, the question is: how do we implement the scalar multiplication $[k]P$ for the given scalar $k \in \mathbb{Z}_r$ and point $P \in E(\overline{\mathbb{F}}_q)$?

First idea: let us simply add P to itself k times. Well, the complexity would be $O(k)$ in this case, which is even harder than solving the discrete logarithm problem (recall that the discrete logarithm problem has a complexity of $O(\sqrt{k})$). Yikes.

So there should be a better way. In this section we will limit ourselves to the double-and-add method, but the curious reader can look up the NAF (Non-Adjacent Form) method, windowed methods, GLV scalar decomposition and many other methods, which we are not going to cover in this course.

The idea of the double-and-add method is following: we represent the N-bit scalar k in binary form, say $k = (k_{N-1}, k_{N-2}, \dots, k_0)_2$, then we calculate

$P, [2]P, [4]P, [8]P, \dots, [2^{N-1}]P$ (which is simply applying the doubling multple times) and then add the corresponding points (corresponding to positions where $k_i = 1$) to get the result. Formally, we specify the Algorithm 3.

Algorithm 3: Double-and-add method for scalar multiplication

```
   Input  : $P \in E(\mathbb{F}_q)$ and $k \in \mathbb{Z}_r$
   Output : Result of scalar multiplication $[k]P \in E(\mathbb{F}_q)$
1  Decompose $k$ to the binary form: $(k_0, k_1, \dots, k_{N-1})$
2  $R \leftarrow \mathcal{O}$
3  $T \leftarrow P$
4  for $i \in \{0, \dots, N-1\}$ do
5      if $k_i = 1$ then
6          $R \leftarrow R \oplus T$
7      end
8      $T \leftarrow [2]T$
9  end
   Return : Point $R$
```

Good news: now we have a complexity of $O(N) = O(\log k)$, which is way much better that a linear one. In fact, many more optimized methods have the same assymptotic complexity (meaning, a logarithmic one), so it turns out that we cannot do much better than that. However, the main advantages of other, more optimized methods is that we can avoid making too many additions (here, in the worst case, we have to make N additions), which is a costly operation (and more expensive than doubling).

Moreover, here we can use projective coordinates to make the addition and doubling operation more efficient! After all, typically the number of operations is even more than 300, so making 300 inversions in affine form is not an option.

5.3 Elliptic Curve Pairing

Pairing is the core object used in threshold signatures, zk-SNARKs constructions, and other cryptographic applications.

Consider the *Decisional Diffie-Hellman problem* which we described in Section 2 (based on g^α, g^β and g^γ, decide whether $\gamma = \alpha\beta$). Turns out that for curves where the so-called *embedding degree*[10] is small enough, this problem is easy to solve. This might sound like a quite bad thing, but it turns out that although some information about the discrete logarithm is leaked, it is not enough to break the security of the system (basically, solve the *Computational*

[10]We will mention what that is is later, but still this term is quite hard to define.

Diffie-Hellman problem). Pairings is the exact object that allows us to solve the Decisional Diffie-Hellman problem.

However, a more interesting use-case which we are going to use in SNARKs is that pairings allows us to check **quadratic conditions** on scalars using their corresponding elliptic curve representation. For example, just given $u = g^{\alpha}, v = g^{\beta}$ we can check whether $\alpha\beta + 5 = 0$ (which is impossible to check without having a pairing).

So what is pairing?

5.3.1 Definition

Definition 5.8. Pairing is a bilinear, non-degenerate, efficiently computable map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1, \mathbb{G}_2$ are two groups (typically, elliptic curve groups) and $\mathbb{G}_T$ is a target group (typically, a set of scalars). Let us decipher the definition:

- **Bilinearity** means essentially the following:

$$e([a]P, [b]Q) = e([ab]P, Q) = e(P, [ab]Q) = e(P, Q)^{ab}.$$

- **Non-degeneracy** means that $e(G_1, G_2) \neq 1$ (where G_1, G_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$, respectively). This property basically says that the pairing is not trivial.
- **Efficient computability** means that the pairing can be computed in a reasonable time.

The definition is illustrated in Figure 5.1.

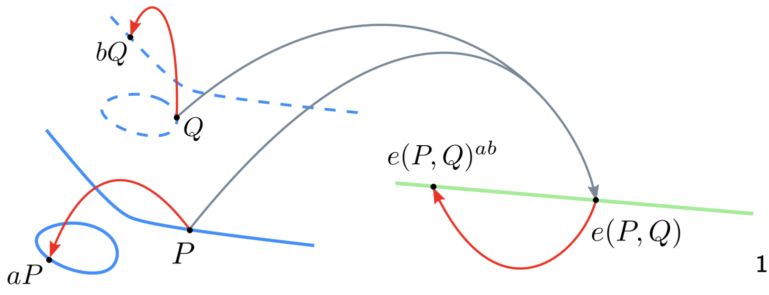


Figure 5.1: Pairing illustration. It does not matter what we do first: (a) compute $[a]P$ and $[b]Q$ and then compute $e([a]P, [b]Q)$ or (b) first calculate $e(P, Q)$ and then transform it to $e(P, Q)^{ab}$. Figure taken from "Pairings in R1CS" talk by Youssef El Housni

Example. Suppose $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}_T = \mathbb{Z}_r$ are scalars. Then, the map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, defined as:

$$e(x, y) = 2^{xy}$$

is pairing. Indeed, it is bilinear. For example, $e(ax, by) = 2^{abxy} = (2^{xy})^{ab} = e(x, y)^{ab}$ or $e(ax, by) = 2^{abxy} = 2^{(x)(aby)} = e(x, aby)$. Moreover, it is non-degenerate, since $e(1, 1) = 2 \neq 1$. And finally, it is obviously efficiently computable.
However, this is a quite trivial example since working over integers is typically not secure. For example, the discrete logarithm over $\mathbb{Z}_r$ can be solved in subexponential time. For that reason, we want to build pairings over elliptic curves.

Example. Pairing for BN254. For BN254 (with equation $y^2 = x^3 + 3$), the pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is defined over the following groups:

- $\mathbb{G}_1$ — points on the regular curve $E(\mathbb{F}_p)$.
- $\mathbb{G}_2$ — r-torsion points on the twisted curve $E'(\mathbb{F}_{p^2})$ over the field extension $\mathbb{F}_{p^2}$ (with equation $y^2 = x^3 + \frac{3}{\xi}$ for $\xi = 9 + u \in \mathbb{F}_{p^2}$).
- $\mathbb{G}_T$ — rth roots of unity $\Omega_r \subset \mathbb{F}_{p^{12}}^{\times}$.

Well, this one is quite intense and even understanding the input and output parameters is quite hard. So let us decipher some components:

- **r-torsion subgroup on the curve** $E(\mathbb{F}_{p^m})$ is simply a set of points, which multiplied by r give the point at infinity (that is, $[r]P = \mathcal{O}$). Formally, $E(\mathbb{F}_{p^m})[r] = \{P \in E(\mathbb{F}_{p^m}) : [r]P = \mathcal{O}\}$. Of course, for the curve $E(\mathbb{F}_p)$, the r-torsion subgroup is simply the whole curve, but that is generally not the case for the twisted curve over the extension field.
- **rth roots of unity** is a set of elements $\Omega_r = \{z \in \mathbb{F}_{p^{12}}^{\times} : z^r = 1\}$. This is a group under multiplication, and it has exactly r elements.

Remark. One might a reasonable question: where does this 12 come from? The answer is following: the so-called **embedding degree** of BN254 curve is $k = 12$. This number is the key to understanding why we are working over such large extensions when calculating the pairing. The formal description is quite hard, but the intuition is following: the embedding degree is the smallest number k such that all the rth roots of unity lie inside the extended field $\mathbb{F}_{p^k}$. If k was smaller, the output of pairing would contain less that r points and some points would be missing, which would make the pairing more trivial. For that reason, we need to have $\Omega_r \subset \mathbb{F}_{p^k}$.

Definition 5.9. The following conditions are equivalent **definitions** of an embedding degree k of an elliptic curve $E(\overline{\mathbb{F}}_p)$:

- k is the smallest positive integer such that $r \mid (p^k - 1)$.
- k is the smallest positive integer such that $\mathbb{F}_{p^k}$ contains all of the r-th roots of unity in $\overline{\mathbb{F}}_p$, that is $\Omega_r \subset \mathbb{F}_{p^k}$.
- k is the smallest positive integer such that $E(\overline{\mathbb{F}}_p)[r] \subset E(\mathbb{F}_{p^k})$

Pretty obvious observation: lower embedding degree is faster to work with, since it allows us to work over smaller fields. But usually, this embedding degree is quite large and we need to craft elliptic curves specifically to have a small embedding degree. For example, a pretty famous curve `secp256k1` has an embedding degree

$$k = \text{0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa74727a26728c1ab49ff8651778090ae0},$$

which is 254-bit long. For that reason, it is natural to define the term *pairing-friendly elliptic curve*.

Definition 5.10. An elliptic curve is called **pairing-friendly** if it has a relatively small embedding degree k (typically, $k \leq 16$).

Remark. One might ask why usually, when dealing with pairings, we do not get to work with field extensions that much (most likely, if you were to write `groth16` from scratch using some mathematical libraries, you will not need to work with $\mathbb{F}_{p^{12}}$ arithmetic specifically). The reason is that typically, libraries implement the following abstraction: given a set of points $\{(P_i, Q_i)\}_{i=1}^n \subset \mathbb{G}_1^n \times \mathbb{G}_2^n$, the function checks whether

$$\prod_{i=1}^{n} e(P_i, Q_i) = 1$$

Note that in this case, we do not need to work with $\mathbb{F}_{p^{12}}$ arithmetic, but rather checking the equality in the target group $\mathbb{G}_T$.
Interesting fact: this condition is specified in the `ecpairing` precompile standard used in Ethereum.

