UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques
i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

A Formal Introduction to Zero-Knowledge Proofs

Autor: Antonio Peso Vilella

Directors: Mr. Bruno Mazorra
Dr. Luís Víctor Dieulefait

Realitzat a: Departament de
Matemàtiques-Informàtica

Barcelona, June 10, 2024

Contents

Abstract

The idea of *zero-knowledge proof* was first introduced by Goldwasser, Micali and Rackoff [GMR89] and has found its way to many real-world applications. The growing need for privacy in information exchange (e.g. transactions, digital signatures, commitment schemes, ...) lead to the development of proofs that yield nothing more than their validity.

We introduce the building blocks for zero-knowledge proofs through mathematical rigour, allowing the reader to gain a solid foundation to research further related topics. We explore some necessary notions of cryptography and probability, as well as computation theory by utilizing Turing machines as an automation abstraction. We delve into the theory of decision problems and the consequent classification through complexity classes, specially $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{BPP}$ and $\mathcal{IP}$. We use the concepts of *repetition* and *interaction* to prove that the decision error for languages in $\mathcal{BPP}$ and $\mathcal{IP}$ can be decreased exponentially and explore the example of *Graph Non-Isomorphism*. We introduce the idea of *zero-knowledge interactive proof systems* and define some variations of its definition. We explore the example of *Graph Isomorphism* and conclude showing that the sequential repetition of zero-knowledge interactive proofs is indeed a zero-knowledge interactive proof.

2020 Mathematics Subject Classification. 03D10, 03D15, 68Q04, 68Q10, 68V15

Chapter 1

Introduction

Information is a powerful asset in today's digitalized world. The need to keep private data *private* has been a major topic in Cryptography and Mathematics and new models of information exchange have appeared in recent years.

Zero-Knowledge proofs were first introduced in the late 80s [GMR89] and are now being embedded in many systems and protocols for their main property: *conveying no additional information other than the least necessary*. Their use is booming with the rise of *Blockchain* technologies and *Cryptocurrencies*, where public transactions need to be verified while keeping information of each party secret. Let us explore an illustrative example of zero-knowledge proof systems:

> `Alice` and `Bob` are in front of a circular cave with a single entrance. The only way to walk through the entire cave is by using a door in the middle of the cave that can only be opened with a magic word. `Alice` claims that she knows the magic word and `Bob` asks her to prove it. `Alice` doesn't want anyone to know the magic word. Also, she doesn't want anyone except for `Bob` to know whether she knows the magic word or not, so `Bob` and her decide to follow a process to convince him and keep any unwanted third party from finding out whether she knows it or not.
>
> There are only two ways to reach the door from the outside entrance: left (i.e. path `A`) and right (i.e. path `B`). The "proof" proceeds as follows:
>
> - `Bob` waits outside the entrance and `Alice` goes in the cave. While `Bob` is outside and unable to see `Alice`, she chooses a path between `A` and `B` and goes to the door.
> - `Bob` enters the cave and screams either `A` or `B`. If `Alice` knows the magic

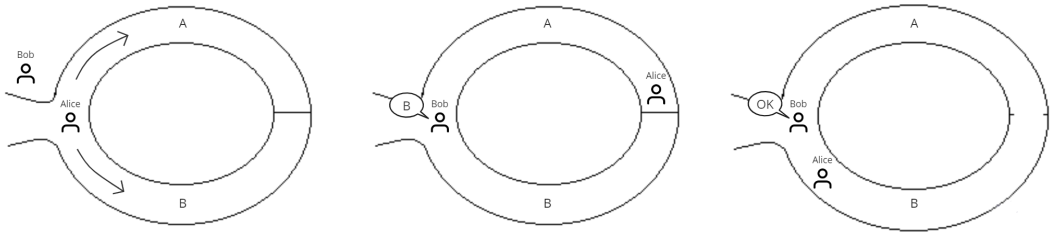

Figure 1.1: Scheme of a successful iteration

word, then she will exit the cave through the path instructed by `Bob`.

- If `Alice` exits through the wrong side, then it will be clear that she doesn't know the magic word. However, if she exits through the right side, then `Bob` can be convinced that she *might* know the magic word (with 50% probability).

Since `Alice` could have been (by chance) on the same path instructed by `Bob`, they decide to repeat the process enough times to convince `Bob` that she *most likely* knows the magic word. If a third party was spying on them, the only thing they would see is `Alice` going in the cave, `Bob` going in as well and then both of them exiting the cave. Hence, if they wanted to know whether `Alice` knows the word, they would have to redo the whole process (or ask `Bob`, but he could be untrustworthy).

This example clearly illustrates what we already introduced: a first entity (`Alice`) with undisclosed information (the magic word) and a second entity (`Bob`) who wants to be convinced of the validity of the first entity's claim. It would be easy for the first entity with the secret information (i.e. the *prover*) to just disclose it to the second entity (i.e. the *verifier*), but it is often not desirable to do so. We will be formalizing all these ideas and proving some results that are crucial for the secure implementation of *protocols* with the property of "yielding nothing more than the validity of the claim" (i.e. *zero-knowledge proof systems*).

The contents of this work are probability based, meaning that there is a chance that our protocols can fail. To mitigate this problem, we will use the power of *repetition* and *independence* to reduce this error as much as necessary, making it *negligible* for all practical purposes. It seems somehow intuitive that, repeating many times an action that has a chance of succeeding to prove something, and we succeed every time (or most times in some cases) we will be confident on the validity of the proof. It is shown in this work that, if we accept a *really* small chance

of error, we can prove claims revealing "no knowledge" (i.e. "zero knowledge") on the specifics of the proof.

It seems pretty general to talk about *information* and *knowledge*. We will discuss further what we mean in Section 3.2.2, but we could sum it up as *computational power*. Zero-knowledge will be used to verify claims on problems that are hard to *compute* (e.g. factoring a large natural number into its prime factors). If an entity knows something that is not computable in a reasonable amount of time (without some extra data), then it is understandable that, unless it possesses such information, it won't be able to solve a related problem (we won't normally ask to solve the original claim, since it would reveal too much information; instead, we will pose some other problems that can only be solved *if* the prover entity actually has the knowledge it claims to have).

This computational component leads to the introduction of *decision problems* and *complexity theory*. We will classify problems by their *computational complexity* and delve into those with a probabilistic aspect. This will relate with the aforementioned "acceptable" error in zero-knowledge proofs and will lead to the formal definition of these. It will be made clear that this definition can vary depending on its practical use, but it will always focus on keeping *any verifier* entity away from any knowledge known by the *prover* entity.

The last underlying idea in this work is *interaction*. In the cave example above, we clearly see how the two entities need to interact with one another. We will explore the interaction between machines (i.e. *Turing machines*) to build *interactive proof systems*, which are the building blocks for zero-knowledge *interactive proof systems*. Though we will end our discussion considering interaction between two parties, in real-world applications these are expensive in terms of time and computation resources. This leads to the study of *non-interactive* systems and protocols, which we propose as a continuation of this work to match the current state of investigation in the field.

1.1 Organization of the Work

This work is heavily based on [Gol06] and follows a similar schema as chapters 1 and 3 of said resource. The organization's intention is to lay the foundations needed to define and understand the concept of *zero-knowledge interactive proof systems*.

The content is split in three different parts: preliminaries, interaction and zero-knowledge proofs. In particular,

1. *Preliminary* introduces some basic concepts related to *cryptography* and results from *probability theory* used in other sections.
2. *Interaction* focuses on computational theory (Turing machines and complexity classes) and the effect of interaction on proof systems (with an example on *Graph Non-Isomorphism*).
3. *Zero-Knowledge Proofs* culminates with the main topic of this work through definitions, formal and informal examples (e.g. *Graph Isomorphism*) and important results.

Every section is built on top of the previous one, so a reader with knowledge on the first (or second) parts, can skip them and go to the next.

At the end, there are two appendixes:

A *Quadratic Resiudosity Protocol* contains a practical example of the zero-knowledge proof system presented in Section 4.1.1. It consists of a computer program in *C* language and its results.

B *Graph Isomorphism Proof* contains a separate part of the extensive proof from Proposition 4.5.

1.2 Notation Cheatsheet

Symbol	Description
$X, Y, Z, \ldots$	Random variables
$\mathbb{E}(X)$	Expected value of the variable X
$\mathrm{Var}(X)$	Variance of the variable X
$\Pr[X = x]$	Probability of X to be equal to x (similar for $\geq$ and $\leq$)
Σ	Alphabet
$a, b, c, \ldots$	Letters or symbols of an alphabet Σ
$x, y, z, \ldots$	Strings or sequence of letters
L	Language or set of strings
(x, L)	Statement s.t. $x \in L$
$\lvert x \rvert$	Length of x
$\{0,1\}^*$	Set of all finite strings
$p(\cdot), t(\cdot)$	Positive polynomials p and t
M	Machine
$M(x)$	Machine's output on input x
R_L	Boolean Relation
$\mathcal{P}, \mathcal{NP}, \mathcal{BPP}, \mathcal{IP}$	Complexity Classes
ITM	Interactive Turing Machine
σ	Identity of a machine
P	Prover
V	Verifier
$\langle A, B \rangle(x)$	Output of a pair of ITMs on input x
$c(\cdot)$	Completeness bound
$s(\cdot)$	Soundness bound

Table 1.1: Notation

Chapter 2

Preliminary

2.1 Introduction to Cryptography

Cryptography (in the modern sense) aims to create and build schemes that can handle information with secrecy (i.e. keep secrets away from any unwanted entity). These schemes must withstand any malicious attempts of deviating them from their desired functionality.

The design of cryptographic systems shouldn't be based on possible strategies taken by some malicious party (since these could escape our imagination or appear with technological breakthroughs) but based on strong and firm foundations. The only assumptions to be taken on any adversary are related to their computational power. In this sense, cryptography can be said to be based on computational complexity.

With the arrival of the digital era, the topics embraced by cryptography have grown in number.

2.1.1 Encryption Schemes

Originally, cryptography focused on the construction of *encryption schemes* (i.e., schemes that provide secret communication over insecure media). These schemes consist of two algorithms: an *encryption* and a *decryption* algorithm. There are normally two parties involved in the process: a sender and a receiver. The sender first applies the encryption algorithm to the original message (called *plaintext*), obtaining the encrypted message (called *ciphertext*). Then, he sends the ciphertext to the receiver, who proceeds to apply the decryption algorithm retrieving this way the original message.

