ZKDL Camp Lecture Notes

Distributed Lab

July 24, 2024

1 Mathematics for Cryptographers I

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

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 (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 $A^n := \underbrace{A \times A \times \cdots \times A}_{n \text{ times}}$ – 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 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

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

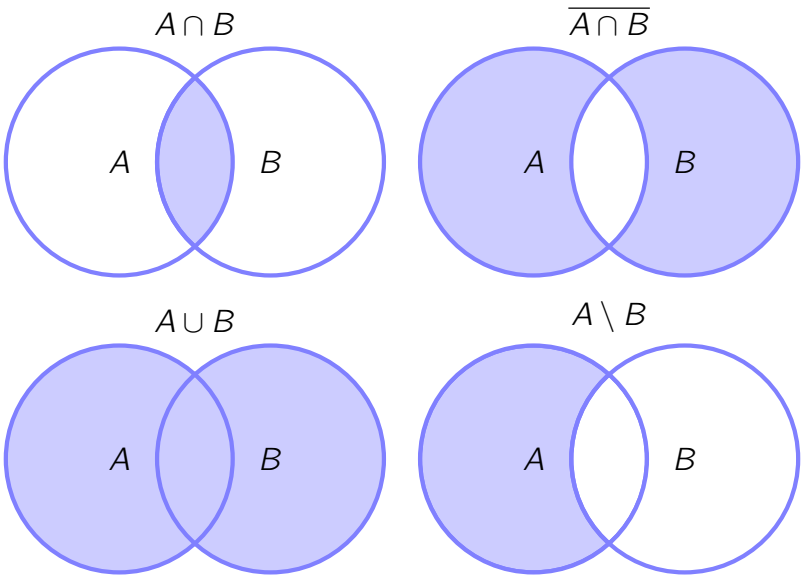

Figure 1: Set operations illustrated with Venn diagrams.

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 $a \xleftarrow{R} A$. For example, $a \xleftarrow{R} \{1, 2, 3, 4, 5, 6\}$ 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 $\langle \mathbf{x}, \mathbf{y} \rangle := \sum_{k=1}^{n} x_k y_k$.

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

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

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 2.

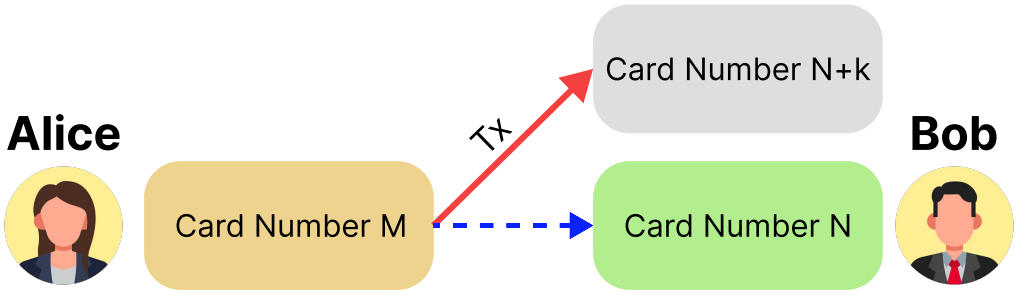

Figure 2: Alice pays to Bob to a card number N, but our awesome system pays to $N+k$ instead. Bob would not be happy...

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.

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

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

Then, it is easy to verify that

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

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 $e_1 e_2$. From the definition of the identity element, we know that $e_1 e_2 = e_1$ and $e_1 e_2 = e_2$. 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,

$$ag = ga = e, \quad bg = gb = e. \tag{3}$$

Now, notice that

$$a = ae = a(gb) = (ag)b = eb = b \tag{4}$$

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

$$C_g = \{h \in \mathbb{G} : hg = gh\} \tag{5}$$

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

$$\langle g \rangle = \{g^n : n \in \mathbb{Z}\} = \{\dots, g^{-3}, g^{-2}, g^{-1}, e, g, g^2, g^3, \dots\}. \tag{6}$$

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

$$\langle 2 \rangle = \{2, 4, 6, 8, 10, 0\} \tag{7}$$

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,

$$\forall a, b \in \mathbb{G} : \varphi(a \oplus b) = \varphi(a) \odot \varphi(b). \tag{8}$$

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:

$$\varphi(a + b) = (a + b) \bmod 12 = (a \bmod 12) + (b \bmod 12) = \varphi(a) + \varphi(b). \tag{9}$$

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:

$$\varphi(a + b) = e^{a+b} = e^a \cdot e^b = \varphi(a) \cdot \varphi(b). \tag{10}$$