5.3.2 Case Study: BLS Signature

One of the most elegant applications of pairings is the BLS Signature scheme. Compared to ECDSA or other signature schemes, BLS can be formulated in three lines.

Suppose we have pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ (with generators G_1, G_2, respectively), and a hash function H, mapping message space $\mathcal{M}$ to $\mathbb{G}_1$.

Definition 5.11. BLS Signature Scheme consists of the following algorithms:

- $\mathsf{Gen}(1^\lambda)$: Key generation. $\mathsf{sk} \xleftarrow{R} \mathbb{Z}_q, \mathsf{pk} \leftarrow [\mathsf{sk}]G_2 \in \mathbb{G}_2$.
- $\mathsf{Sign}(\mathsf{sk}, m)$. Signature is $\sigma \leftarrow [\mathsf{sk}]\mathsf{H}(m) \in \mathbb{G}_1$.
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$. Check whether $e(\mathsf{H}(m), \mathsf{pk}) = e(\sigma, G_2)$.

Let us check the correctness:

$$e(\sigma, G_2) = e([\mathsf{sk}]\mathsf{H}(m), G_2) = e(\mathsf{H}(m), [\mathsf{sk}]G_2) = e(\mathsf{H}(m), \mathsf{pk})$$

As we see, the verification equation holds.

Remark. $\mathbb{G}_1$ and $\mathbb{G}_2$ might be switched: public keys might live instead in $\mathbb{G}_1$ while signatures in $\mathbb{G}_2$.

This scheme is also quite famous for its aggregation properties, which we are not going to consider today.

5.3.3 Case Study: Verifying Quadratic Equations

Example. Suppose Alice wants to convince Bob that he knows such α, β such that $\alpha + \beta = 2$, but she does not want to reveal α, β. She can do the following trick:

1. Alice computes $P \leftarrow [\alpha]G, Q \leftarrow [\beta]G$ — points on the curve.
2. Alice sends (P, Q) to Bob.
3. Bob verifies whether $P \oplus Q = [2]G$.

It is easy to verify the correctness of the scheme: suppose Alice is honest and she sends the correct values of α, β, satisfying $\alpha + \beta = 2$. Then, $P \oplus Q = [\alpha]G \oplus [\beta]G = [\alpha + \beta]G = [2]G$. Moreover, Bob cannot learn α, β since the computational discrete logarithm problem is hard.

Example. Well, now suppose I make the problem just a bit more complicated: Alice wants to convince that she knows α, β such that $\alpha\beta = 2$. And it turns out that elliptic curve points on their own are not enough to verify this. However, using pairings, we can do the following trick: assume we have a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1$ is generated by G_1 and $\mathbb{G}_2$ is generated by G_2. Then, Alice can do the following:

1. Alice computes $P \leftarrow [\alpha]G_1 \in \mathbb{G}_1, Q \leftarrow [\beta]G_2 \in \mathbb{G}_2$ — points on two curves.
2. Alice sends $(P, Q) \in \mathbb{G}_1 \times \mathbb{G}_2$ to Bob.
3. Bob checks whether: $e(P, Q) = e(G_1, G_2)^2$.

Remark. The last verification can be also rewritten as $e(P, Q)e(G_1, G_2)^{-2} = 1$, which is more frequently used in practice.

Example. Finally, let us prove something more interesting. Like, based on (x_1, x_2), whether

$$x_1^2 + x_1x_2 = x_2$$

Alice can calculate $P_1 \leftarrow [x_1]G_1 \in \mathbb{G}_1, P_2 \leftarrow [x_1]G_2 \in \mathbb{G}_2, Q \leftarrow [x_2]G_2 \in \mathbb{G}_2$. Then, the condition can be verified by checking whether

$$e(P_1, P_2 \oplus Q)e(G_1, \ominus Q) = 1$$

Let us see the correctness of this equation:

$$\begin{aligned} e(P_1, P_2 \oplus Q)e(G_1, \ominus Q) &= e([x_1]G_1, [x_1 + x_2]G_2)e(G_1, [x_2]G_2)^{-1} \\ &= e(G_1, G_2)^{x_1(x_1+x_2)}e(G_1, G_2)^{-x_2} = e(G_1, G_2)^{x_1^2+x_1x_2-x_2} \end{aligned} \tag{1}$$

Now, if this is 1, then $x_1^2 + x_1x_2 = x_2$, which was exactly what we wanted to prove.

5.4 Exercises

Exercise 1. What is **not** a valid equivalence relation $\sim$ over a set $\mathcal{X}$?

(A) $a \sim b$ iff $a + b < 0$, $\mathcal{X} = \mathbb{Q}$.
(B) $a \sim b$ iff $a = b$, $\mathcal{X} = \mathbb{R}$.
(C) $a \sim b$ iff $a \equiv b \pmod 5$, $\mathcal{X} = \mathbb{Z}$.
(D) $a \sim b$ iff the length of $a =$ the length of b, $\mathcal{X} = \mathbb{R}^2$.
(E) $(a_1, a_2, a_3) \sim (b_1, b_2, b_3)$ iff $a_3 = b_3$, $\mathcal{X} = \mathbb{R}^3$.

Exercise 2. Suppose that over $\mathbb{R}$ we define the following equivalence relation: $a \sim b$ iff $a - b \in \mathbb{Z}$ $(a, b \in \mathbb{R})$. What is the equivalence class of 1.4 (that is, $[1.4]_\sim$)?

(A) A set of all real numbers.
(B) A set of all integers.
(C) A set of reals $x \in \mathbb{R}$ with the fractional part of x equal to 0.4.
(D) A set of reals $x \in \mathbb{R}$ with the integer part of x equal to 1.
(E) A set of reals $x \in \mathbb{R}$ with the fractional part of x equal to 0.6.

Exercise 3. Which of the following pairs of points in homogeneous projective space $\mathbb{P}^2(\mathbb{R})$ are **not** equivalent?

(A) $(1 : 2 : 3)$ and $(2 : 4 : 6)$.
(B) $(2 : 3 : 1)$ and $(6 : 9 : 3)$.
(C) $(5 : 5 : 5)$ and $(2 : 2 : 2)$.
(D) $(4 : 3 : 2)$ and $(16 : 8 : 4)$.

Exercise 4. The main reason for using projective coordinates in elliptic curve cryptography is:

(A) To reduce the number of point additions in algorithms involving elliptic curves.
(B) To make the curve more secure against attacks.
(C) To make the curve more efficient in terms of memory usage.
(D) To reduce the number of field multiplications when performing scalar multiplication.
(E) To avoid making too many field inversions in complicated algorithms involving elliptic curves.

Exercise 5. Suppose $k = 19$ is a scalar and we are calculating $[k]P$ using the double-and-add algorithm. How many elliptic curve point addition operations will be performed?

(A) 0. (B) 1. (C) 2. (D) 3. (E) 4.

Exercise 6. What is the minimal number of inversions needed to calculate the value of expression (over $\mathbb{F}_p$)

$$\frac{a-b}{(a+b)^4} + \frac{c}{a+b} + \frac{d}{a^2+c^2},$$

for the given scalars $a, b, c, d \in \mathbb{F}_p$?

(A) 1. (B) 2. (C) 3. (D) 4. (E) 5.

Exercise 7. Given pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with G_1 — generator of $\mathbb{G}_1$ and $G_2 \in \mathbb{G}_2$ — generator of $\mathbb{G}_2$, which of the following is **not** equal to $e([3]G_1, [5]G_2)$?

(A) $e([5]G_1, [3]G_2)$.
(B) $e([4]G_1, [4]G_2)$.
(C) $e([15]G_1, G_2)$.
(D) $e([3]G_1, G_2)e(G_1, [12]G_2)$.
(E) $e(G_1, G_2)^{15}$.

Exercise 8*. *Unit Circle Proof.* Suppose Alice wants to convince Bob that she knows a point on the unit circle $x^2 + y^2 = 1$. Suppose we are given a symmetric pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ for $\mathbb{G}_1 = \mathbb{G}_2 = \langle G \rangle$ and Alice computes $P \leftarrow [x]G, Q \leftarrow [y]G$. She then proceeds to sending (P, Q) to Bob. Which of the following checks should Bob perform to verify that Alice indeed knows a point on the unit circle?

(A) Check if $e(P, Q)e(Q, P) = 1$.
(B) Check if $e([2]P, [2]Q) = e(G, G)$.
(C) Check if $e([2]P, Q)e(Q, [2]P) = 1$.
(D) Check if $e(P, P) + e(Q, Q) = 1$.
(E) Check if $e(P, P)e(Q, Q) = e(G, G)$.

6 Commitment Schemes

6.1 Commitments

Definition 6.1. A cryptographic commitment scheme allows one party to commit to a chosen statement (such as a value, vector, or polynomial) without revealing the statement itself. The commitment can be revealed in full or in part at a later time, ensuring the integrity and secrecy of the original statement until the moment of disclosure.

Before delving into the details, here is the intuition of cryptographic commitments.

Imagine putting a letter with some message into a box and locking it with your key. You then give that box to your friend, who cannot open it without the key. In this scenario, you have made a commitment to the message inside the box. You cannot change the content of the letter, as it is in your friend's possession. At the same time, your friend cannot access the letter since they do not have the key to unlock the box.

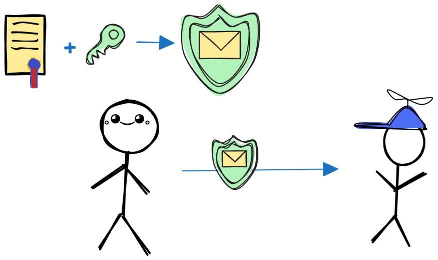


Figure 6.1: Commitment scheme

Definition 6.2 (Commitment Scheme)**.** Commitment Scheme $\Pi_{\text{commitment}}$ is a tuple of three algorithms: $\Pi_{\text{commitment}} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$.