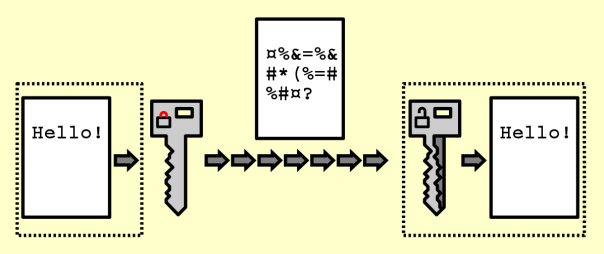

Figure 2.1: Encryption scheme

If there would be a third malicious party "listening" on the communication (say a *wire-tapper*) they would only be able to read the ciphertext. The wire-tapper wants to gain insight on the plaintext and he could very well know which decryption algorithm is being used. To prevent this, the receiver must have some information that is not known to anyone else (maybe just the sender, if it's a trusted party). This secret information is known as the *decryption key*.

At this point, it is fair to raise the question of what do we mean by "security". The modern definition is based on the *computational complexity* to extract any information about the plaintext from the cihpertext. In this modern sense, we are pondering the question of the *feasibility* of the endeavour, not whether it is *possible* or not. For example, using "pseudo-random generators" (see [Gol06]) one can increase the length of a key to make it seem longer and more secure. Another example that stems from the computational complexity definition is *public-key encryption schemes* (see [KL07]) which have an *asymmetric key distribution*. In these schemes, the encryption key is released to the public, who is able to encrypt the plaintext. The catch is that there exists a private key needed to decrypt the ciphertext. So even if someone has a plaintext and an encryption key, it is not *feasible* to extract the decryption key.

2.1.2 Digital Signatures

The digital era brought great changes to the way people did business and a new problem arose when trying to secure commitments between different parties: the need to create unique and non-reproducible signatures that could identify a party in a digital setting. *Digital signatures* are, roughly speaking, the digital version of handwritten signatures. These new signatures needed to fulfill three requirements:

- to be *efficiently generated* on a specific document,

- to be *efficiently verified* as a signature from a specific party on a specific document, and
- to not be *efficiently reproducible* by someone other than the original party.

In other words, the aim is to provide all parties of the system a way of making self-binding statements and of ensuring that one party cannot make statements that would bind another party.

Following this train of thought, one could redefine cryptography as the study of problems in which one wishes to limit the effects of dishonest parties. These problems are treated in a more general way by "fault-tolerant" protocols.

2.1.3 Fault-Tolerant Protocols

A *cryptographic* or "fault-tolerant" *protocol* (see [Sch24]) is a distributed algorithm describing precisely the interactions between two or more entities, achieving certain security objectives.

Cryptographic protocols concern themselves with problems such as simultaneity (e.g. contract signing), secure implementation (e.g. implementing secret and incorruptible voting) and *zero-knowledge* proofs. In this work, we will focus entirely on the latter problem. We will give the theoretical basis to define and understand zero-knowledge proof systems from a probabilistic and computational point of view as explained in [Gol06].

2.2 Probability Theory Background

This section focuses on probabilistic concepts used throughout this work and expects some basic prior knowledge from the reader (e.g. probability distribution, expected value, variance, etc.). We will work only with discrete random variables, unless stated otherwise.

2.2.1 Basic Concepts

Throughout this work, we will consider sets and sequences of discrete random variables. Next, we will introduce some of the most important ideas that we will need.

Definition 2.1. *Two random variables X and Y are **identically distributed** if each random variable has the same probability distribution.*

Definition 2.2. *Two random variables X and Y are **independent** if $Pr[X = x \cap Y = y] = \Pr[X = x] \cdot \Pr[Y = y]$, where x and y are possible values for X and Y, respectively.*

Definition 2.3. *A **Bernoulli distribution** is a discrete distribution that has two possible outcomes: a success (denoted by 1) and a failure (denoted by 0). We denote the probability of success by p and the probability of failure by $1 - p$. If a random variable X follows a Bernoulli distribution with parameter p, we write $X \sim Bernoulli(p)$.*

It will be useful to know the expected value and the variance of a Bernoulli distribution. Let $X \sim Bernoulli(p)$, then

$$\mathbb{E}(X) = p$$

$$\mathrm{Var}(X) = p(1 - p)$$

We will be working with Bernoulli distributions where the probability is not explicitly given, but instead it is bounded. Namely, let X_1 and X_2 be Bernoulli distributions with bounded probability and let $p \in [0, 1]$ be the lower (resp. upper) bound for the success probability of X_1 (resp. X_2). Then

$$\Pr[X_1 = 0] \leq 1 - p,\ \Pr[X_1 = 1] \geq p$$

or

$$\Pr[X_2 = 0] \geq 1 - p,\ \Pr[X_2 = 1] \leq p$$

The expected values are:

$$\mathbb{E}(X_1) = \Pr[X_1 = 0] \cdot 0 + \Pr[X_1 = 1] \cdot 1 = \Pr[X_1 = 1] \geq p$$
$$\textit{and}$$
$$\mathbb{E}(X_2) = \Pr[X_2 = 0] \cdot 0 + \Pr[X_2 = 1] \cdot 1 = \Pr[X_2 = 1] \leq p$$

The variances depend on the distance between p and $\frac{1}{2}$ and whether they are a lower or an upper bound. We will only need two combinations out of these four. Let X_1 and X_2 be Bernoulli distributions with bounded probability and let $p_1, p_2 \in [0, 1]$ such that $p_1 \geq \frac{1}{2}$ and $p_2 \leq \frac{1}{2}$ be the lower (resp. upper) bound for the success probability of X_1 (resp. X_2). Then

$$\mathrm{Var}(X_1) \leq p_1(1 - p_1)$$
$$\textit{and}$$
$$\mathrm{Var}(X_2) \leq p_2(1 - p_2)$$

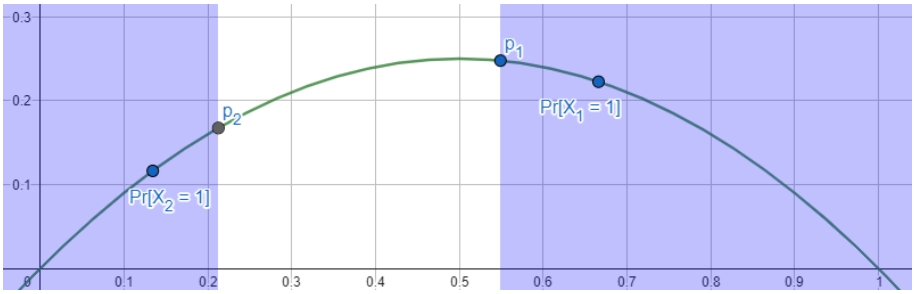

Figure 2.2: Sketch of the bounds for X_1 and X_2

2.2.2 Chebyshev's Inequality

Chebyshev's inequality will be used in this work to proof certain propositions. A first step to understand its use is Markov's inequality, which is typically used in cases in which one knows very little about the distribution of the random variable.

Proposition 2.4 (***Markov's Inequality***)**.** *Let X be a non-negative discrete random variable and v a real number. Then*

$$\Pr[X \geq v] \leq \frac{\mathbb{E}(X)}{v} \tag{2.1}$$

Proof.

$$\begin{aligned}\mathbb{E}(X) &= \sum_x \Pr[X = x] \cdot x \\ &\geq \sum_{x<v} \Pr[X = x] \cdot 0 + \sum_{x \geq v} \Pr[X = x] \cdot v \\ &= \Pr[X \geq v] \cdot v\end{aligned}$$

□

If one has a good upper bound of a random variable's variance, a "possibly stronger" bound can be derived from Markov's inequality.
Note. It will be useful to remember the definition of variance:

$$\mathrm{Var}(X) \stackrel{\text{def}}{=} \mathbb{E}\left[(X - \mathbb{E}(X))^2\right] \tag{2.2}$$

Proposition 2.5 (***Chebyshev's Inequality***)**.** *Let X be a random variable, and $\delta > 0$. Then*

$$\Pr[|X - \mathbb{E}(X)| \geq \delta] \leq \frac{\mathrm{Var}(X)}{\delta^2} \tag{2.3}$$

Proof. We define a random variable $Y := (X - \mathbb{E}(X))^2$ and apply the Markov inequality:

$$\begin{aligned}\Pr[|X - \mathbb{E}(X)| \geq \delta] &= \Pr\left[(X - \mathbb{E}(X))^2 \geq \delta^2\right] \\ &\stackrel{2.1}{\leq} \frac{\mathbb{E}\left[X - \mathbb{E}(X))^2\right]}{\delta^2} \stackrel{2.2}{=} \frac{\mathrm{Var}(X)}{\delta^2}\end{aligned}$$

□

Chebyshev's inequality is particularly useful to analyze the error probability of approximation via repeated sampling.

2.2.3 Chernoff's Bound

Chernoff's bound can be found in many different shapes, depending on the purpose of its use. We will use it in the context of *independent* random variables with the same expected value.
Note. The random variables $X_1, X_2, \ldots, X_n$ are (totally) independent if

$$\Pr[X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n] = \Pr[X_1 = x_1] \Pr[X_2 = x_2] \cdots \Pr[X_n = x_n]$$

Proposition 2.6 (***Chernoff's Bound***). *Let $q \leq \frac{1}{2}$, and let $X_1, X_2, \ldots, X_n$ be independent random variables taking values 0 or 1 almost surely, so that $\Pr[X_i = 1] = q$ for each i. Then for all $\varepsilon, 0 < \varepsilon \leq q(1-q)$, we have*

$$\Pr\left[\left|\frac{\sum_{i=1}^{n} X_i}{n} - q\right| \geq \varepsilon\right] < 2 \cdot e^{-\frac{\varepsilon^2}{2q(1-q)} \cdot n} \tag{2.4}$$

2.2.4 Ensembles

An important concept that arises in computational complexity is the property of two sequences (e.g. sequences of strings or sequences of random variables) to be indistinguishable by any efficient algorithm. This discussion concerns two infinite sequences of distributions (rather than two fixed distributions) and these are called **probability ensembles**.