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 $\mathbb{Z}_p := \{0, 1, 2, \ldots, p-2, p-1\}$. 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 $12 \equiv 2 = c'$ (mod 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:

$$ax \equiv 1 \pmod{p}. \tag{11}$$

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 $\{0, 1, \ldots, p-2, p-1\}$, 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

$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n = \sum_{k=0}^{n} c_k x^k, \tag{12}$$

where $c_0, c_1, \ldots, c_n$ 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

$$\mathbb{F}[x] = \left\{ p(x) = \sum_{k=0}^{n} c_k x^k : c_k \in \mathbb{F},\ k = 0, \ldots, n \right\}. \tag{13}$$

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 $p(x) = c_0 + c_1 x + c_2 x^2 + \dots$ 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

$$p(x) = (x - x_0)q(x) \tag{14}$$

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

$$f = q \cdot g + r, \ 0 \leq \deg r < \deg g \tag{15}$$

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:

$$f \operatorname{div} g = x^2 - x + 1, \quad f \bmod g = 1. \tag{16}$$

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 $(c_0, c_1, \ldots, c_n) \in \mathbb{F}^n$. 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 3. Now, let us see how we can interpolate the polynomial practically.

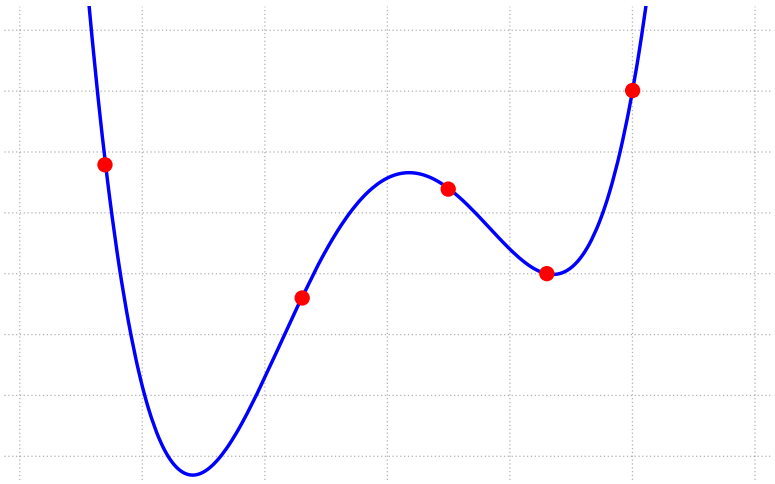

Figure 3: 5 points on the plane uniquely define the polynomial of degree 4.

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

$$L(x) = \sum_{i=0}^{n} y_i \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}. \tag{17}$$

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

$$\ell_i(x_j) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, \tag{18}$$

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 $\alpha_0, \ldots, \alpha_n \in \mathbb{F}$ such that

$$p(x) = \sum_{i=0}^{n} \alpha_i \ell_i(x). \tag{19}$$

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

$$L(x) = 1 \cdot \frac{x-1}{0-1} + 2 \cdot \frac{x-0}{1-0} = (-1) \cdot (x-1) + 2 \cdot x = x + 1 \tag{20}$$

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 $\alpha_1, \ldots, \alpha_n \in F$ 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 $(\alpha_1, \ldots, \alpha_t)$ 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 $\mathcal{I} \subset \{1, \ldots, n\}$ of size t.

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

$$\mathsf{Comb}(\mathcal{I}, \{\alpha_i\}_{i\in\mathcal{I}}) = \alpha. \tag{21}$$