1. $\mathsf{Setup}(1^\lambda)$: returns public parameter pp for both comitter and verifier;
2. $\mathsf{Commit}(\mathsf{pp}, m)$: returns a commitment c to the message m using public parameters pp and, optionally, a secret opening hint r;
3. $\mathsf{Open}(\mathsf{pp}, c, m, r)$: verifies the opening of the commitment c to the message m with an opening hint r.

Definition 6.3 (Commitment Scheme)**.** Properties of commitment schemes:

1. *Hiding:* verifier should not learn any information about the message given only the commitment c. To put it formally, we define a game:
   (a) Adversary chooses two messages m_1, m_2 and sends to the challenger.
   (b) Challenger chooses a random bit b, commits to both messages: $c_1 \leftarrow \mathsf{Commit}(pp, m_1)$, $c_2 \leftarrow \mathsf{Commit}(pp, m_2)$, and sends c_b to the adversary.
   (c) Adversary guesses a bit $\hat{b}$.

   We define the hiding advantage of a PPT adversary $\mathcal{A}$ as

$$\mathsf{HideAdv}[\mathcal{A}, \Pi_{\text{commitment}}] := \left| \Pr[b = \hat{b}] - \frac{1}{2} \right|$$

   We say that the commitment scheme $\Pi_{\text{commitment}}$ is *hiding* if for any adversary, the aforementioned advantage is negligible.

2. *Binding:* prover could not find another message m_1 and open the commitment c without revealing the commited message m. To put it formally, we define a game:
   (a) Adversary chooses five values: commitment c and two distinct pairs (m_0, r_0) and (m_1, r_1).
   (b) Adversary computes $b_j \leftarrow \mathsf{Open}(\mathsf{pp}, c, m_j, r_j)$.

   Define the advantage in the binding game as:

$$\mathsf{BindAdv}[\mathcal{A}, \Pi_{\text{commitment}}] = \Pr[b_0 = b_1 \neq 0 \wedge m_0 \neq m_1]$$

   We say that the commitment scheme is binding if for any adversary, such advantage is negligible.

6.1.1 Hash-based commitments

As the name implies, we are using a cryptographic hash function H in such scheme.

1. Prover selects a message m from a message space M which he wants to commit to: $m \leftarrow \mathbb{M}$
2. Prover samples random value r (usually called blinding factor) from a challange space $\mathcal{C} \subset \mathbb{Z}$: $r \xleftarrow{R} \mathcal{C}$
3. Both values will be concatenated and hashed with the hash function H to produce the commitment: $c = H(m \parallel r)$

Commitment should be shared with a verifier. During the opening stage, prover reveals (m, r) to the *Verifier*. To check the commitment, verifier computes: $c_1 = H(m \parallel r)$.

If $c_1 = c$, prover has revealed the correct pair (m, r).

It should be noted that a cryptographic hash function aims to provide collision resistance, meaning that the probability two different messages will result in one output is negligible. Because the *Verifier* knows the hash function digest c before the *Prover* reveals m and r, the *Prover* would need to find a collision $H(m' \parallel r') = H(m \parallel r)$ to be able to convince the *Verifier* that m' value was committed.

However, due to the collision resistance, finding such m' and r' is computationally infeasible. Which means the *Prover* won't be able to convince the *Verifier* that the commitment was done to another value providing a *binding* property.

A cryptographically secure hash function is a one-way function, which means that finding the hash preimage is almost as hard as bruteforcing all possibile input values. Given large challenge space, the probability of the *Verifier* of finding (m, r) such that $H(m, r) = c$ is negligible, which ensures *hiding* property of the commitment scheme.

6.1.2 Pedersen commitments

Pedersen commitments allow us to represent arbitrarily large vectors with a single elliptic curve point, while optionally hiding any information about the vector. Pedersen commitment uses a public group $\mathbb{G}$ of order q and two random public generators G and U: $U = [u]G$. Secret parameter u should be unknown to anyone, otherwise the *Binding* property of the commitment scheme will be violated. EC point U is chosen randomly using "Nothing-up-my-sleeve" to assure no one knows the discrete logarithm of a selected point.

Remark. Transparent random points generation
User can pick the publicly chosen random number (like a hash of project name, first numbers of π, etc), and hash that result to obtain another value. If that results in an x value that lies on the elliptic curve, use that as the random point and hash the (x, y) pair again (to obtain the next one, it needed). Otherwise, if the x-value does not land on the curve, increment x until it does. Because the committer is not generating the points, they don't know their discrete log.

Pedersen commitment scheme algorithm:

1. Prover and Verifier agrees on G and U points in a elliptic curve point group $\mathbb{G}$, q is the order of the group.

2. Prover selects a value m to commit and a blinder factor r: $m \leftarrow \mathbb{Z}_q$, $r \overset{R}{\leftarrow} \mathbb{Z}_q$
3. Prover generates a commitment and sends it to the Verifier: $c \leftarrow [m]G + [r]U$

During the opening stage, prover reveals (m, r) to the verifier. To check the commitment, verifier computes: $c_1 = [m]G + [r]U$.

If $c_1 = c$, prover has revealed the correct pair (m, r).

Remark. In case the discrete logarithm of U is leaked, the *binding* property can be violated by the *Prover*:

$$c = [m]G + [r]U = [m]G + [r \cdot u]G = [m + r \cdot u]G$$

For example, $(m + u, r - 1)$ will have the same commitment value:

$$[m + u + (r - 1) \cdot u]G = [m + u - u + r \cdot u]G = [m + r \cdot u]G$$

Commitment aggregation

Pedersen commitment have some advantages compared to hash-based commitments. Additively homomorphic property allows to accumulate multiple commitments into one. Consider two pairs: $(m_1, r_1), (m_2, r_2)$.

$$\begin{aligned} c_2 &= [m_1]G + [r_1]U, \\ c_2 &= [m_2]G + [r_2]U, \\ c_a = c_1 + c_2 &= [m_1 + m_2]G + [r_1 + r_2]U \end{aligned}$$

This works for any number of commitments, so we can encode as many points as we like in a single one. For example, if a set of balances is committed, the sum of any subset can be proven without revealing the exact value of each balance. This is achieved by disclosing the sum of the balances and the corresponding sum of the blinding factors.

6.1.3 Vector commitments

Vector commitment schemes allows to commit to a vector of values rather than a value and a blinding term.

Pedersen Vector Commitments

Suppose we have a set of random elliptic curve points $(G_1, \ldots, G_n)$ of cyclic group $\mathbb{G}$ (that we do not know the discrete logarithm of), a vector $(m_1, m_2 \ldots m_n)$ and a random value r. We can do the following:

$$c = [m_1]G_1 + [m_2]G_2 \ldots + [m_n]G_n + [r]Q$$

Since the *Prover* does not know the discrete logarithm of the generators, they don‘t know the discrete logarithm of [C]. Hence, this scheme is binding: they can only reveal $(v_1, \ldots, v_n)$ to produce $[C]$ later, they cannot produce another vector.

Prover can later open the commitment by revealing the vector $(m_1, m_2 \ldots m_n)$ and a blinding term r.

Merkle Tree based Vector Commitments

A naive approach for a vector commitment would be hash the whole vector. More sophisticated scheme uses divide-and-conquer approach by building a binary tree out of vector elements.

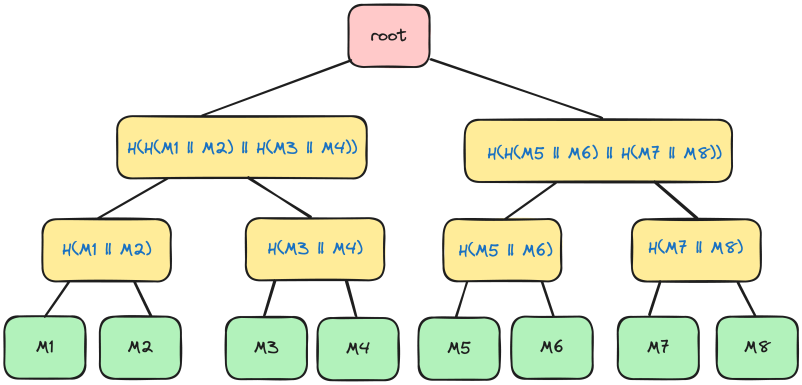


Figure 6.2: Merkle Tree structure

A Merkle Tree is a data structure to efficiently and securely verify the commiments to a vector of data. It is a binary tree where each leaf node represents a hash of a data block, and each non-leaf node is a hash of its child nodes' concatenated hashes. The top node, called the root hash or Merkle root, uniquely represents the entire data set. By comparing this root with a known valid root, one can quickly verify the authenticity and integrity of the data without needing to examine the entire dataset.

To prove the inclusion of element into the tree, a corresponding Merkle Branch is used. On the example below, M_1 inclusion is proved, and $(M_2, H(M_3 \parallel M_4), H(H(M_5 \parallel M_6) \parallel H(M_7 \parallel M_8)))$ is an inclusion branch vector.

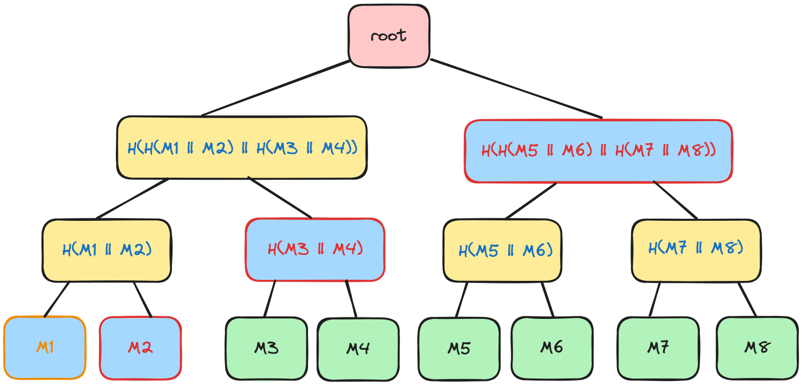


Figure 6.3: Merkle Tree inclusion proof branch

One of Merkle tree key advantages is that it allows for the selective disclosure of specific elements within the data set without revealing the rest.