Definition 2.7 (**Probability Ensemble**). *Let I be a countable index set. An **ensemble indexed by** I is a sequence of random variables indexed by I. Namely, any $X = \{X_i\}_{i \in I}$, where each X_i is a random variable, is an ensemble indexed by I.*

We will normally use a subset of $\{0,1\}^*$ where $\{0,1\}^*$ is the set of all finite strings (see Definition 3.3). An ensemble of the form $X = \{X_w\}_{w \in \{0,1\}^*}$ will have each X_w ranging over strings of length $p(|w|)$ (see definition 3.5), where $p(\cdot)$ is some positive polynomial.

Chapter 3

Interaction

In this chapter, we will focus on the concept of interaction and its (surprisingly) powerful utility. Firstly, we will set the foundations of what (or rather *who*) exactly *interacts* with what and we will formalize it. Secondly, we will introduce some cases in which we can have interaction and why does it make a difference whether there is interaction or not. Finally, we will see the main topic of this chapter with *interactive proof systems* and the well-known (non-trivial) example of *Graph Non-Isomorphism*.

3.1 The Computational Model

As mentioned in previous sections, the modern definition of cryptography is based on computational complexity. In this section, we will go through the definitions of language and some complexity classes and we will establish some assumptions taken throughout this work.

3.1.1 Decision Problems

A **computational problem** is just a question that can be solved by an **algorithm**. A **decision problem** [Koz97] is a "yes" or "no" question. More specifically,

Definition 3.1. *A **decision problem** is a function with a one-bit output: "yes" or "no". To specify a decision problem, one must specify*

- *the set A of possible inputs, and*
- *the subset $B \subseteq A$ of "yes" instances.*

For example, to decide if a given graph is connected, the set of possible inputs is the set of all graphs and the "yes" instances are the connected graphs.

We will always take the set of possible inputs to a decision problem to be the set of finite-length strings over some fixed alphabet (definitions below). Other types of data (e.g. graphs, the set of natural numbers $\mathbb{N}$, trees, etc.) can be encoded naturally as strings. This abstraction allows us to deal with a single type of data and a few basic operations.

Definition 3.2. *An **alphabet** is any finite set Σ. We call elements in Σ **letters** or **symbols** and denote them by a, b, c, ...*

Definition 3.3. *A **string** over Σ is any finite-length sequence of letters. We denote them by x, y, z, ...*

Definition 3.4. *A **language** L is a set of strings . A **statement** is a tuple (x, L) s.t. $x \in L$.*

Definition 3.5. *The **length** of a string x is a function $|\cdot| : L \to \mathbb{N}$ returning the number of letters in x. We denote it by $|x|$.*

The alphabet we will be using is $\Sigma = \{0,1\}$ (i.e. bits) and $L \subset \{0,1\}^*$ where $\{0,1\}^*$ is the set of all finite strings. An example of a string $x \in L$ could be $x = 01010$ and its length is $|x| = 5$.

At the beginning of this section, we already established the need of an algorithm to solve a decision problem. The "tools" we are going to use to *compute* these algorithms and find out whether a string x is a solution (i.e. $x \in L$) or not (i.e. $x \notin L$) are **Turing machines**.

Turing machines are a model of computation believed to compute any algorithm (we will use the concept of Turing machine as a synonym of algorithm). The formal definition won't be used much throughout this work, but it is good to have a rigorous definition [Gas14] of it, since it establishes our theoretical base.

Definition 3.6 (*formal*). *A **(deterministic) Turing machine** M is a tuple (Q, Σ, δ, s) where*

- *Q is a finite set of states. It has the states s, q_{acc}, q_{rej}.*
- *Σ is a finite alphabet. It contains the symbol #.*
- *$\delta : Q \setminus \{q_{acc}, q_{rej}\} \times \Sigma \to \Sigma \times \{L, R\} \times Q$ is a transition function determining the next move.*
- *$s \in Q$ is the start state, q_{acc} is the accept state, q_{rej} is the reject state.*

We use the following convention:

1. *On input $x \in \Sigma^*, x = x_1 \cdots x_n$, the machine starts with tape*

$$\#x_1 x_2 \cdots x_n \#\#\#\# \cdots$$

   that is one way infinite.

2. *The head is initially looking at the x_n.*

3. *If $\delta(q, \sigma) = (\tau, L, p)$ then the symbol σ is overwritten with τ, the head moves to the Left and the state changes from q to p.*

4. *If $\delta(q, \sigma) = (\tau, R, p)$ then the symbol σ is overwritten with τ, the head moves to the Right and the state changes from q to p.*

5. *If the machine halts in state q_{acc} then we say M ACCEPTS x. If the machine halts in state q_{rej} then we say M REJECTS x.*

We will denote the output of M on input x as

$$M(x) = 1 \iff M\,ACCEPTS\,x$$
$$M(x) = 0 \iff M\,REJECTS\,x$$

From the previous definition, we will mainly use the notion of *deciding* (i.e. accepting or rejecting a string x). In this sense, we can talk about the *decidability* of a language:

Definition 3.7. *A language L is **decidable** if there is a Turing Machine M such that*

$$x \in L \to M(x) = 1$$
$$x \notin L \to M(x) = 0$$

3.1.2 Complexity Classes $\mathcal{P}$ and $\mathcal{NP}$

We will now define complexity classes that capture the power of efficient algorithms. As is common in complexity theory, these classes are defined in terms of decision problems, where the set of inputs where the answer should be "yes" is specified by a language $L \subset \{0,1\}^*$.

Each complexity class is defined by the type of algorithm (i.e. Turing machine) that decides over a set of languages.

Definition 3.8. *A **deterministic algorithm** A is an algorithm that, given a particular input, will always produce the same output. We say that A runs in **polynomial time** if there exists a polynomial $t : \mathbb{N} \to \mathbb{N}$ such that, for every input $x \in \{0,1\}^*$, its computation takes at most $t(|x|)$ steps.*

Definition 3.9 (***alternative***)**.** *A **deterministic Turing machine** M is a hypothetical machine capable of simulating any deterministic algorithm. We say that M runs in **polynomial time** if the simulated deterministic algorithm runs in polynomial time.*

Having established the first type of algorithms, we can now define a complexity class:

Definition 3.10 (**Complexity Class** $\mathcal{P}$)**.** *The complexity class $\mathcal{P}$ is the class of languages L for which there exists a deterministic polynomial-time Turing Machine M such that*

- $\forall x \in L \Rightarrow M(x) = 1$,
- $\forall x \notin L \Rightarrow M(x) = 0$.

The complexity class $\mathcal{P}$ is associated with languages that are efficient to recognize. The complexity class $\mathcal{NP}$ is associated with computational problems having solutions that, once given, can be efficiently tested for validity.

Definition 3.11 (**Complexity Class** $\mathcal{NP}$)**.** *The complexity class $\mathcal{NP}$ is the class of languages L for which there exists a Boolean relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$ and a polynomial $p(\cdot)$ such that*

- *R_L can be recognized in (deterministic) polynomial time, and*
- *$x \in L$ iff there exists a y such that $|y| \leq p(|x|)$ and $(x, y) \in R_L$.*

*Such a y is called a **witness for membership** of $x \in L$.*

Then, $\mathcal{NP}$ consists of the set of languages for which there exists short proofs of membership that can be efficiently verified. It is an open problem to determine whether $\mathcal{P} \neq \mathcal{NP}$.

3.1.3 Complexity Class $\mathcal{BPP}$

For the sake of this work, we need to consider algorithms that include a probabilistic element to them. Probabilistic algorithms play a central role in cryptography. They are needed to allow legitimate parties to generate secrets. There are two equivalent ways in which we can view probabilistic Turing Machines (i.e. algorithms).

One way is to allow the Turing Machine to make random moves (i.e. internal "coin tosses"). Looking back at Definition 3.6, we can modify the transition function:

$$\delta : Q - \{q_{acc}, q_{rej}\} \times \Sigma \times \{0,1\} \rightarrow \Sigma \times \{L, R\} \times Q$$

The new transition function maps each pair of $(\langle state\rangle, \langle symbol\rangle, \langle coin\ toss\rangle)$ to two possible triples of the form $(\langle symbol\rangle, \langle direction\rangle, \langle state\rangle)$. At each step, the machine chooses at random (with probability $1/2$ for each possibility) and then acts accordingly. These random choices are known as the *internal coin tosses* of the machine. Under these conditions, a machine M outputs a random variable, denoted by $M(x)$, induced by its internal coin tosses. We denote by $\Pr[M(x) = y]$ as the probability that, on input x, machine M outputs y. Since we consider only polynomial-time machines, we can assume, without loss of generality, that the number of coin tosses made by M on input x is independent of their outcome and is denoted by $t_M(x)$.

Another way of looking at probabilistic Turing Machines is to add an auxiliary input tape with all the coin tosses (i.e. we consider deterministic Turing Machines with two inputs). From this point of view, the coin tosses are "external" (i.e. *external coin tosses*). We denote the auxiliary input as r which is uniformly chosen $r \in \{0,1\}^{t_M(x)}$ and the output of a machine M as $M(x)_r$ (i.e. output of M on input x and auxiliary input r).

We sum up these concepts with the next two definitions:

Definition 3.12. *A **probabilistic algorithm** A is an algorithm which relies on a certain degree of randomness (typically uniformly random "coin tosses"). We say that A runs in **polynomial time** if there exists a polynomial $t : \mathbb{N} \to \mathbb{N}$ such that, for every input $x \in \{0,1\}^*$ and every sequence of coin tosses, its computation takes at most $t(|x|)$ steps and a polynomial amount of random bits.*

Definition 3.13. *A **probabilistic Turing Machine** M is a hypothetical machine capable of simulating any probabilistic algorithm. We say that M runs in **polynomial time** if the simulated probabilistic algorithm runs in polynomial time.*

The complexity class capturing these computations is the class denoted $\mathcal{BPP}$ (definition below). The probability we are talking about in this complexity class refers to the event in which the machine makes the correct verdict on string x.

Definition 3.14 (Complexity Class $\mathcal{BPP}$). *The complexity class $\mathcal{BPP}$ (i.e. Bounded-Probability Polynomial Time) is the class of languages L for which there exists a probabilistic polynomial-time Turing machine M such that*

- $\forall x \in L \Rightarrow \Pr[M(x) = 1] \geq \frac{2}{3}$,
- $\forall x \notin L \Rightarrow \Pr[M(x) = 0] \geq \frac{2}{3}$.