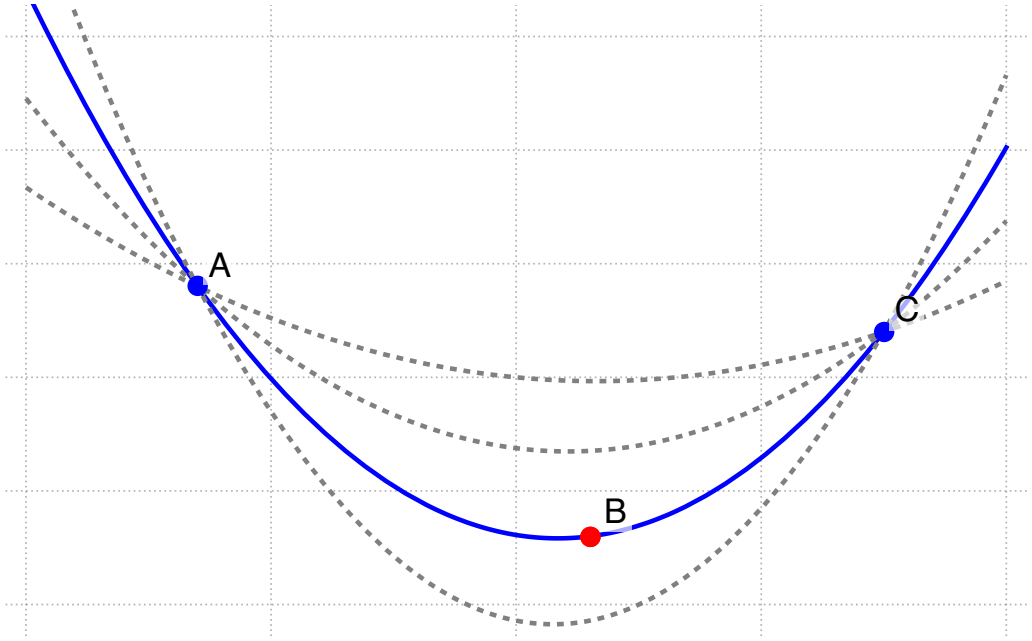

Figure 4: Suppose we have $t = 3$. Having only 2 points means knowing two blue points without knowing the red one. There are infinitely many quadratic polynomials passing through these two points (gray dashed lines). However, knowing the third red point allows us to uniquely determine the polynomial and thus get its value at 0. Note that this is illustrated over $\mathbb{R}$, but for $\mathbb{F}_q$ the logic is similar.

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 $k_1, \ldots, k_{t-1} \xleftarrow{R} \mathbb{F}_q$ and define the polynomial

$$\omega(x) := \alpha + k_1 x + k_2 x^2 + \cdots + k_{t-1} x^{t-1} \in \mathbb{F}_q^{\leq (t-1)}[x], \tag{22}$$

  and then compute $\alpha_i \leftarrow \omega(i) \in \mathbb{F}_q,\ i = 1, \ldots, n$. Return $(\alpha_1, \ldots, \alpha_n)$.
- $\mathsf{Comb}(\mathcal{I}, \{\alpha_i\}_{i \in \mathcal{I}})$: 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 4.

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

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

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

```rust
/// 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:

```rust
/// 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
        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()
    }
```

```rust

    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.

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 $(x_1, y_1) \oplus (x_2, y_2) = (x_1 x_2, y_1 y_2)$.

Which statements are **true**?

a) Only 1.

b) Only 1 and 2.

[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.

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) This polynomial has no roots over $\mathbb{F}_5$.

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 to determine.

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 $\varphi : \mathrm{GL}(2, \mathbb{R}) \to \mathbb{R}$:

$$\varphi_1\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc,\ \varphi_2\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = a + d \tag{23}$$

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 Mathematics for Cryptographers II

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**:

$$(\forall k \in \mathcal{K})\,(\forall m \in \mathcal{M}) : \{D(k, E(k, m)) = m\} \tag{24}$$

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 $\hat{b}$.

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:

$$\mathrm{SSAdv}[\mathcal{E}, \mathcal{A}] := \left| \Pr\left[\hat{b} = b\right] - \frac{1}{2} \right| \tag{25}$$

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 $1/2$.