6.1.4 Polynomial commitment

Polynomial commitment can be used to prove that the commited polynomial satisfies certain properties $P(x_1, x_2, \dots, x_n) = y$, without revealing what the polynomial is. The commitment is generally succint, which means that it is much smaller than the polynomial it represents.

The KZG polynomial commitment scheme

The KZG (Kate-Zaverucha-Goldberg) is a polynomial commitment scheme:

1. *One-time "Powers-of-tau" trusted setup stage*. During trusted setup a set of elliptic curve points is generated. Let G be a generator point of some pairing-friendly elliptic curve group $\mathbb{G}$, s some random value in the order of the G point and d be the maximum degree of the polynomials we want to commit to. Public parameters of a trusted setup are calculated as:

$$[\tau^0]G, [\tau^1]G, \dots, [\tau^d]G$$

   Parameter τ should be deleted after the ceremony. If it is revealed, the *binding* property of the commitment scheme can be broken. This parameter is usually called the *toxic waste*.

2. *Commit to polynomial*. Given the polynomial $p(x) = \sum_{i=0}^{d} p_i x^i$, compute the commitment $c = [p(\tau)]G$ using the trusted setup. Although the committer cannot compute $p(\tau)$ directly since the value of τ is unknown, he can compute it using values $([\tau^0]G, [\tau^1]G, \dots, [\tau^d]G)$:

$$[p(\tau)]G = [\sum_{i=0}^{d} p_i \tau^i]G = \sum_{i=0}^{d} p_i [\tau^i]G$$

3. *Prove an evaluation*. To prove that at some point x_0 polynomial equals y_0 ($p(x_0) = y_0$), compute polynomial

   $$q(x) = \tfrac{p(x) - y_0}{x - x_0}.$$

   Polynomial $q(x)$ is called "quotient polynomial" and only exists if and only if $p(x_0) = y_0$:

   (a) If $p(x_0) = y_0$, we define $r(x) := p(x) - y_0$;
   (b) $r(x)$ has x_0 as a root, as $r(x_0) = 0$ by the definition. That is why there exists $q(x)$, such that $r(x) = q(x) \cdot (x - x_0)$;
   (c) Hence, the expression $q(x) = \frac{p(x) - y_0}{x - x_0}$ is a polynomial.

   The existance of this quotient polynomial serves as a proof of the evaluation. *Prover* calculates proof $\pi = [q(\tau)]G$ and sends it to the *Verifier*.
4. *Verify the proof*. Given a commitment $c = [p(\tau)]G$, an evaluation $p(x_0) = y_0$ and a proof $[q(\tau)]G$, we need to ensure that $q(\tau) \cdot (\tau - x_0) = p(\tau) - y_0$. This can be done using trusted setup without knowledge of τ using bilinear mapping:

   $$e([q(\tau)]G_1, [\tau]G_2 - [x']G_2) = e([p(\tau)]G_1 - [y]G_1, G_2)$$

   Polynomial commiment schemes such as KZG are used in zero knowledge proof system to encode circuit constraints as a polynomial, so that verifier could check random points to ensure that the constraints are met.

6.2 Exercises

Exercise 1. Dmytro and Denis were watching a horse race. Confident in his ability to predict the outcome, Dmytro decided to commit to his prediction. However, in his haste, he forgot to use a blinding factor. Now, Dmytro is concerned that Denis might discover his prediction before the race ends, which would defeat the purpose of his commitment.

We define a dummy hash function $H(a) = (a \cdot 13 + 17) \pmod{41}$. Dmytro used a *hash-based commitment* and H as a hash function. Set of race horse numbers is $(3, 5, 8, 15)$. Help Denis to find out the horse number Dmytro have made a commitment to, if commitment equals $C = 39$.

(A) 3. (B) 5. (C) 8. (D) 15.

Exercise 2. Denis made a setup (points G and U) for a Pedersen commitment scheme and commited values $(m, r) = (3, 7)$ to Dmytro by sending him $\mathcal{C} = [3]G + [7]U$. Dmytro did not verify the setup. Turns out that Denis knows that $U = [6]G$. Denis is planning to send a different message from the one he originally committed to to $m_2 = 15$. Which values (m_2, r_2) should he send to Dmytro at the opening stage?

(A) $(15, 5)$ (B) $(15, 7)$ (C) $(15, 4)$ (D) $(3, 5)$

Exercise 3. We define a dummy hash function $H(a, b) = (a \cdot 3 + b \cdot 7) \pmod{41}$. You have a Merkle tree built with depth 4 using hash function H with root equal 37. Which inclusion proof is valid for element 3? Position defines how leaves should be hashed:

- if $left \to h_i = Hash(h_{i-1}, branch[i])$
- if $right \to h_i = Hash(branch[i], h_{i-1})$

(A) branch: $[4, 16, 13]$, position: $[left, right, left]$

(B) branch: $[1, 40, 3]$, position: $[left, left, left]$

(C) branch: $[5, 12, 13]$, position: $[right, right, left]$

(D) branch: $[4, 17, 13]$, position: $[left, right, left]$

Exercise 4. Given a polynomial $p(x) = x^3 - 10x^2 + 31x - 30$, Oleksandr wants to prove that $p(2) = 0$. To do that, according to the KZG commitment scheme, he constructs the quotient polynomial $q(x)$ and wants to show that $q(\tau) \cdot (\tau - 2) = p(\tau)$. Assuming Oleksandr has conducted these steps correctly, what value of $q(x)$ has Oleksandr calculated?

(A) $q(x) = 2x^2 + 4x - 6$ (C) $q(x) = x^2 - 8x + 15$

(B) $q(x) = x^3 - 10x^2 + 30x - 28$ (D) $q(x) = x^2 + 5x + 18$

7 Introduction to Zero-Knowledge Proofs

7.1 Motivation

Finally, we came to the most interesting part of the course: zero-knowledge proofs. Before we start with SNARKs, STARKs, Bulletproofs, and other zero-knowledge proof systems, let us first define what the zero-knowledge is. But even before that, we need to introduce some formalities. For example, what are "proof", "witness", and "statement" — terms that are so widely used in zero-knowledge proofs.

Let us describe the typical setup. We have two parties: **prover** $\mathcal{P}$ and a **verifier** $\mathcal{V}$. The prover wants to convince the verifier that some statement is true. Typically, the statement is not obvious (well, that is the reason for building proofs after all!) and therefore there might be some "helper" data, called **witness**, that helps the prover to prove the statement. The reasonable question is whether the prover can simply send witness to verifier and call it a day. Of course since you are here, reading this lecture, it is obvious that the answer is no. More specifically, by introducing zk-SNARKs, STARKs, and other proving systems, we will try to mitigate the following issues:

- **Zero-knowledge:** The prover wants to convince the verifier that the statement is true without revealing the witness.
- **Argument of knowledge:** Moreover, typically we want to make sure that the verifier, besides the statement correctness, ensures that the prover **knows** such a witness related to the statement.
- **Succinctness:** The proof should be short, ideally logarithmic in the size of the statement. This is crucial for practical applications, especially in the blockchain space where we cannot allow to publish long proofs on-chain. Moreover, verification should be efficient as well.

Example. Suppose, given a hash function[a] $H : \{0,1\}^* \to \{0,1\}^\ell$, the prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ that he knows the preimage $x \in \{0,1\}^*$ such that $H(x) = y$ for some given public value $y \in \{0,1\}^\ell$. The properties listed above are interpreted as follows:

- **Zero-knowledge:** The prover $\mathcal{P}$ does not want to reveal *anything* about the pre-image x to the verifier $\mathcal{V}$.
- **Argument of knowledge:** Given a string $y \in \{0,1\}^\ell$ it is not sufficient for a prover to merely state that y has a pre-image. The prover $\mathcal{P}$ must demonstrate to a verifier $\mathcal{V}$ that he **knows** such a pre-image $x \in \{0,1\}^*$.

- **Succinctness:** If the hash function takes n operations to compute[b], the proof should be **much** shorter than n operations. State-of-art solutions can provide proofs that are $O((\log n)^c)$ (polylogarithmic) is size! Moreover, verification time of such proof is also typically polylogarithmic (or even $O(1)$ in some cases).

[a]The notation $\{0,1\}^*$ means binary strings of arbitrary length

[b]Note that "number of operations" is very vague term. One way to measure the "size" of some computational problem is specifying the number of gates in the arithmetical circuit $\mathtt{C}(x,w)$, representing the computation of this problem (denoted as $|\mathtt{C}|$, respectively).

7.2 Relations and Languages

Recall that **relation** $\mathcal{R}$ is just a subset of $\mathcal{X} \times \mathcal{Y}$ for two arbitrary sets $\mathcal{X}$ and $\mathcal{Y}$. Now, we are going to introduce the notion of a *language of true statements* based on $\mathcal{R}$.

Definition 7.1 (Language of true statements)**.** Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ be a relation. We say that a statement $x \in \mathcal{X}$ is a **true** statement if $(x,y) \in \mathcal{R}$ for some $y \in \mathcal{Y}$, otherwise the statement is called **false**. We define by $\mathcal{L}_{\mathcal{R}}$ (the language over relation $\mathcal{R}$) the set of all true statements, that is:

$$\mathcal{L}_{\mathcal{R}} = \{x \in \mathcal{X} : \exists y \in \mathcal{Y} \text{ such that } (x,y) \in \mathcal{R}\}.$$

Now, what is the purpose of introducing relations and languages? The idea is that relation is a natural way to formalize the notion of a statement and witness. Namely, we denote the elements of $\mathcal{X}$ as statements and the elements of $\mathcal{Y}$ as witnesses.