The phrase *bounded probability* indicates that the success probability is bounded away from $1/2$. By replacing the constant $2/3$ by any other constant grater than $1/2$ will not change the class defined. In fact, the constant $2/3$ can be replaced by $1 - 2^{-|x|}$ and the class will remain invariant [Vad12]. Let us proof these last two statements.

Before starting the proofs, it is very important to establish what type of probability distribution fits the description of a probabilistic Turing Machine. As we have seen in the formal Definition 3.6 of Turing Machine, the only possible outputs are 0 or 1. When working with languages in $\mathcal{BPP}$, the probabilities attached to the two possible outputs are bounded by a constant (on the length of the input x). Namely, if $q \in [0, 1]$ then

- $\forall x \in L \Rightarrow \Pr[M(x) = 0] \leq 1 - q$, $\Pr[M(x) = 1] \geq q$,
- $\forall x \notin L \Rightarrow \Pr[M(x) = 0] \geq 1 - q$, $\Pr[M(x) = 1] \leq q$.

This will lead us to work with bounds on the expected value and variance of a probabilistic Turing machine (see Section 2.2.1).

Proposition 3.15 (Equivalent definition of $\mathcal{BPP}$. Part 1). *Let L be a language. $L \in \mathcal{BPP}$ iff there exists a polynomial $p(\cdot)$ and a probabilistic polynomial-time Turing machine M such that*

- $\forall x \in L \Rightarrow \Pr[M(x) = 1] \geq \frac{1}{2} + \frac{1}{p(|x|)}$, and
- $\forall x \notin L \Rightarrow \Pr[M(x) = 0] \geq \frac{1}{2} + \frac{1}{p(|x|)}$.

Proof. $\Longrightarrow$) Suppose $L \in \mathcal{BPP}$. By Definition 3.14, there exists a probabilistic polynomial-time Turing machine M with two-sided error probability at most $\frac{1}{3}$. Let $p = k$ be a positive constant value polynomial. Using the same machine M, we can find a bound for k:

$$\frac{2}{3} \geq \frac{1}{2} + \frac{1}{k} \iff k \geq 6$$

Finally, since $L \in \mathcal{BPP}$, using $p = k \geq 6$ and M, we obtain the desired result.

$\Longleftarrow$) Let M be a probabilistic polynomial-time Turing machine as described above. Let us construct another probabilistic polynomial-time Turing machine M' that, on input x, runs $p(|x|)^2$ independent copies of M (i.e. M_i) and rules by

majority. Namely,

$$M'(x) := \frac{\sum_{i=1}^{p(|x|)^2} M_i(x)}{p(|x|)^2}, \text{ and}$$
$$M' \, ACCEPTS \, x \iff M'(x) > \frac{1}{2}$$

We are going to apply Chebyshev's inequality to show that M' satisfies the proposition.
First, let us find a bound for the expected value $\mathbb{E}(M'(x))$ and the variance $\mathrm{Var}(M'(x))$ of M'. Let $x \in L$, then

$$\mathbb{E}(M'(x)) = \mathbb{E}\left[\frac{\sum_{i=1}^{p(|x|)^2} M_i(x)}{p(|x|)^2}\right] \underset{independent}{\overset{M_i(x)}{=}} \frac{\sum_{i=1}^{p(|x|)^2} \mathbb{E}\left(M_i(x)\right)}{p(|x|)^2} \geq$$
$$= \frac{\cancel{p(|x|)^2}\left(\frac{1}{2} + \frac{1}{p(|x|)}\right)}{\cancel{p(|x|)^2}} = \frac{1}{2} + \frac{1}{p(|x|)} \tag{3.1}$$

$$\mathrm{Var}(M'(x)) = \mathrm{Var}\left[\frac{\sum_{i=1}^{p(|x|)^2} M_i(x)}{p(|x|)^2}\right] \underset{independent}{\overset{M_i(x)}{=}} \frac{\sum_{i=1}^{p(|x|)^2} \mathrm{Var}\left(M_i(x)\right)}{p(|x|)^4} \leq$$
$$= \frac{\cancel{p(|x|)^2}\left(\frac{1}{2} + \frac{1}{p(|x|)}\right)\left(\frac{1}{2} - \frac{1}{p(|x|)}\right)}{p(|x|)^{\cancel{4}}} = \frac{\frac{1}{4} - \frac{1}{p(|x|)^2}}{p(|x|)^2} \tag{3.2}$$

Where the last inequality comes from having the largest possible variance (i.e. when $\Pr[M(x) = 1] = \frac{1}{2} + \frac{1}{p(|x|)}$).

Suppose that $x \in L$ (analogous for $x \notin L$). Then

$$\begin{aligned}
\Pr\left[M'(x) \leq \frac{1}{2}\right] &= \Pr\left[M'(x) + \frac{1}{p(|x|)} \leq \frac{1}{2} + \frac{1}{p(|x|)}\right] \overset{3.1}{\leq} \\
&\leq \Pr\left[M'(x) + \frac{1}{p(|x|)} \leq \mathbb{E}(M'(x))\right] = \\
&= \Pr\left[M'(x) - \mathbb{E}(M'(x)) \leq -\frac{1}{p(|x|)}\right] = \\
&= \Pr\left[-M'(x) + \mathbb{E}(M'(x)) \geq \frac{1}{p(|x|)}\right] \leq \\
&\leq \Pr\left[\left|M'(x) - \mathbb{E}(M'(x))\right| \geq \frac{1}{p(|x|)}\right] = \\
&= \Pr\left[\left|M'(x) - \mathbb{E}(M'(x))\right| \geq \delta\right] \overset{2.3}{\leq} \\
&\leq \frac{\mathrm{Var}\left[M'(x)\right]}{\delta^2} \overset{3.2}{\leq} \frac{\frac{\frac{1}{4} - \frac{1}{p(|x|)^2}}{\cancel{p(|x|)^2}}}{\frac{1}{\cancel{p(|x|)^2}}} = \frac{1}{4} - \frac{1}{p(|x|)^2} \leq \frac{1}{4} \leq \frac{1}{3}
\end{aligned}$$

and finally

$$\Pr\left[M'(x) > \frac{1}{2}\right] = 1 - \Pr\left[M'(x) \leq \frac{1}{2}\right] \geq \frac{2}{3}$$

□

We have seen that the error bound decreases by repetition (with a speed of $O(n^c)$ on input's length n). It is possible to get an exponential decrease on the bound:

Proposition 3.16 (Equivalent definition of $\mathcal{BPP}$. Part 2). *Let L be a language. $L \in \mathcal{BPP}$ iff for every positive polynomial $p(\cdot)$ there exists a probabilistic polynomial-time Turing machine M such that*

- $\forall x \in L \Rightarrow \Pr[M(x) = 1] \geq 1 - 2^{-p(|x|)}$, and
- $\forall x \notin L \Rightarrow \Pr[M(x) = 0] \geq 1 - 2^{-p(|x|)}$.

Proof. $\Longrightarrow$) Suppose $L \in \mathcal{BPP}$. By Definition 3.14, there exists a probabilistic polynomial-time Turing machine M with bounded probability at most $\frac{2}{3}$. Let p be any positive polynomial.
We will use a similar setup as the previous proof. Let us construct another probabilistic polynomial-time Turing machine M' that, on input x, runs $p(|x|)$ copies of

M and rules by majority. Namely,

$$M'(x) := \frac{\sum_{i=1}^{p(|x|)} M_i(x)}{p(|x|)}, \text{ and}$$
$$M' \, ACCEPTS \, x \iff M'(x) > \frac{1}{2}$$

We are going to apply Chernoff's inequality to show that M' satisfies the proposition.

Note. Let $p \in \mathbb{R}[x]$ be any positive real polynomial. Then

$$e^{-p(x)} < 2^{-p(x)} \text{ for a big enough x} \tag{3.3}$$

Suppose that $x \in L$ (analogous for $x \notin L$). Then

$$\begin{aligned}
\Pr\left[M'(x) \leq \frac{1}{2}\right] &= \Pr\left[M'(x) - \frac{1}{3} \leq \frac{1}{6}\right] = \\
&= \Pr\left[-M'(x) + \frac{1}{3} \geq -\frac{1}{6}\right] \leq \\
&\leq \Pr\left[\left|M'(x) - \frac{1}{3}\right| \geq \frac{1}{6}\right] = \\
&= \Pr\left[\left|M'(x) - q\right| \geq \varepsilon\right] \overset{2.4}{\leq} \\
&\leq 2 \cdot e^{-\frac{\varepsilon^2}{2 \cdot q(1-q)} p(|x|)} = \\
&= 2 \cdot e^{-\frac{\left(\frac{1}{6}\right)^2}{2 \cdot \frac{1}{3}\left(1-\frac{1}{3}\right)} p(|x|)} = \\
&= 2 \cdot e^{-\frac{1}{16} p(|x|)} \overset{3.3}{<} 2^{-p(|x|)}
\end{aligned}$$

and finally

$$\Pr\left[M'(x) > \frac{1}{2}\right] = 1 - \Pr\left[M'(x) \leq \frac{1}{2}\right] \geq 1 - 2^{-p(|x|)}$$

$\Longleftarrow$) Let p be a positive polynomial and M a probabilistic polynomial-time Turing machine satisfying the proposition. Since we can pick any polynomial, let us find a bound for it:

$$1 - 2^{-p(x)} \geq \frac{2}{3} \iff \frac{1}{3} \geq 2^{-p(x)} \iff 2^{p(x)} \geq 3 \iff p(x) \geq \frac{\log(3)}{\log(2)} \tag{3.4}$$

Finally, choosing any polynomial $p(x) \geq \frac{\log(3)}{\log(2)}$ we get the desired result (e.g. $p(x) = 2$). Namely,

- $\forall x \in L \Rightarrow \Pr[M(x) = 1] \geq 1 - 2^{-p(|x|)} \overset{3.4}{\geq} \frac{2}{3}$, and
- $\forall x \notin L \Rightarrow \Pr[M(x) = 0] \geq 1 - 2^{-p(|x|)} \overset{3.4}{\geq} \frac{2}{3}$.

□

The main use for these last two propositions is to understand that the probability of making an *incorrect* verdict can be made as small as necessary.