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

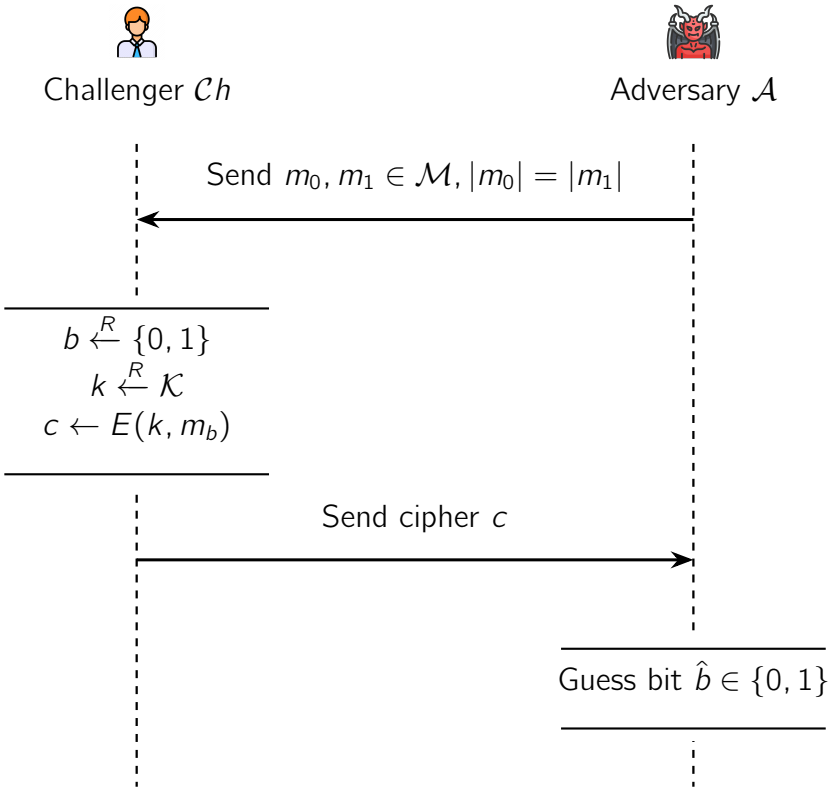

Figure 5: The game between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}h$ for defining the semantic security.

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

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

$$\lim_{n\to\infty} f(n)n^c = 0 \tag{26}$$

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

$$\lim_{n\to\infty} 2^{-n}n^c = 0 \tag{27}$$

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

$$\lim_{n\to\infty} \frac{1}{n} \times n = 1 \neq 0 \tag{28}$$

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:

$$\left| \Pr\left[b = \hat{b} \;\middle|\; \begin{array}{c} m_0, m_1 \leftarrow \mathcal{A},\ k \xleftarrow{R} \mathcal{K},\ b \xleftarrow{R} \{0,1\} \\ c \leftarrow E(k, m_b) \\ \hat{b} \leftarrow \mathcal{A}(c) \end{array} \right] - \frac{1}{2} \right| < \mathrm{negl}(\lambda) \tag{29}$$

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 Discrete Logarithm Assumption (DL)

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

Definition 2.4. 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 2.5. 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!

2.1.3 Computational Diffie-Hellman (CDH)

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 2.6. 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 $\mathsf{CDHadv}[\mathcal{A}, \mathbb{G}]$, as the probability that $\hat{w} = w$.

Definition 2.7. The **Computational Diffie-Hellman Assumption** holds in the group $\mathbb{G}$ if for any efficient adversary $\mathcal{A}$ the advantage $\mathsf{CDHadv}[\mathcal{A}, \mathbb{G}]$ is negligible.

2.2 Decisional Diffie-Hellman (DDH)

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 $\mathbb{G}$. For-

mally, we define this problem as follows.

Definition 2.8. 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 $\mathsf{DDHadv}[\mathcal{A}, \mathbb{G}]$, as

$$\mathsf{DDHadv}[\mathcal{A}, \mathbb{G}] := \left| \Pr\left[b = \hat{b}\right] - \frac{1}{2} \right| \tag{30}$$

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

Theorem 2.9. 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 2.10. 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

$$u^n = (g^{2\alpha'})^n = g^{2n\alpha'} = (g^{2n})^{\alpha'} = 1^{\alpha'} = 1 \tag{31}$$

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. ■

2.2.1 Why this is needed?

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.

2.3 Basic Number Theory

2.3.1 Primes

Primes are often used when doing almost any cryptographic computation. A prime number is a natural number ($\mathbb{N}$) that is not a product of two smaller natural number. In other words, the prime number is divisible only by itself and 1. The first primes are: 2, 3, 5, 7, 11...

2.3.2 Deterministic prime tests

A primality test is deterministic if it outputs `True` when the number is a prime and `False` when the input is composite with probability 1. An example of a deterministic prime test is `Trial_Division_Test`. Here is an example implementation in Rust:

```rust
fn is_prime(n: u32) -> bool {
    let square_root = (n as f64).sqrt() as u32;

    for i in 2.. = square_root {
        if n % i == 0 {
            return false;
        }
    }

    true
}
```

Deterministic tests often lack efficiency. For instance, even with square root optimization, the asymptotic complexity is $O(\sqrt{N})$. While further optimizations are possible, they do not change the overall asymptotic complexity.

In cryptography, N can be extremely large — 256 bits, 512 bits, or even 6144 bits. An algorithm is impractical when dealing with such large numbers.

2.3.3 Probabilistic prime tests

A primality test is probabilistic if it outputs `True` when the number is a prime and `False` when the input is composite with probability less than 1. Such test is often called a pseudoprimality test. Fermat Primality and Miller-Rabin Primality Tests are examples of probabilistic primality test. Both of them use the idea of **Fermat's Little Theorem**:

Theorem 2.11. Let p be a prime number and a be an integer not divisible by p. Then $a^{p-1} - 1$ is always divisible by p: $a^{p-1} \equiv 1 \pmod{p}$

The key idea behind the Fermat Primality Test is that if for some a not divisible by n we have $a^{n-1} \not\equiv 1 \pmod{n}$ then n is definitely NOT prime. Athough, with such an approach, we might get a false positive, as you cannot state for sure that n is prime. For example, consider $n = 15$ and $a = 4$. $4^{15-1} \equiv 1 \pmod{15}$, but $n = 15 = 3 \cdot 5$ is composite. To solve this issue, a is picked many times, decreasing the chances of a false positive. The probability that a composite number is mistakenly called prime for k iterations is $2^{-k} = \frac{1}{2^k}$.