Further, we denote by w the witness for the statement $x \in \mathcal{L}_{\mathcal{R}}$. Oftentimes, one might also encounter notation ϕ to denote the statement, but we will stick to x for simplicity.

Example. Suppose we want to prove the following claim: number $n \in \mathbb{N}$ is the product of two large prime numbers $(p,q) \in \mathbb{N} \times \mathbb{N}$. Here, the relation is the following:

$$\mathcal{R} = \{(n,p,q) \in \mathbb{N}^3 : n = p \cdot q \text{ where } p, q \text{ are primes}\}$$

In this particular case, the language of true statements is defined as

$$\mathcal{L}_{\mathcal{R}} = \{n \in \mathbb{N} : \exists p, q \text{ are primes such that } n = pq\}$$

Therefore, our initial claim we want to prove is $n \in \mathcal{L}_{\mathcal{R}}$. The witness for this statement is the pair (p,q), where p and q are prime numbers such that $n = p \cdot q$ and typically (but not always) we want to prove this without revealing our witness: p and q. For example, one valid witness for $n = 15$ is $(3, 5)$, while $n = 16$ does not have any valid witness, so $16 \notin \mathcal{L}_{\mathcal{R}}$.

Example. Another example of claim we want to prove is the following: number $x \in \mathbb{Z}_N^{\times}$[a] is a **quadratic residue** modulo N, meaning there exists some $w \in \mathbb{Z}_N^{\times}$ such that $x \equiv w^2 \pmod{N}$ (naturally, w is called a *square root* of x). The relation in this case is:

$$\mathcal{R} = \{(x, w) \in (\mathbb{Z}_N^{\times})^2 : x \equiv w^2 \pmod{N}\}$$

In this case, $\mathcal{L}_{\mathcal{R}} = \{x \in \mathbb{Z}_N^{\times} : \exists w \in \mathbb{Z}_N^{\times} \text{ such that } x \equiv w^2 \pmod{N}\}$. Here, our square root of x modulo N, that is w, is the witness for the statement $x \in \mathcal{L}_{\mathcal{R}}$. For example, for $N = 7$ we have $4 \in \mathcal{L}_{\mathcal{R}}$ since 5 is a valid witness: $5^2 \equiv 4 \pmod{7}$, while $3 \notin \mathcal{L}_{\mathcal{R}}$ since there is no valid witness for 3.

[a]By $\mathbb{Z}_N^{\times}$ we denote the multiplicative group of integers modulo N. In other words, this is a set of integers $\{x \in \mathbb{Z}_N : \text{gcd}\{x, N\} = 1\}$.

However, we want to limit ourselves to languages that has reasonably efficient verifier (since otherwise the problem is not really practical and therefore of little interest to us). For that reason, we define the notion of a *NP Language* and from now on, we will be working with such languages.

Definition 7.2 (NP Language)**.** A language $\mathcal{L}_{\mathcal{R}}$ belongs to the NP class if there exists a polynomial-time verifier $\mathcal{V}$ such that the following two properties hold:

- **Completeness:** If $x \in \mathcal{L}_{\mathcal{R}}$, then there is a witness w such that $\mathcal{V}(x, w) = 1$ with $|w| = \text{poly}(|x|)$. Essentially, it states that true claims have *short*[a] proofs.
- **Soundness:** If $x \notin \mathcal{L}_{\mathcal{R}}$, then for any w it holds that $\mathcal{V}(x, w) = 0$. Essentially, it states that false claims have no proofs.

[a]"Short" is a pretty relative term which would further differ based on the context. Here, we assume that the proof is "short" if it can be computed in polynomial time. However, in practice, we will want to make the proofs even shorter: see SNARKs and STARKs.

However, this construction on its own is not helpful to us. In particular, without having any randomness and no interaction, building practical proving systems is very hard. Therefore, we need some more ingredients to make proving NP statements easier.

7.3 Interactive Probabilistic Proofs

As mentioned above, we will bring two more ingredients to the table: **randomness** and **interaction**:

- **Interaction:** rather than simply passively receiving the proof, the verifier $\mathcal{V}$ can interact with the prover $\mathcal{P}$ by sending challenges and receiving responses.
- **Randomness:** verifier can send random coins (challenges) to the prover, which the prover can use to generate responses.

Remark. For those who have already worked with SNARKs, the above introduction might seem very confusing. After all, what we are aiming for is to build **non-interactive** zero-knowledge proofs. However, as it turns out, there are a plenty of ways to make *some* interactive proofs non-interactive. We will discuss this in more detail later.

Now, one of the drastic changes is the following: if $x \notin \mathcal{L}_{\mathcal{R}}$, then the verifier $\mathcal{V}$ should reject the claim with overwhelming[11] probability (in contrast to strict probability of 1). Let us consider the concrete example.

7.3.1 Example: Quadratic Residue Test

Again, suppose for relation $\mathcal{R} = \{(x, w) \in (\mathbb{Z}_N^\times)^2 : x \equiv w^2 \pmod{N}\}$ and corresponding language $\mathcal{L}_{\mathcal{R}} = \{x \in \mathbb{Z}_N^\times : \exists w \in \mathbb{Z}_N^\times \text{ such that } x \equiv w^2 \pmod{N}\}$ the prover $\mathcal{P}$ wants to convince the verifier that the given x is in language $\mathcal{L}_{\mathcal{R}}$. Again, sending w is not an option, as we want to avoid revealing the witness. So how can we proceed? The idea is that the prover $\mathcal{P}$ should prove that he *could* prove it if he felt like it.

So here how it goes. The prover $\mathcal{P}$ can first sample a random $r \xleftarrow{R} \mathbb{Z}_N^\times$, calculate $a \leftarrow r^2 \pmod{N}$ and say to the verifier $\mathcal{V}$:

- Hey, I could give you the square roots of both a and $ax \pmod{N}$ and that would convince you that the statement is true! But in this case, you would know w[12].
- So instead of providing both values simultaneously, you will choose which one you want to see: either r or $r \times w \pmod{N}$. This way, after a couple of such rounds, you will not learn w but you will be convinced that I know it.

That being said, formally the interaction between prover $\mathcal{P}$ and verifier $\mathcal{V}$ can be described as follows:

[11] Some technicality: as you know from the Lecture 2, the value $\varepsilon = \mathsf{negl}(\lambda)$ is called negligible since it is very close to 0. In turn, the value $1 - \varepsilon$ is called *overwhelming* since it is close to 1.

[12] If verifier gets both r and $rw \pmod{N}$, he can divide the latter by former and get w

1. $\mathcal{P}$ samples $r \xleftarrow{R} \mathbb{Z}_N^\times$ and sends $a \leftarrow r^2 \pmod{N}$ to $\mathcal{V}$.
2. $\mathcal{V}$ sends a random bit $b \xleftarrow{R} \{0,1\}$ to $\mathcal{P}$.
3. If $b = 0$, the prover sends $z \leftarrow r$, otherwise, if $b = 1$, he sends $z \leftarrow rw \pmod{N}$.
4. The verifier checks whether $z^2 \equiv a \times x^b \pmod{N}$.
5. Repeat the process for $\lambda \in \mathbb{N}$ rounds.

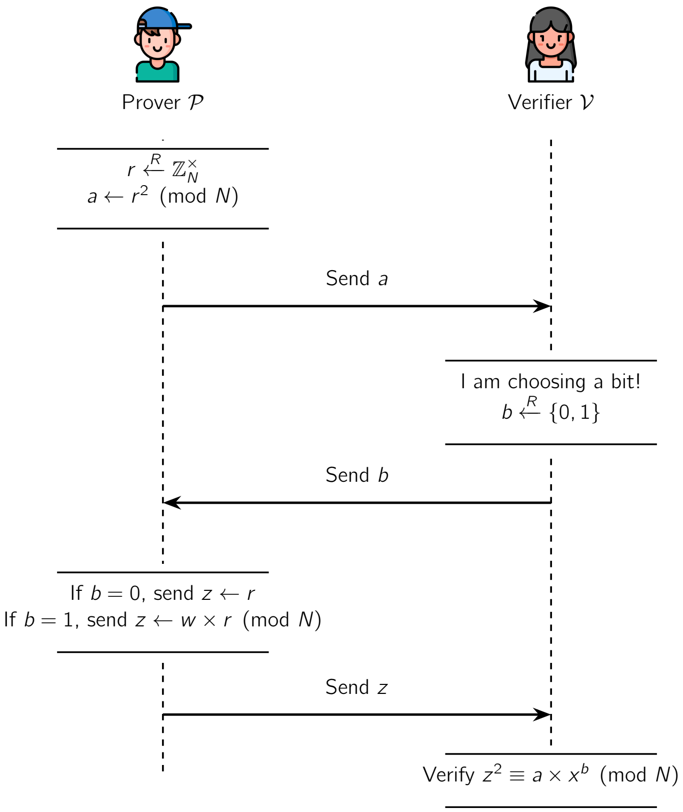

Prover $\mathcal{P}$
Verifier $\mathcal{V}$
$r \xleftarrow{R} \mathbb{Z}_N^\times$
$a \leftarrow r^2 \pmod{N}$
Send a
I am choosing a bit!
$b \xleftarrow{R} \{0,1\}$
Send b
If $b = 0$, send $z \leftarrow r$
If $b = 1$, send $z \leftarrow w \times r \pmod{N}$
Send z
Verify $z^2 \equiv a \times x^b \pmod{N}$

Figure 7.1: The interactive protocol between prover $\mathcal{P}$ and verifier $\mathcal{V}$ for the quadratic residue test.