Definition 3.17. *A function $f : \mathbb{N} \to \mathbb{R}$ is **negligible** if for every positive polynomial $p(\cdot)$ there exists an N such that for all $n > N$,*

$$f(n) < \frac{1}{p(n)}$$

In other words, f is negligible if it decreases faster than the reciprocal of any polynomial.

Proposition 3.18. *Let $f : \mathbb{N} \to \mathbb{R}$ be a negligible function and p be a polynomial. The function $f'(n) := p(n) \cdot f(n)$ is negligible.*

Then, negligible functions stay that way when multiplied by any fixed polynomial. We conclude that that languages in $\mathcal{BPP}$ can be recognized by probabilistic polynomial-time Turing machines with a *negligible* error probability. It follows that an event that occurs with negligible probability would be highly unlikely to occur even if we repeated the experiment polynomially many times.

3.1.4 Intractability Assumptions

The differentiation between $\mathcal{NP}$ and $\mathcal{BPP}$ is of great interest, since we consider *intractable* those tasks that cannot be performed by probabilistic polynomial-time machines. On the other hand, the adversarial tasks in which we shall be interested can be performed by non-deterministic polynomial-time machines (because the solutions, once found, can be tested for validity). Thus, the content of this work is only interesting if $\mathcal{NP} \nsubseteq \mathcal{BPP}$ (it is not known whether $\mathcal{BPP} \nsubseteq \mathcal{NP}$).

3.2 Interactive Proof

In this section we introduce interactive proof systems which play an important role in zero-knowledge interactive proofs and we will present the non-trivial example of graph isomorphism. Firstly, we will discuss what do we understand by *proof* and *knowledge* and introduce the concept of interactive Turing machines. Secondly, we will define how these machines interact with each other. Finally, we will explore the aforementioned example.

3.2.1 Proofs

The concept of *proof* can be said to have a static nature in mathematics. Proofs are viewed as fixed objects derived from axioms and commonly-agreed (or even self-evident) deviation rules and are as fundamental as their consequences (i.e. theorems).
In other areas of study, proofs can be said to have a more dynamic nature (e.g. proving your age by showing your ID, evidence in a court of law, etc.). By *dynamic* we mean to say that it requires an **interaction** and this is the approach most appropriate to understand the notion of zero-knowledge proofs.

Having established that we will understand proofs from an interactive point of view, there are two concepts that arise naturally: the *prover* and the *verifier*.
The **prover** is the entity providing the proof, which can sometimes be hidden or implicit. In contrast, the **verifier** is usually more explicit and its role is central in our dynamic definition of proof. The role of the verifier is known as the *verification process* and it is considered to be relatively easy (the burden is placed on the prover to supply the proof). It is important to note that the verifier has a "distrustful attitude" towards the prover (else, no proof would be needed). This is not so different in traditional mathematics, where the prover usually requires a long time while the verifier (i.e. the mathematics community) usually needs less time to asses its validity.

Two fundamental properties of any verification procedure are: *soundness* and *completeness*. **Soundness** captures the ability of the verifier to not accept false statements from any prover. **Completeness** captures the existence of some prover with the ability to convince the verifier of a true statement (belonging to some predefined set of true statements). Though it is known that not every set of true statements has a proof system, in this work we will focus only on those that do have one.

The complexity class $\mathcal{NP}$ captures the aforementioned asymmetry between prover and verifier. Recall from the Definition 3.11 that any language $L \in \mathcal{NP}$ is characterized by a polynomial-time-recognizable boolean relation R_L such that

$$L = \{x : \exists y \text{ s.t. } (x, y) \in R_L\}$$

and $(x, y) \in R_L$ iff $|y| \leq p(|x|)$ for some positive polynomial $p(\cdot)$. Hence, the verification procedure for membership claims of the form "$x \in L$" consists of applying the (polynomial-time) algorithm for recognizing R_L to the claim x and a prospective proof y. Any y satisfying $(x, y) \in R_L$ is considered a *proof* of membership of $x \in L$. Thus, correct statements are the only ones to have proofs in this proof system. Here it is easy to see that the verification procedure is "easy" (i.e. polynomial-time), but coming up with proofs may be "difficult".

3.2.2 Knowledge

The concept of *knowledge* is central to the discussion of zero-knowledge proofs. Though defining what knowledge is would seem the correct procedure, it is a very complex topic. Thus, we will focus on what a *gain in knowledge* is and we will not give a formal definition on the previous question. This approach, though seemingly imprecise, it is sufficient regarding cryptography and, more precisely, zero-knowledge proofs.

As we have seen in this work, the fact that some statement (i.e. (x, L)) is recognizable in polynomial-time (i.e. $x \in L$) is of great importance, since it is efficient to compute. Thus, knowledge is related to **computational difficulty** and we say that we *gain knowledge* if we receive a result of a computation that is infeasible to us and we *gain no knowledge* if we can compute the result ourselves.

To get a better understanding on this discussion, let's see an example. Consider a conversation between `Alice` and `Bob` in which `Bob` asks `Alice` whether or not a graph is Eulerian. We know that a graph is Eulerian iff it is connected and all its vertices have even degrees. We can run a polynomial-time algorithm (actually linear-time) to check such a statement, so `Bob` *has gained no knowledge*, since he could run the algorithm himself. In contrast, if `Bob` asks `Alice` whether or not the graph is Hamiltonian and `Alice` (somehow) answers this question, then we can say that `Bob` has gained some knowledge (since we don't know any polynomial-time algorithm to answer such question). We say that `Bob` *has gained knowledge* from the interaction if his computational ability, concerning the publicly known graph, *has increased*.

3.2.3 Interactive Turing Machines

A central paradigm in zero-knowledge proofs is **interaction** (e.g. between `Alice` and `Bob` in the previous example). We will understand interaction as the exchange of *information* (in a very broad sense) between two (or more) parties on a **common input** (i.e. the statement to be proved). We first introduce the notion of *interactive machine* and then the notion of interaction between two such machines.

Definition 3.19 (**Interactive Turing Machine**). *An **interactive Turing machine (ITM)** is a (deterministic) multi-tape Turing machine. The tapes and the names of their contents are:*

- *a read-only input tape containing the **input**,*
- *a read-only random tape containing the **random input**,*
- *a read-and-write work tape,*
- *a write-only output tape containing the **output**,*
- *a read-only communication tape containing the **message received**,*
- *a write-only communication tape containing the **message sent**, and*
- *a read-and-write switch tape consisting of a single cell containing the current **active machine**.*

*Each ITM is associated with a single bit $\sigma \in \{0,1\}$, called its **identity**. An ITM is said to be **active** if the content of its switch tape equals the machine identity. Otherwise the machine is said to be **idle**. While being idle, the state of the machine, the locations of its heads on the various tapes and the contents of the writable tapes of the ITM are not modified.*
Without loss of generality, the machine movement on both communication tapes are in only one direction (e.g. from left to right).

We will consider the content of the random input to be uniformly and independently chosen from an infinite sequence of internal coin tosses. Hence, interactive Turing machines are **probabilistic**. Though there is an infinite amount, consider that, in a finite computation, only a finite prefix is read.
The distinction between the input tape and the read-only communication tape might seem unnecessary (as well as between the output tape and the write-only communication tape), but we establish such difference because we will not consider a single interactive Turing machine. Instead, we will always consider a pair of interactive Turing machines where some of its tapes match.

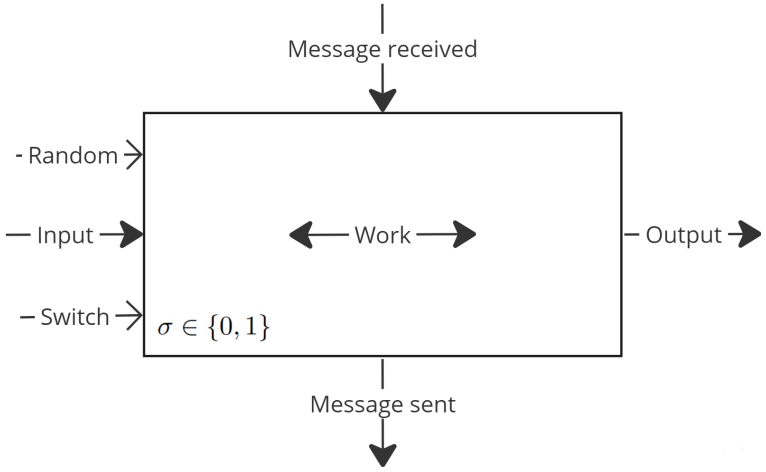

Figure 3.1: Interactive Turing machine schema

Definition 3.20 (Linked ITMs). *Two interactive Turing machines are said to be **linked** if they have opposite identities, their input and switch tapes coincide and the read-only communication tape of one machine coincides with the write-only communication tape of the other machine and vice versa. The other tapes of both machines (i.e. the random tape, the work tape and the output tape) are distinct.*

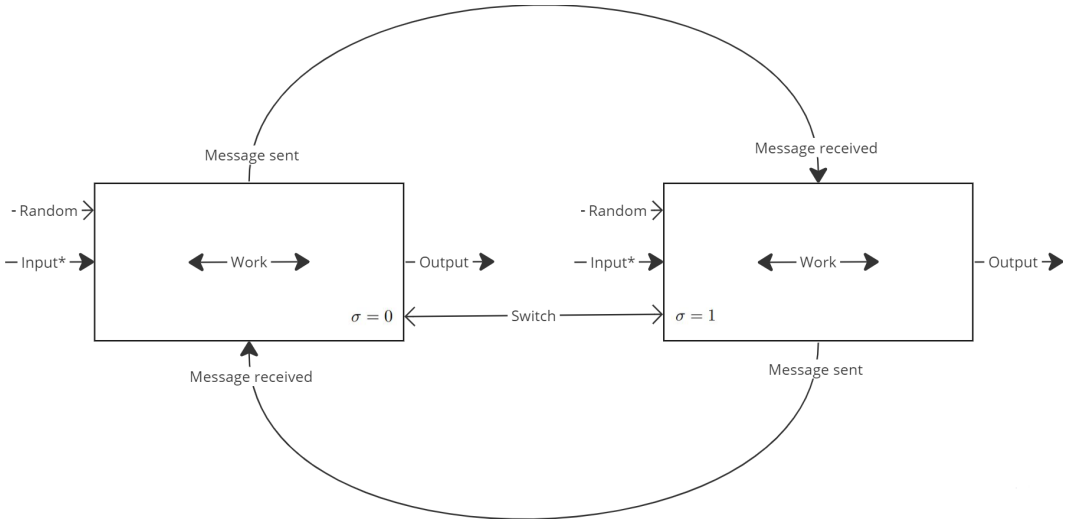

Figure 3.2: Linked interactive Turing machines (*Input** is equal for both machines)

Definition 3.21 (Joint Computation of Two ITMs). *The **joint computation** of a linked pair of ITMs, on a common input x, is a **sequence of pairs** representing the local*

*configurations of both machines. That is, each pairs consists of two strings, each representing the local configuration of one of the machines. In each such pair of local configurations, one machine (not necessarily the same one) is active, while the other machine is idle. The first pair in the sequence consists of initial configurations corresponding to the common input x, with the content of the switch tape set to zero. If one machine halts while the switch tape still holds its identity, then we say that both machines have **halted**. The **outputs** of both machines are determined at that time.*

It is interesting to notice that none of the interactive Turing machines have an individual input tape, even thought they do have an individual random tape. We will consider this added feature further down this work, since it is important for practical purposes to give a local input tape to both machines (e.g. to use some partial piece of information only available to the specific interactive Turing machine). Until then, we will keep this simpler definition, since it is sufficient to demonstrate the power of this concept.