There exists a problem with such an algorithm in the form of **Carmichael numbers**, which are numbers that are Fermat pseudoprime to all bases. To put it simply, no matter how many times you check whether the number is prime using this type of primality test, it will always stay positive, even though the number is composite. The good thing is that Carmichael numbers are pretty rare. The bad thing is that there are infinitely many of them.

Even though this algorithm is probabilistic (which does not guarantee the correctness of the output) and has a vulnerability in the form of *Carmichael numbers*, it runs with an asymptotic complexity $O(\log^3 n)$. This is much better for large numbers and is often used in cryptography. Here is a pseudocode implementation of this algorithm:

```
# n = number to be tested for primality
# k = number of times the test will be repeated
def is_prime(n, k):
    i = 1
    while i <= k:
        a = rand(2, n - 1)

        if a^(n - 1) != 1 (mod n):
            return False

        i++

    return True
```

Miller-Rabin primality test, is a more advanced form of Fermat primality test. The main difference is it is not vulnerable to *Carmichael numbers*, which makes it much better to use in practice.

2.3.4 Greatest Common Divisor

Greatest common divisor (GCD) of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers.

> **Example.** $gcd(8, 12) = 4$, $gcd(3, 15) = 3$, $gcd(15, 10) = 5$.

Computing GCD using Euclid's algorithm. The is based on the fact that, given two positive integers a and b such that $a > b$, the common divisors of a and b are the same as the common divisors of $a - b$ and b. It can be observed, that it can be further optimized, by using $amodb$, instead of $a - b$. For example, $gcd(26, 8) = gcd(18, 8) = gcd(10, 8) = gcd(2, 8)$ can be optimized to $gcd(26, 8) = gcd(26 \pmod 8, 8) => gcd(2, 8)$ Algorithm can be implemented using recursion. Base of the recursion is $gcd(a, 0) = a$.

```
int gcd(a, b):
 if (b == 0):
     return a
 return gcd(b, a % b)
```

Provided algorithm work with $O(log(N))$ asymptotic complexity.

2.3.5 Least common multiple

Least common multiple (LCM) of two integers a and b, is the smallest positive integer that is divisible by both a and b.

The least common multiple can be computed from the greatest common divisor with the formula: $lcm(a, b) = \frac{|ab|}{gcd(a,b)}$

```
int lcm(a, b):
 return a * (b / gcd(a, b))
```

2.3.6 Modular inverse

Modular multiplicative inverse of an integer a is an integer b such that $a \cdot b \equiv 1 \pmod{m}$. In prime fields it is commonly used as a division operation.

One of the ways to compute the modular inverse is by using Euler's theorem:
$a^{\phi(m)} \equiv 1 \pmod{m}$, where ϕ is Euler's totient function.
For prime numbers, where $\phi(m) = m - 1$:
$a^{m-2} \equiv a^{-1} \pmod{m}$.

```
a_inverse = powmod(a, m-2, m) # where powmod(base, power,
↪ modulus)
```

2.3.7 Reed-Solomon codes

Reed-Solomon codes allows to restore lost or corrupted data, implement threshold secret sharing and is used in some ZK protocols. Given a vector of data V a polynomial P is constructed using Lagrange interpolation. Polynomial with degree n can be uniquely defined using $(n + 1)$ unique points. Defining more points on the same polynomial add a redundancy, which can be used to restore the polynomial even if some points are missing. Common choices for a set of evaluation points include $0, 1, 2, ..., n - 1$.

The error-correcting ability of a Reed-Solomon code is $n - k$, the measure of redundancy in the block. If the locations of the error symbols are not known in advance, then a Reed-Solomon code can correct up to $n - k/2$ erroneous symbols, i.e., it can correct half as many errors as there are redundant symbols added to the block.

2.3.8 Schwartz-Zippel Lemma

Lemma 2.12. 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 $\leq \frac{d}{|S|}$.

2.4 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 $\mathsf{SSAdv}[\mathcal{A}] = \left|\Pr\left[b = \hat{b}\right] - \frac{1}{2}\right|$, 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 ($\mathrm{negl}(\lambda)$).

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 $m \xleftarrow{R} \mathcal{M}, k \xleftarrow{R} \mathcal{K}$.
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.