Now, let us show that the provided protocol is indeed **complete** and **sound**.

Completeness. Suppose the verifier chose $b = 0$ and thus the prover has sent $z = r$. The check would be $r^2 \equiv a \times x^0 \pmod{N}$ which is equivalent to $r^2 \equiv a \pmod{N}$. This obviously holds.

If, in turn, the verifier chose $b = 1$ and the prover sent rw, the check would be $(rw)^2 \equiv ax^{1-0} \pmod{N}$ which is equivalent to $r^2w^2 \equiv ax \pmod{N}$. Since $a = r^2 \pmod{N}$ and $x = w^2 \pmod{N}$, this check also obviously holds.

Soundness. Here, we need to prove that for any dishonest prover who does not know w, the verifier will reject the claim with overwhelming probability. One can show the following, which we are not going to prove (yet, this is quite easy to show):

Proposition 7.3. If $x \notin \mathcal{L}_{\mathcal{R}}$, then for any prover $\mathcal{P}$, the verifier $\mathcal{V}$ will reject the claim with probability at least $1/2$.

By making λ rounds, the probability of rejection is $\left(\frac{1}{2}\right)^{\lambda} = \mathsf{negl}(\lambda)$ and therefore the verifier can be convinced that $x \in \mathcal{L}_{\mathcal{R}}$ with overwhelming probability of $1 - 2^{-\lambda}$.

To denote the interaction between algorithms $\mathcal{P}$ and $\mathcal{V}$ on the statement x, we use notation $\langle \mathcal{P}, \mathcal{V} \rangle(x)$. Finally, now we are ready to define the notion of an **interactive proof system**.

Definition 7.4. A pair of algorithms $(\mathcal{P}, \mathcal{V})$ is called an **interactive proof** for a language $\mathcal{L}_{\mathcal{R}}$ if $\mathcal{V}$ is a polynomial-time verifier and the following two properties hold:

- **Completeness:** For any $x \in \mathcal{L}_{\mathcal{R}}$, $\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(x) = \mathsf{accept}] = 1$.
- **Soundness:** For any $x \notin \mathcal{L}_{\mathcal{R}}$ and for any prover $\mathcal{P}^*$, we have

$$\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle(x) = \mathsf{accept}] \leq \mathsf{negl}(\lambda)$$

Definition 7.5. Besides classes **P** and **NP**, we now have one more class: **the class of interactive proofs** (**IP**):

$$\mathbf{IP} = \{\mathcal{L} : \text{there is an interactive proof } (\mathcal{P}, \mathcal{V}) \text{ for } \mathcal{L}\}.$$

7.4 Zero-Knowledge

Turns out that defining the zero-knowledge to even such a simplistic interactive proof system is not that easy. Informally, we give the following definition.

Definition 7.6. An interactive proof system $(\mathcal{P}, \mathcal{V})$ is called **zero-knowledge** if for any polynomial-time verifier $\mathcal{V}^*$ and any $x \in \mathcal{L}_{\mathcal{R}}$, the interaction $\langle \mathcal{P}, \mathcal{V}^* \rangle(x)$ gives nothing new about the witness w.

Definition 7.7. The pair of algorithms $(\mathcal{P}, \mathcal{V})$ is called a **zero-knowledge interactive protocol** if it is *complete*, *sound*, and *zero-knowledge*.

Basically, the specified interaction is a **proof**! The prover $\mathcal{P}$ can convince the verifier $\mathcal{V}$ that the statement is true without revealing the witness – that is what we need (quite of).

Remark. The above definition is very informal and, for the most part, complete for the purposes of this course. If you do not want to dive into the formalities, you can skip the next part of this section. However, if you are curious about some technicalities, feel free to continue reading.

7.4.1 The Verifier's View

Suppose that the interaction between $\mathcal{V}$ and $\mathcal{P}$ has ended with the successful verification. What has $\mathcal{V}$ learned? Well, first things first, he has learned that the statement is true, that is $x \in \mathcal{L}_{\mathcal{R}}$. However, he has also learned something more: he has learned the transcript of the interaction, that is the sequence of messages between $\mathcal{P}$ and $\mathcal{V}$.

Definition 7.8. Interaction between $\mathcal{P}$ and $\mathcal{V}$ consists of prover's messages (e.g., commitments and responses) $(m_1, m_2, \dots, m_\ell)$, verifier's queries $(q_1, q_2, \dots, q_\ell)$, and $\mathcal{V}$'s random coins $(r_1, r_2, \dots, r_\ell)$. The view of the verifier $\mathcal{V}$, denoted as $\mathsf{view}_{\mathcal{V}}(\mathcal{P}, \mathcal{V})[x]$, is a random variable $(m_1, r_1, q_1, m_2, r_2, q_2, \dots, m_\ell, r_\ell, q_\ell)$ that is determined by the random coins of $\mathcal{V}$ and the messages of $\mathcal{P}$ after the interaction with the statement x. See Figure below.

Example. Suppose that for the aforementioned protocol with $N = 3 \times 2^{30} + 1$, the conversation between the prover $\mathcal{P}$, who wants to convince that $1286091780 \in \mathcal{L}_R$, and $\mathcal{V}$ is the following:

1. During the first round, $\mathcal{P}$ sends 672192003 to $\mathcal{V}$.
2. $\mathcal{V}$ sends $b = 0$ to $\mathcal{P}$.
3. $\mathcal{P}$ sends 2606437826 to $\mathcal{V}$.
4. $\mathcal{V}$ verifies that indeed $2606437826^2 \equiv 672192003 \pmod{N}$.
5. During the second round, $\mathcal{P}$ sends 2619047580 to $\mathcal{V}$.
6. $\mathcal{V}$ chooses $b = 1$ and sends to $\mathcal{P}$.
7. $\mathcal{P}$ sends 1768388249 to $\mathcal{V}$.
8. $\mathcal{V}$ verifies that $1768388249^2 \equiv 2619047580 \times 1286091780 \pmod{N}$.
9. Conversation ends.

The view of the verifier $\mathcal{V}$ is the following:

$$\mathsf{view}_{\mathcal{V}}(\mathcal{V}, \mathcal{P})[1286091780] = (672192003, 0, 2606437826, 2619047580, 1, 1768388249)$$

Essentially, the view that you currently has witnessed is the same as one that $\mathcal{V}$ has seen. After this interaction, you have not learned anything about the witness w that prover $\mathcal{P}$ knows and which we, as of now, has not revealed to you.
In fact, you can verify by yourself, that the witness was $w = 3042517305$ and two randomnesses were $r_1 = 2606437826$ and $r_2 = 3023142760$.
One final note that is essential for the further discussion: variable $\mathsf{view}_{\mathcal{V}}(\mathcal{P}, \mathcal{V})[x]$ is a random variable. For example, for our particular case, both bits could be 0 or both bits could be 1.

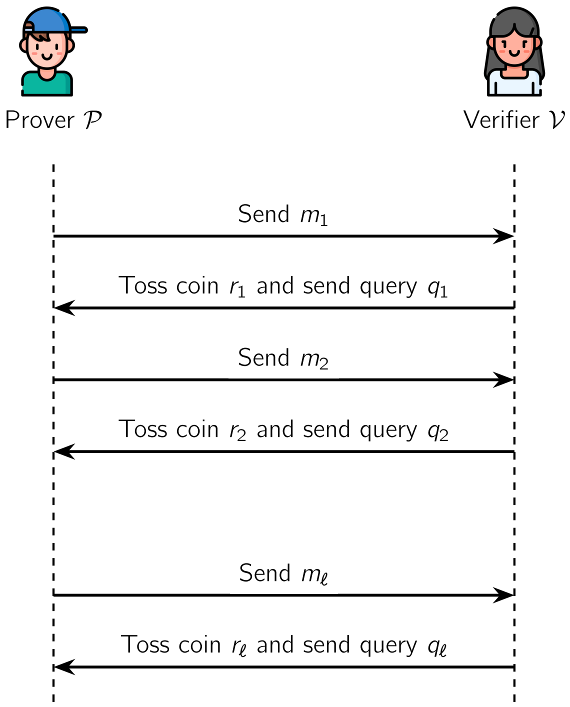

Prover $\mathcal{P}$
Verifier $\mathcal{V}$
Send m_1
Toss coin r_1 and send query q_1
Send m_2
Toss coin r_2 and send query q_2
Send m_ℓ
Toss coin r_ℓ and send query q_ℓ

Figure 7.2: The interactive protocol between prover $\mathcal{P}$ and verifier $\mathcal{V}$. Prover's messages consist of messages $\{m_k\}_{k=1}^{\ell}$, verifier's messages consist of queries $\{q_k\}_{k=1}^{\ell}$, and additionally verifier samples random coins $\{r_k\}_{k=1}^{\ell}$.

7.4.2 The Simulation Paradigm

The key idea is the following: $\mathsf{view}_{\mathcal{V}}(\mathcal{V},\mathcal{P})[x]$ gives nothing new to the verifier $\mathcal{V}$ about the witness w. But it gives nothing new, if he could have *simulated* this view on his own, without even running the interaction. In other words, the "simulated" and "real" views should be *computationally-indistinguishable*. But let us define the computational indistinguishability first.

Definition 7.9 (Computational Indistinguishability)**.** Given two random distributions D_0 and D_1, define the following challenger-adversary game:

1. The challenger randomly samples $x_0 \xleftarrow{R} D_0, x_1 \xleftarrow{R} D_1$ and a bit $b \xleftarrow{R} \{0,1\}$.
2. The challenger sends (x_0, x_1, b) to the adversary.
3. The adversary $\mathcal{A}$ outputs a bit $\hat{b}$.