It is natural to consider the time-complexity of an interactive Turing machine (as we did for other Turing machines). The time-complexity of an interactive Turing machine is dependent on the **length of its input**.

Definition 3.22 (Complexity of an ITM). *We say that an interactive Turing machine A has **time-complexity** $t : \mathbb{N} \to \mathbb{N}$ if for every interactive Turing machine B and every string x, it holds that when interacting with machine B, on common input x, machine A always (i.e. regardless of the content of its random tape and B's random tape) halts within $t(|x|)$ steps. In particular, we say that A is **polynomial-time** if there exists a polynomial p such that A has time-complexity p.*

We have seen that time-complexity is an upper bound of the machine and it is independent of the content of the input. In particular, an interactive Turing machine with time-complexity $t(\cdot)$ may read, on input x, only a prefix of total length $t(|x|)$ of the messages sent to it.

3.2.4 Interactive Proof Systems

As mentioned in a previous section 3.2.1, there are two central figures in any proof (from our dynamic definition of proof): the *prover* and the *verifier*. A *proof system* is defined in terms of its *verification procedure* (i.e. the verifier) and proofs are external to this verification procedure (i.e. from the prover). We stress that the verification procedure does not generate proofs, it only checks their validity. *Interactive proof systems* capture the nature of "interaction with the outside" and can consist of several message exchanges (as long as the verification procedure

happens in polynomial time).

We also mentioned that a proof system must fulfill two important properties: *completeness* (i.e. accept true statements) and *soundness* (i.e. not accept false statements). These properties will be now given with a certain error margin, in other words, there is a probabilistic aspect to interactive proof system. Hence, we can redefine *completeness* as accepting true statements with a "high" probability and *soundness* as accepting false statements with "low" probability.
As we saw with probabilistic Turing machines, the output of a pair of interactive Turing machines is a random variable. Let A and B be a pair of interactive Turing machines and let x be their common input. We denote $\langle A, B\rangle(x)$ the random variable representing the output of B after interacting with machine A on input x, when the random input on each machine is uniformly and independently chosen.

Definition 3.23 (**Interactive Proof System**). *A pair of ITMs (P, V) is called an **interactive proof system for a language** L if machine V is polynomial-time and the following two conditions hold:*

- ***Completeness**: $\forall x \in L$,*

$$\Pr\left[\langle P, V\rangle(x) = 1\right] \geq \frac{2}{3}$$

- ***Soundness**: $\forall x \notin L$ and every interactive Turing machine B,*

$$\Pr\left[\langle B, V\rangle(x) = 1\right] \leq \frac{1}{3}$$

From the definition, we can observe the probabilistic nature of interactive proof systems in both the completeness and the soundness conditions. It is important to understand some details in the definition. Firstly, the verifier V must be a probabilistic polynomial-time Turing machine, while the prover P has no resource bounds on its computing power. Secondly, the completeness property refers only to the interaction between the prescribed prover P and the verifier V, while the soundness condition refers to all potential machines interacting with V (i.e. all potential provers). Finally, the error probability is very similar to that of the complexity class $\mathcal{BPP}$ ans can be made exponentially small by repetition.

We can now define a new complexity class containing all languages with an interactive proof system.

Definition 3.24 (**Complexity Class $\mathcal{IP}$**). *The class $\mathcal{IP}$ consists of all languages having interactive proof systems.*

We will now see an interesting relationship between the class $\mathcal{IP}$ and the classes $\mathcal{NP}$ and $\mathcal{BPP}$. This shows the power of interactive proof systems and the potential for zero-knowledge proofs.

Proposition 3.25. *Let $L \in \mathcal{NP}$. Then, there exists an interactive proof system for the language L.*

Proof. Let $L \in \mathcal{NP}$. By Definition 3.11, there exists a polynomial $p(\cdot)$ and a boolean relation R_L recognizable in polynomial time, such that

$$L = \{x : \exists y \text{ s.t. } |y| \leq p(|x|) \text{ and } (x, y) \in R_L\}$$

Lets consider the pair of interactive Turing machines (P, V) such that, on common input x, P sends a witness y and V checks the conditions $|y| \leq p(|x|)$ and $(x, y) \in R_L$. Clearly, V is polynomial-time (since both conditions are computable in polynomial time by definition) and

- $\forall x \in L$,
$$\Pr\left[\langle P, V\rangle(x) = 1\right] = 1 \geq \frac{2}{3}$$
- $\forall x \notin L$, and every interactive Turing machine B,
$$\Pr\left[\langle B, V\rangle(x) = 1\right] = 0 \leq \frac{1}{3}$$

□

This is a specific type of proof system, where the interaction is unidirectional (from prover to verifier) and both machines are deterministic (i.e. no probabilistic aspect). Thus, the verifier will always accept an input in the language (i.e. output 1) and reject any other input (i.e. output 0).

Corollary 3.26. $\mathcal{BPP} \cup \mathcal{NP} \subseteq \mathcal{IP}$

Proof. It follows from Proposition 3.25 that $\mathcal{NP} \subseteq \mathcal{IP}$. Let L be a language such that $L \in \mathcal{BPP}$. Languages in $\mathcal{BPP}$ can be viewed as each having a verifier that decides on membership without interaction. Namely, let V be an interactive Turing machine with a configuration following that of the existing probabilistic polynomial-time Turing machine from Definition 3.14. Let P be any prover. Then,

- $\forall x \in L$,
$$\Pr\left[\langle P, V\rangle(x) = 1\right] = \Pr\left[V(x) = 1\right] \geq \frac{2}{3}$$

- $\forall x \notin L$, and every interactive Turing machine B,

$$\Pr\left[\langle B, V\rangle(x) = 1\right] = \Pr\left[V(x) = 1\right] \leq \frac{1}{3}$$

It follows that $\mathcal{BPP} \subseteq \mathcal{IP}$. □

It is an open problem whether or not $\mathcal{BPP} \subseteq \mathcal{NP}$.

As we saw with the complexity class $\mathcal{BPP}$, we can change the constant bounds in Definition 3.23 for other functions such that the class $\mathcal{IP}$ remains invariant. These functions can be made *negligible* (see 3.17) by repetition so that the error probability is as small as needed.

Definition 3.27 (Generalized Interactive Proof System). *Let $c, s : \mathbb{N} \to \mathbb{R}$ be functions satisfying $c(n) > s(n) + \frac{1}{p(n)}$ for some polynomial $p(\cdot)$. An interactive pair (P, V) is called a **(generalized) interactive proof system for the language** L, with **completeness bound** $c(\cdot)$ and **soundness bound** $s(\cdot)$, if*

- *(modified) **Completeness**: $\forall x \in L$,*

$$\Pr\left[\langle P, V\rangle(x) = 1\right] \geq c(|x|)$$

- *(modified) **Soundness**: $\forall x \notin L$ and every interactive Turing machine B,*

$$\Pr\left[\langle B, V\rangle(x) = 1\right] \leq s(|x|)$$

*The function $g(\cdot)$ defined as $g(n) \stackrel{\text{def}}{=} c(n) - s(n)$ is called the **acceptance gap** of (P, V) and the function $e(\cdot)$, defined as $e(n) \stackrel{\text{def}}{=} \max\{1 - c(n), s(n)\}$, is called the **error probability** of (P, V). In particular, s is the **soundness error** of (P, V) and $1 - c$ is its **completeness error**.*

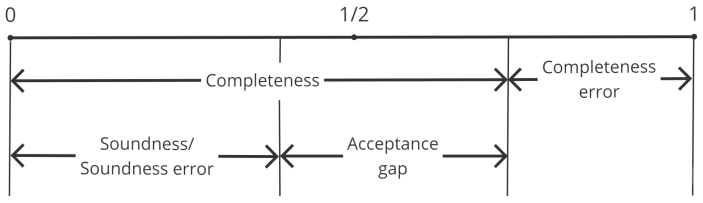

Figure 3.3: Bounds for a Generalized Interactive Proof System

Proposition 3.28. *The following three conditions are equivalent:*

1. $L \in \mathcal{IP}$ *(As in Definition 3.23).*

2. L has a *very strong* interactive proof system: *For every positive polynomial $p(\cdot)$, there exists an interactive proof system for the language L, with error probability bounded above by $2^{-p(\cdot)}$ (i.e. Definition 3.27 with two additional conditions: $c(n) < 1 - 2^{-p(n)}$ and $s(n) > 2^{-p(n)}$).*

3. *There exists a positive polynomial $p(\cdot)$ and a generalized interactive proof system for a language L, with acceptance gap bounded below by $1/p(\cdot)$. Furthermore, completeness and soundness bounds for this system (i.e. $c(n)$ and $s(n)$) can be computed in time polynomial in n.*