We define the advantage of the adversary $\mathcal{A}$ in distinguishing D_0 and D_1 as

$$\mathsf{Indadv}[\mathcal{A}, D_0, D_1] = \left|\Pr[b=\hat{b}] - \frac{1}{2}\right|$$

Distributions D_0 and D_1 are called **computationally indistinguishable**, denoted as $D_0 \approx D_1$, if for any polynomial-time adversary $\mathcal{A}$ and polynomial number of rounds in the game, the advantage $\mathsf{Indadv}[\mathcal{A}, D_0, D_1]$ is negligible.

Finally, we are ready to define the **zero-knowledge**.

Definition 7.10 (Zero-Knowledge)**.** An interactive protocol $(\mathcal{P},\mathcal{V})$ is **zero-knowledge** for a language $\mathcal{L}_{\mathcal{R}}$ if for every poly-time verifier $\mathcal{V}^*$ there exists a poly-time simulator Sim such that for any valid statement $x \in \mathcal{L}_{\mathcal{R}}$:

$$\mathsf{view}_{\mathcal{V}^*}(\mathcal{P},\mathcal{V}^*)[x] \approx \mathsf{Sim}(x)$$

However, the condition that verifier might be arbitrary is rather strong. Therefore, we introduce the notion of **honest-verifier zero-knowledge**.

Definition 7.11. Honest-Verifier Zero-Knowledge (HVZK) An interactive protocol $(\mathcal{P},\mathcal{V})$ is **honest-verifier zero-knowledge** for a language $\mathcal{L}_{\mathcal{R}}$ if there exists a probabilistic poly-time simulator Sim such that for any valid statement $x \in \mathcal{L}_{\mathcal{R}}$[a]:

$$\mathsf{view}_{\mathcal{V}}(\mathcal{P},\mathcal{V})[x] \approx \mathsf{Sim}(x)$$

[a]Below, we assume that the verifier $\mathcal{V}$ is honest: he is following the protocol.

7.5 Proof of Knowledge

Now, the main issue with the above definition is that *we have proven the statement correctness, but we have not proven that the prover* **knows** *the witness*. These are completely two different things. Let us demonstrate why.

Example. Suppose that the prover $\mathcal{P}$ wants to convince the verifier that he knows the discrete logarithm of a given point $P \in E(\mathbb{F}_p)$ on a cyclic elliptic curve $E(\mathbb{F}_p)$ of order r. This corresponds to the relation and the corresponding language:

$$\mathcal{R} = \{(P, \alpha) \in E(\mathbb{F}_p) \times \mathbb{Z}_r : P = [\alpha]G\},$$
$$\mathcal{L}_{\mathcal{R}} = \{P \in E(\mathbb{F}_p) : \exists \alpha \in \mathbb{Z}_r \text{ such that } P = [\alpha]G\}$$

But here is the catch: actually, $\mathcal{L}_{\mathcal{R}} = E(\mathbb{F}_p)$ since any point P has a witness α such that $P = [\alpha]G$ (recall that the curve is cyclic)! So proving that $P \in \mathcal{L}_{\mathcal{R}}$ is completely useless! Rather, we want to prove that the prover knows α, not the fact that the point has a discrete logarithm.

That is why instead of **proof**, we need a **proof of knowledge**. This leads to even another weird paradigm used for the rigorous definition: the **extractor**. Basically, the knowledge of witness means that we can *extract* the witness while interacting with the prover. Yet, the *extractor* can do more than the verifier: he can call the prover however he wants and he can also rewind the prover (for example, run some pieces multiple times). This sometimes is referred to as "extractor $\mathcal{E}$ uses $\mathcal{P}$ as an oracle". Now, let us move to the formal definition.

Definition 7.12 (Proof of Knowledge)**.** The interactive protocol $(\mathcal{P}, \mathcal{V})$ is a **proof of knowledge** for $\mathcal{L}_{\mathcal{R}}$ if exists a poly-time extractor algorithm $\mathcal{E}$ such that for any valid statement $x \in \mathcal{L}_{\mathcal{R}}$, in expected poly-time $\mathcal{E}^{\mathcal{P}}(x)$ outputs w such that $(x, w) \in \mathcal{R}$.

Lemma 7.13. The protocol from Section 7.3.1 is a proof of knowledge for the language $\mathcal{L}_{\mathcal{R}}$.

Proof. Let us define the extractor $\mathcal{E}$ for the statement x as follows:

1. Run the prover to receive $a \equiv r^2 \pmod{N}$ (r is chosen randomly from $\mathbb{Z}_N^*$).
2. Set verifier's message to $b = 0$ to get $z_1 \leftarrow r$.
3. **Rewind** and set verifier's message to $b = 1$ to get $z_2 \leftarrow rw \pmod{N}$.

4. Output $z_2/z_1 \pmod{N}$

The extractor $\mathcal{E}$ will always output w if $x \in \mathcal{L}_{\mathcal{R}}$. $\square$

Remark. Note that extractor is given much more than the verifier: he can call the prover multiple times and he can also rewind the prover. This is the main difference between the verifier and the extractor.

7.6 Fiat-Shamir Heuristic

7.6.1 Random Oracle

In cryptography, one frequently encounters the term *cryptographic oracle*. In this section, we are not going to dive into the technical details of what that is, yet it is useful to have a general understanding of what it is.

Definition 7.14 (Cryptographic Oracle)**.** Informally, *cryptographic oracle* is simply a function $\mathcal{O}$ that gives in $O(1)$ an answer to some typically very hard problem.

Example. Consider the Computational Diffie-Hellman (CDH) problem on the cyclic elliptic curve $E(\mathbb{F}_p)$ of prime order r with a generator G. Recall that such problem consists in computing $[\alpha\beta]G$ given $[\alpha]G$ and $[\beta]G$ where $\alpha, \beta \in \mathbb{Z}_r$.
Typically, it is believed that the Diffie-Hellman problem is hard (meaning, for any adversary strategy the probability of solving the problem is negligible). However, we *could* assume that such problem can be solved in $O(1)$ by a cryptographic oracle $\mathcal{O}_{\text{CDH}} : ([\alpha]G, [\beta]G) \mapsto [\alpha\beta]G$. This way, we can rigorously prove the security of some cryptographic protocols *even* if the Diffie-Hellman problem is suddenly solved.

One of the most popular cryptographic oracles is the **random oracle** $\mathcal{O}_{\text{R}}$. Let us define how the random oracle works.

Suppose someone is inputting x to the random oracle $\mathcal{O}_{\text{R}}$. The oracle $\mathcal{O}_{\text{R}}$ does the following:

1. If x has been queried before, the oracle returns the same value as it returned before.
2. If x has not been queried before, the oracle returns a randomly uniformly sampled value from the output space.

Remark. Of course, the sudden appearance of the random oracle is not a magic trick. In practice, the random oracle is typically implemented as a hash function. Of course, formally, the hash function is not a random oracle, yet it is a very good approximation and it is reasonable to assume that the hash function behaves like a random oracle.

7.6.2 Fiat-Shamir Transformation

Now, the main issue with the interactive proofs is that they are... Well, *interactive*. Ideally, we simply want to accumulate a proof π, publish it (say, in blockchain) so that anyone (essentially, being the verifier) could check its validity. So we need some tools to make *some* interactive protocols non-interactive. This is, of course, not always possible, but there are some ways to achieve this.

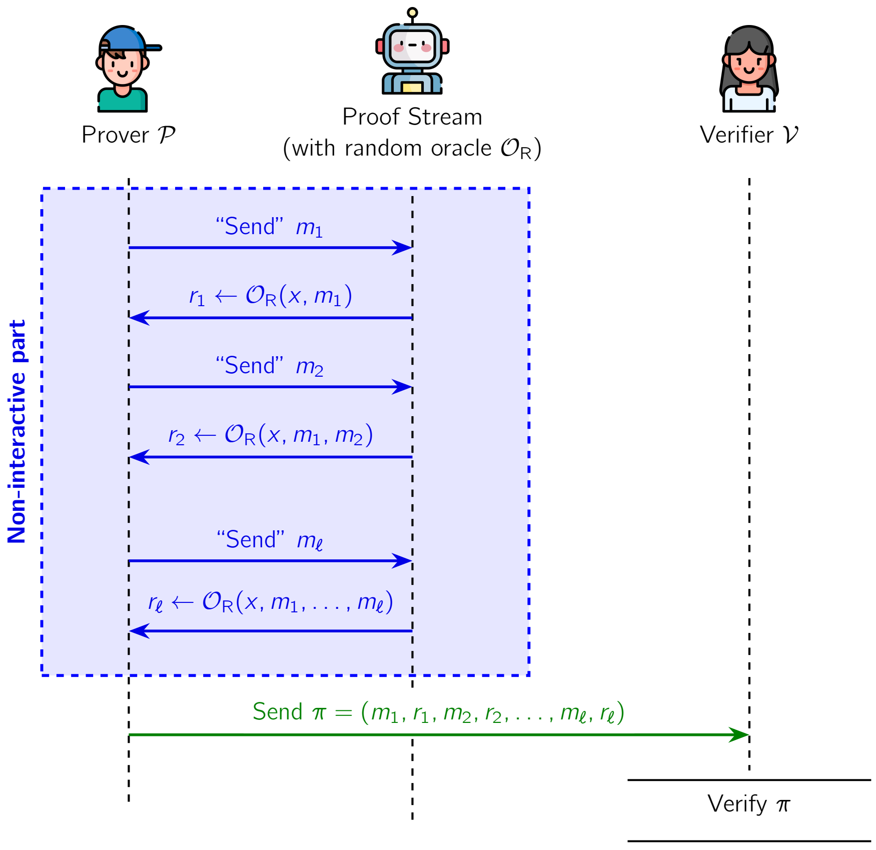
While different protocols use different ways to achieve this, one of the most popular methods (which, in particular, is used in STARKs) is the **Fiat-Shamir heuristic**. The idea is the following: instead of verifier sending the challenges, we can replace them with the random oracle applied to all the previous messages.