Proof. We will only proof (1) $\iff$ (3), since (2) $\iff$ (1) can be easily derived from it and the proof of Proposition 3.16.
3 $\implies$ 1) Let (P', V') be a linked pair of ITMs that, on input x, repeat $p(|x|)^2$ times the interactive proof (P, V) and rule by number of accepting executions up to $s(|x|)$. Namely,

$$\langle P', V' \rangle(x) := \frac{\sum_{i=1}^{p^2(|x|)} \langle P_i, V_i \rangle(x)}{p^2(|x|)}, \text{ and}$$
$$(P', V') \; ACCEPTS \; x \iff \langle P', V' \rangle(x) > s(|x|)$$

We are going to apply Chebyshev's inequality to show that (P', V') satisfies the completeness condition.
First, let us find a bound for the expected value $\mathbb{E}(\langle P', V' \rangle(x))$ and the variance $\mathrm{Var}(\langle P', V' \rangle(x))$. Let $x \in L$, then:

$$\mathbb{E}(\langle P', V' \rangle(x)) = \mathbb{E}\left[\frac{\sum_{i=1}^{p(|x|)^2} \langle P_i, V_i \rangle(x)}{p(|x|)^2} \right] \underset{independent}{\overset{\langle P_i, V_i \rangle(x)}{=}} \frac{\sum_{i=1}^{p(|x|)^2} \mathbb{E}\left(\langle P_i, V_i \rangle(x)\right)}{p(|x|)^2} \geq$$
$$= \frac{\cancel{p(|x|)^2} c(|x|)}{\cancel{p(|x|)^2}} = c(|x|) \tag{3.5}$$

$$\mathrm{Var}(\langle P', V' \rangle(x)) = \mathrm{Var}\left[\frac{\sum_{i=1}^{p(|x|)^2} \langle P_i, V_i \rangle(x)}{p(|x|)^2} \right] \underset{independent}{\overset{\langle P_i, V_i \rangle(x)}{=}} \frac{\sum_{i=1}^{p(|x|)^2} \mathrm{Var}(\langle P_i, V_i \rangle(x))}{p(|x|)^4} \leq$$
$$= \frac{\cancel{p(|x|)^2} c(|x|)(1 - c(|x|))}{p(|x|)^{\cancel{4}}} = \frac{c(|x|)(1 - c(|x|))}{p(|x|)^2} \tag{3.6}$$

Where the last inequality comes from having the largest possible variance (i.e. when $\Pr[\langle P, V\rangle(x) = 1] = c(|x|)$).
Suppose that $x \in L$. Let us check the completeness bound:

$$\begin{aligned}
\Pr\left[\langle P', V'\rangle(x) \leq s(|x|)\right] &= \Pr\left[\langle P', V'\rangle(x) + \frac{1}{p(|x|)} \leq s(|x|) + \frac{1}{p(|x|)}\right] \leq \\
&\leq \Pr\left[\langle P', V'\rangle(x) + \frac{1}{p(|x|)} \leq c(|x|)\right] = \\
&= \Pr\left[\langle P', V'\rangle(x) - c(|x|) \leq -\frac{1}{p(|x|)}\right] = \\
&= \Pr\left[-\langle P', V'\rangle(x) + c(|x|) \geq \frac{1}{p(|x|)}\right] \leq \\
&\leq \Pr\left[\left|\langle P', V'\rangle(x) - c(|x|)\right| \geq \frac{1}{p(|x|)}\right] = \\
&= \Pr\left[\left|\langle P', V'\rangle(x) - \mathbb{E}(\langle P', V'\rangle(x))\right| \geq \delta\right] \overset{2.3}{\leq} \\
&\leq \frac{\mathrm{Var}\left[\langle P', V'\rangle(x)\right]}{\delta^2} \overset{3.6}{\leq} \frac{\frac{c(|x|)(1-c(|x|)}{\cancel{p(|x|)^2}}}{\frac{1}{\cancel{p(|x|)^2}}} = \\
&= c(|x|)(1 - c(|x|) \leq \frac{1}{4} \leq \frac{1}{3}
\end{aligned}$$

and finally

$$\Pr\left[\langle P', V'\rangle(x) > s(|x|)\right] = 1 - \Pr\left[\langle P', V'\rangle(x) \leq s(|x|)\right] \geq \frac{2}{3}$$

Suppose that $x \notin L$ and let B be a interactive Turing machine. The soundness bound is found in an analogous way to the completeness bound (changing the rule bound appropriately), though we must consider a possible cheating B (i.e. B doesn't use independent coin tosses at every repetition). Though this could seem as an issue, the verifier V' will only take independent internal coin tosses and won't be affected by this possibly cheating prover B [AB09].

$1 \Longrightarrow 3$) Let $s = \frac{1}{3}$, $c = \frac{2}{3}$ and $p = 4$. Then,

$$c > s + \frac{1}{p} \qquad \textit{and} \qquad \Pr[\langle P, V\rangle(x) = 1] \geq \frac{2}{3}$$

□

Thus, any language in $\mathcal{IP}$ has a (generalized) interactive proof system with negligible error probability. The proof to this proposition is similar to the proofs

for Propositions 3.15 and 3.16 using repetitive executions. It is possible to prove it by considering *sequential* repetitions or *parallel* repetitions, but the former is easier (see [Gol06]).

3.2.5 Graph Non-Isomorphism

> `Alice` claims that cola stored in a glass bottle tastes better than cola stored in a can. `Bob` doesn't believe her, so they decide to test `Alice`'s claim.
> `Bob` selects randomly a bottle or a can of cola and pours it into a glass without `Alice` seeing it. `Alice` drinks from the glass and tells `Bob` whether the cola was stored in a can or in a bottle. If `Alice` is right, `Bob` is convinced of her claim with a 50% probability. If they repeat the process, `Bob` will be further convinced or `Alice` will be proven to be mistaken.

This story introduces the idea behind an interactive proof system for the language known as **Graph Non-Isomorphism** (or *GNI*) [Rub17], which is the set of *pairs of non-isomorphic graphs*. This language is *not known to be in* $\mathcal{BPP} \cup \mathcal{NP}$.

Let G and H be two graphs with n nodes and (P, V) an ITM described below. On input G and H, the interactive proof system (P, V) for *GNI* will *ACCEPT* if $G \ncong H$ (i.e. G and H are not isomorphic) and will *REJECT* if $G \cong H$ (i.e. G and H are isomorphic):

1. V uses its private random coins to compute a graph G' that is isomorphic to G and a graph H' that is isomorphic to H.
2. V then flips a coin to decide whether to send (G, G') or (G, H') to the prover.
3. P returns a bit indicating the result of coin flipped by V in step 2.

Pair sent to P	Correct response if $G \ncong H$	P response	V output
(G, G')	$\cong$	$\cong$	Continue
(G, G')	$\cong$	$\ncong$	Reject and Stop
(G, H')	$\ncong$	$\cong$	Continue
(G, H')	$\ncong$	$\ncong$	Reject and Stop

Table 3.1: Table of possible response and outputs.

Proposition 3.29. $GNI \in \mathcal{IP}$.

Proof. If G is not isomorphic to H, then H' is not isomorphic to G, so the prover will always be able to determine whether it was sent (G, G') or (G, H') by simply testing if the second graph is isomorphic to G (since the prover claims to have a superior computational power). Therefore, if the two graphs are not isomorphic and the prover and the verifier both follow the protocol, the prover can always determine the result of the coin flip. Namely,

- $\forall x = \{G \ncong H\} \in GNI$,

$$\Pr\left[\langle P, V\rangle(x) = 1\right] = 1$$

However, if G and H are isomorphic, then G' and H' are statistically indistinguishable, since they are equivalent up to isomorphism. Therefore, the prover cannot distinguish between (G, G') and (G, H') better than random guessing. More formally, let q be the fraction of random permutations π such that the prover outputs that $(G, \pi(G))$ are not isomorphic. For a given round of the protocol, we have the probability that the prover fails to pass the challenge is:

$$\Pr\left[P \text{ fails the round}\right] = \frac{1}{2}q + \frac{1}{2}(1 - q) = \frac{1}{2}$$

This probability is over the graphs produced by the verifier, since both G' and H' are permutations of G. Since the prover cannot do better than random guessing when $G \cong H$, repeating the loop above twice will result in the following probabilities:

- $\forall x \notin GNI$, and every interactive Turing machine B,

$$\Pr\left[\langle B, V\rangle(x) = 1\right] \leq \frac{1}{4}$$

Therefore, the completeness and soundness conditions are fulfilled. Clearly, the verifier V is polynomial-time, since it only computes a randomized permutation of a graph and checks whether the answer of P is correct or not. □

Chapter 4

Zero-Knowledge Proof

4.1 Zero-Knowledge Proof Systems

In this section, we will finally introduce the concept of *zero-knowledge proof* (also known as *ZKP*). Firstly, we will explore some examples and define different types of zero-knowledge proofs. Secondly, we will focus on the well-known protocol for *Graph Isomorphism*. Finally, we will state a central result for zero-knowledge proofs and explore further related concepts.

4.1.1 Motivation

The term *zero-knowledge* refers to the concept of proving a statement without giving away "anything more" than the validity of the statement. As we have discussed previously, we say we *gain knowledge* if, after an interaction, we are able to **compute** something we weren't able to compute before.

Imagine you are driving your car and an officer stops you and asks whether or not you have a driving permit. The most common thing to do is give your physical driving license to the officer for him or her to check. In doing so, you (i.e. the *prover*) has given a proof (i.e. the driving license) to the officer (i.e. the *verifier*) and the officer can now easily accept or decline your proof. But there is a "problem" in this interaction: the officer can get your personal information from the driving license (when all he or she wanted to do was answer the question *Do you have a driving permit?*). This is a clear example of an interaction that is not *zero-knowledge*. Another example would be answering the question *Are these two graphs isomorphic?* by giving away an explicit isomorphism between them (see section 4.1.5).

This motivates the research on zero-knowledge proof systems, where the interaction yields *nothing* beyond the validity of the assertion (though we normally refer to computational power, this is still a good depiction of the interest of zero-knowledge).

Let us explore the well-known (and fairly simple) example of *Quadratic Residuosity Protocol*. It is a direct result of the Chinese remainder theorem that, knowing the factorization of a natural number N, there is an efficient algorithm to check if a given number is a quadratic residue and, if so, to find a square root [Tre15]. However, it is believed to be hard to find square roots and to check residuosity modulo N if the factorization of N is not known.