Here how it goes. Suppose we have an interactive protocol $(\mathcal{P}, \mathcal{V})$ for the statement x. As previously defined, the interaction between $\mathcal{P}$ and $\mathcal{V}$ consists of prover's messages $(m_1, m_2, \dots, m_\ell)$, verifier's queries $(q_1, q_2, \dots, q_\ell)$, and verifier's random coins $(r_1, r_2, \dots, r_\ell)$. In case all the queries are public random coins, such interactive protocol is called **public-coin protocol** (or, more formally, **Arthur-Merlin protocol**). However, as it turns out, when all the verifier's queries are simply uniformly sampled random values, it is an overkill to use the interactive protocol. Instead, suppose at some point the verifier got messages $m_1, m_2, \dots, m_{\ell'}$ ($\ell' \leq \ell$) from the prover. Then, instead of verifier sampling some random value $r_{\ell'}$, we can simply use the random oracle $\mathcal{O}_{\mathrm{R}}$ as follows: $r_{\ell'} \leftarrow \mathcal{O}_{\mathrm{R}}(x, m_1, m_2, \dots, m_{\ell'})$. Practically, instead of random oracle $\mathcal{O}_{\mathrm{R}}$ we use the hash function H, and use: $r_{\ell'} \leftarrow H(x \parallel m_1 \parallel m_2 \parallel \cdots \parallel m_{\ell'})$.

Remark. Sometimes, to simulate the "interaction" with the verifier, one uses the "Fiat-Shamir Channel". Its main purpose is to simulate the verifier's queries and random coins. For example, one might implement it as a class/struct with the following methods:

1. `send_message(`m`)`: "sends" the message m to the verifier. Under the hood, the proof stream accumulates the current state s and appends m to it.
2. `sample()`: returns the challenge r from the random oracle $\mathcal{O}_{\mathrm{R}}$, applied to the current state s.
3. `get_proof()`: returns the proof π, being the history of interaction, that the prover can publish.

One can check the `winterfell` Rust library or a simpler non-production implementation of the Fiat-Shamir Channel in Golang for more details.



Prover $\mathcal{P}$
Proof Stream (with random oracle $\mathcal{O}_R$)
Verifier $\mathcal{V}$
Non-interactive part
"Send" m_1
$r_1 \leftarrow \mathcal{O}_R(x, m_1)$
"Send" m_2
$r_2 \leftarrow \mathcal{O}_R(x, m_1, m_2)$
"Send" m_ℓ
$r_\ell \leftarrow \mathcal{O}_R(x, m_1, \dots, m_\ell)$
Send $\pi = (m_1, r_1, m_2, r_2, \dots, m_\ell, r_\ell)$
Verify π

Figure 7.3: The non-interactive protocol between prover $\mathcal{P}$ and verifier $\mathcal{V}$ using Fiat-Shamir Transformation. In **blue** we marked a non-interactive part of the protocol, being the "communication" between a prover and a proof stream. In **green** we marked the final proof π that is sent to the verifier.

The process is illustrated in Figure 7.3. The Fiat-Shamir looks as follows:

1. First, the prover $\mathcal{P}$ "sends" the first message m_1 to the verifier $\mathcal{V}$. Here, "sending" is not an actual sending, but rather its simulation.
2. If we had an interactive protocol, the verifier $\mathcal{V}$ would send the random challenge r_1 to the prover $\mathcal{P}$. Instead, we use the random oracle $\mathcal{O}_R$ to get $r_1 \leftarrow \mathcal{O}_R(x, m_1)$.
3. Then, using this challenge, prover does his part in the protocol, and sends the next message m_2.

4. Again, if we had an interactive protocol, the verifier would send the next challenge r_2 to the prover. Instead, we use the random oracle $\mathcal{O}_{\mathrm{R}}$ to get $r_2 \leftarrow \mathcal{O}_{\mathrm{R}}(x, m_1, m_2)$, which gets "sent" to the prover.
5. The process continues until the protocol is finished.

Note that the whole process can be done by a prover with no interaction with the "verifier". In this case, one of the ways to represent the proof π is to publish the transcript of the interaction (that is, all the messages sent by the prover and challenges computed using the random oracle). This is exactly what is done in STARKs.

The reason why such transformation works is that the random oracle $\mathcal{O}_{\mathrm{R}}$ is a *random* function. Therefore, the challenges $r_1, r_2, \dots$ are *random* values, and the prover cannot predict them (for example, by fabricating messages to have some specific output). That being said, the following theorem holds (which, of course, we are not going to prove since the proof is complicated).

Theorem 7.15. Suppose that $(\mathcal{P}, \mathcal{V})$ is a public-coin interactive argument of knowledge for some language $\mathcal{L}_{\mathcal{R}}$ with a negligible soundness error. Then, the Fiat-Shamir transformation of $(\mathcal{P}^{\mathcal{O}_{\mathrm{R}}}, \mathcal{V}^{\mathcal{O}_{\mathrm{R}}})$ is a non-interactive argument for $\mathcal{L}_{\mathcal{R}}$ with negligible soundness error in the random oracle model $\mathcal{O}_{\mathrm{R}}$.

7.7 Exercises

Exercise 1. When dealing with RSA protocol, one frequently encounters the following relation where e is a prime number and $n \in \mathbb{N}$:

$$\mathcal{R} = \left\{(w, x) \in \mathbb{Z}_n^\times \times \mathbb{Z}_n^\times : w^e = x\right\}$$

Which of the following is the language $\mathcal{L}_\mathcal{R}$ that corresponds to the relation $\mathcal{R}$?

(A) Integers from $\mathbb{Z}_n^\times$ which have a modular root of e-th degree.
(B) Integers from $\mathbb{Z}_n^\times$ which are divisible by e.
(C) Integers x from $\mathbb{Z}_n^\times$ with properly defined expression x^e.
(D) Integers from $\mathbb{Z}_n^\times$ which are prime.
(E) Integers from $\mathbb{Z}_n^\times$ for which e is a primitive root.

Exercise 2. Suppose that for some interactive protocol $(\mathcal{P}, \mathcal{V})$ during one round, the probability that the verifier $\mathcal{V}$ accepts a false statement is $1/8$. How many rounds of interaction are needed to guarantee 120 bits of security? Assume here that n bits of security means that the probability of accepting a false statement is at most 2^{-n}.

(A) 30. (B) 40. (C) 60. (D) 90. (E) 120.

Exercise 3. Recall that for relation $\mathcal{R} = \{(w, x) \in \mathbb{Z}_N^\times \times \mathbb{Z}_N^\times : x = w^2\}$ we defined the following interactive protocol $(\mathcal{P}, \mathcal{V})$ to prove that $x \in \mathcal{L}_\mathcal{R}$:

- $\mathcal{P}$ samples $r \xleftarrow{R} \mathbb{Z}_N^\times$ and sends $a = r^2$ to $\mathcal{V}$.
- $\mathcal{V}$ sends a random bit $b \in \{0, 1\}$ to $\mathcal{P}$.
- $\mathcal{P}$ sends $z = r \cdot w^b$ to $\mathcal{V}$.
- $\mathcal{V}$ accepts if $z^2 = a \cdot x^b$, otherwise it rejects.

Suppose we use the protocol $(\mathcal{P}, \mathcal{V}^*)$ where the "broken" verifier $\mathcal{V}^*$ always outputs $b = 1$. Which of the following statements is true?

(A) Both the soundness and completeness of the protocol are preserved.
(B) The soundness of the protocol is preserved, but the completeness is broken.
(C) The completeness of the protocol is preserved, but the soundness is broken.
(D) Both the soundness and completeness of the protocol are broken.

Exercise 4. What is the difference between the cryptographic proof and the proof of knowledge?

(A) Cryptographic proof is a proof of knowledge that is secure against malicious verifiers.

(B) Cryptographic proof is a proof of knowledge that is secure against malicious provers.

(C) Cryptographic proof merely states the correctness of a statement, while the proof of knowledge also guarantees that the prover knows the witness.

(D) While cryptographic proof states that witness exists for the given statement, the proof of knowledge makes sure to make this witness unknown to the verifier.

(E) Proof of knowledge does not require verifier to know the statement, while cryptographic proof does.

Exercise 5. What is the purpose of introducing the extractor?

(A) To introduce the algorithm that simulates the malicious verifier trying to extract the witness from the prover.

(B) To define what it means that the prover knows the witness.

(C) To give the verifier the ability to extract the witness from the prover during the interactive protocol.

(D) To define the security of the interactive protocol that uses a more powerful verifier that can extract additional information from the prover.

(E) To give prover more power to extract randomness generated by the verifier.

Exercise 6. What it means that the interactive protocol $(\mathcal{P}, \mathcal{V})$ is a zero-knowledge?

(A) The verifier $\mathcal{V}$ cannot know whether the given statement is true or false.

(B) The verifier $\mathcal{V}$ cannot know whether the prover $\mathcal{P}$ knows the witness.

(C) View of the prover $\mathcal{P}$ in the protocol is indistinguishable from the view of the verifier $\mathcal{V}$.

(D) Any view of any verifier $\mathcal{V}$ can be simulated using some polynomial-time algorithm, outputting computationally indistinguishable distribution from the given view.

(E) The prover $\mathcal{P}$ can convince the verifier $\mathcal{V}$ that the statement is true without knowing the witness.

Hint: View of the participant in the protocol consists of all data he has access to during the protocol execution. For example, verifier $\mathcal{V}$'s view consists of the messages he sends and receives, as well as the random coins he generates.

contact@distributedlab.com