Let $N, r \in \mathbb{N}$ be two integers such that $N > r$. We want to answer the question *Is there an* $x \in \mathbb{N}$ *such that* $x^2 \equiv r \mod N$? A possible *prover* may claim to know the value of x but doesn't want to give it away. In order for a *verifier* to check whether or not the prover knows the value of x, they can both interact in the subsequent form:

- Common input: $N, r \in \mathbb{N}$ such that $N > r$ and N be the product of two unknown odd primes.
- The prover picks a random $y \in \mathbb{N}$ such that $N > y$ and sends $a := y^2 \mod N$ to the verifier.
- The verifier picks a uniformly distributed random bit $b \in \{0, 1\}$ and sends it to the prover.
- The prover sends back

$$\begin{aligned} c &:= y && \text{if } b = 0 \\ c &:= y \cdot x \mod N && \text{if } b = 1 \end{aligned}$$

- The verifier outputs 1 if

$$\begin{aligned} c^2 &\equiv a \mod N && \text{if } b = 0 \\ c^2 &\equiv a \cdot r \mod N && \text{if } b = 1 \end{aligned}$$

  and outputs 0 otherwise.

It is easy to see that, if the prover does know the value of x, then the output of the interaction will always be 1 (verifier *ACCEPTS* with probability 1). On the other hand, if the prover doesn't know the value of x, the verifier REJECTS with

probability $\frac{1}{2}$. Indeed, suppose r is not a quadratic residue. If the verifier chooses $b = 0$ and the prover guesses it, then the verifier would *ACCEPT* a false claim with probability $\frac{1}{2}$. Also, it is not possible that both a and $a \cdot r$ are quadratic residues. If $a \equiv y^2 \mod N$ and $a \cdot r \equiv w^2 \mod N$, then $r \equiv w^2(y^{-1})^2 \mod N$, meaning that r is also a perfect square. Thus, the verifier *REJECTS* with probability $\frac{1}{2}$.

4.1.2 Simulation in ZKPs

Zero-knowledge is a property of the prescribed prover P in an interactive proof system (P, V) for a language L. It captures P's robustness against attempts (not just from the prescribed verifier V) to gain knowledge by interacting with it. Thus, we can say that whatever can be efficiently computed after interacting with P on input $x \in L$ can also be efficiently computed from x.

A central concept related to zero-knowledge proofs is the idea of **simulation**. We have talked informally about what does it mean to be zero-knowledge, establishing that, if an interactive proof system is zero-knowledge, then there is no real gain in computational power. Thus, we will always need some kind of algorithm (i.e. Turing machine) to show that the interaction in the process can be *simulated* without actually going through it.

The *simulation paradigm* postulates that whatever a party can (efficiently) do by itself cannot be considered a gain from interaction with the outside. Thus, no "real gain" can occur whenever we are able to present a simulation (the opposite is not necessarily true: failure to provide a simulation of an interaction with the outside does not necessarily mean that this interaction results in some "real gain"). A trivial example of languages with zero-knowledge interactive proof systems are all languages in $\mathcal{BPP}$ (see Definition 3.14) in which the prover does nothing and the verifier checks by itself whether to accept or reject the common input. In terms of simulation, one can present for every probabilistic polynomial-time machine V^* (i.e. for every verifier V^*) a simulator M^* that is essentially identical to V^*.

4.1.3 Perfect ZKP

We will start with a natural and easy to understand definition (although more "strict") of zero-knowledge interactive proof systems. In the next section, we will "relax" this definition since it is not necessary in a practical sense.

Definition 4.1 (Perfect Zero-Knowledge). *Let (P, V) be an interactive proof system for some language L. We say that (P, V) is **perfect zero-knowledge** if for every proba-*

bilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* such that $\forall x \in L$ the following two conditions hold:

1. *With probability at most $\frac{1}{2}$, on input x, machine M^* outputs a special symbol denoted $\perp$ (i.e. $\Pr[M^*(x) = \perp] \leq \frac{1}{2}$)*
2. *Let $m^*(x)$ be a random variable describing the distribution of $M^*(x)$ conditioned on $M^*(x) \neq \perp$. Namely, $\forall \alpha \in \{0,1\}^*$, $\Pr[m^*(x) = \alpha] = \Pr[M^*(x) = \alpha | M^*(x) \neq \perp]$. Then, the following random variables are identically distributed (see Definition 2.1):*
   - *$\langle P, V^* \rangle(x)$ (i.e. the output of the interactive machine V^* after interacting with the interactive machine P on common input x)*
   - *$m^*(x)$ (i.e. the output of machine M^* on input x, conditioned on not being $\perp$)*

*Machine M^* is called a **perfect simulator** for the interaction of V^* with P.*

The last definition is a formalization of the previous discussions on the concept of *yielding nothing*. The statistical difference between the random variables $\langle P, V^* \rangle(x)$ and $M^*(x)$ can be made negligible (in $|x|$) by changing the bound on the probability that machine M^* will output $\perp$ (on input x) from $\frac{1}{2}$ to $2^{-p(|x|)}$ (similarly to Proposition 3.16), for any positive polynomial $p(\cdot)$. Hence, whatever the verifier efficiently computes after interacting with the prover can be efficiently computed (with an extremely small error) by the simulator (and hence by the verifier himself).

4.1.4 Computational ZKP

The definition of *perfect zero-knowledge* is too strict (it is not necessary to "perfectly simulate" the output of V^* after it interacts with P). Thus, we introduce the concept of *computational indistinguishability* which yields the same results from a computational point of view.

Definition 4.2 (Polynomial-Time Indistinguishability). *Let $S \subset \{0,1\}^*$ be a set of strings and $X \stackrel{def}{=} \{X_w\}_{w \in S}$ and $Y \stackrel{def}{=} \{Y_w\}_{w \in S}$ be two ensembles (see Definition 2.7). We say that X and Y are **indistinguishable in polynomial time** if for every probabilistic polynomial-time algorithm (i.e. Turing machine) D, every positive polynomial $p(\cdot)$ and all sufficiently long $w \in S$,*

$$|\Pr[D(X_w, w) = 1] - \Pr[D(Y_w, w) = 1]| < \frac{1}{p(|w|)}$$

*We often say **computational indistinguishability** instead of indistinguishability in polynomial time.*

This last definition reminds of the concept of *negligible function* (we are actually saying that the function $|\Pr[D(X_n, n) = 1] - \Pr[D(Y_n, n) = 1]|$ with $n \in \mathbb{N}$ is negligible as in Definition 3.17). Now we can modify the definition of perfect *ZKP* to avoid the conditional probability of machine M^* outputting $\perp$.

Definition 4.3 (Computational Zero-Knowledge). *Let (P, V) be an interactive proof system for some language L. We say that (P, V) is **computational zero-knowledge** (or just **zero-knowledge**) if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* such that $\forall x \in L$ the following two ensembles are computationally indistinguishable:*

- $\{\langle P, V^* \rangle(x)\}_{x \in L}$
- $\{M^*(x)\}_{x \in L}$

*Machine M^* is called a **simulator** for the interaction of V^* with P.*

In other words, we will consider an interactive proof system (P, V) to be (computational) zero-knowledge if we can simulate the interaction between any verifier V^* and the prescribed prover P on common input x (i.e. if there exists a simulator M^*). Thus, the interaction gives *no knowledge*. We can also talk about a single interactive Turing machine A being zero-knowledge on a language L if whatever can be efficiently computed after interacting with A on input $x \in L$ can be computed from x itself.

We have formulated zero-knowledge from the point of view of the output of $\langle P, V^* \rangle(x)$, but we could also consider the viewpoint of verifier V^*. By "viewpoint of verifier V^*" we mean the entire sequence of the local configurations of the verifier during an interaction (execution) with the prover. In other words, we just need to consider the internal coin tosses of machine V^* and the messages received from the prover P, since we can compute the same output from these. This leads to an alternative formulation of (computational) zero-knowledge which will be very useful in the upcoming sections.

Definition 4.4 (Computational Zero-Knowledge: Alternative Formulation). *Let (P, V), L and V^* be as defined in Definition 4.3. We denote by $view^P_{V^*}(x)$ a random variable describing the content of the random tape of V^* and the messages V^* receives from P during a joint computation on common input x. We say that (P, V) is **(computational) zero-knowledge** if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* such that the ensembles $\{view^P_{V^*}(x)\}_{x \in L}$ and $\{M^*(x)\}_{x \in L}$ are computationally indistinguishable.*

We have mainly replaced $\langle P, V^* \rangle(x)$ with $view^P_{V^*}(x)$ since it is possible to compute in polynomial time the former from the latter. Although it is a less natural definition of *ZKP*, it is more convenient to use and it is equivalent to it [Gol06]. It is important to notice that this change can also be applied to the first definition of perfect zero-knowledge proof in the same manner (see Definition 4.1).

4.1.5 Graph Isomorphism

In section 4.1.2, we already established that every language in $\mathcal{BPP}$ is a trivial example of zero-knowledge proof system. We will now introduce a well-known example of a language not known to be in $\mathcal{BPP}$, we will give a *perfect ZKP* for the language and we will proof that it is indeed a perfect zero-knowledge proof system (as in Definition 4.1 but using the changes introduced in Definition 4.4).

Let *GI* (i.e. *Graph Isomorphism*) be the set of **pairs of isomorphic graphs**. Let us construct a perfect zero-knowledge proof for *GI*:

- **Common input**: A pair of two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Let ϕ be an isomorphism between the input graphs. Namely,

  $$\phi : V_1 \to V_2$$

  such that $(u_1, v_1) \in E_1 \iff (\phi(u_1), \phi(v_1)) = (u_2, v_2) \in E_2$.

- **Prover's first step** (P1): The prover selects a random isomorphic copy of G_2 and sends it to the verifier. Namely, the prover selects at random , with uniform probability distribution, a permutation π from the set of permutations over the vertex set V_2 and constructs a graph with vertex set V_2 and edge set

  $$F \stackrel{def}{=} \{(\pi(u), \pi(v)) : (u, v) \in E_2\}$$

  The prover sends (V_2, F) to the verifier.

- **Verifier's first step** (V1): Upon receiving a graph $G' = (V', E')$ from the prover, the verifier asks the prover to show an isomorphism between G' and one of the input graphs, chosen at random by he verifier. Namely, the verifier uniformly selects $\sigma \in \{1, 2\}$ and sends it to the prover (who is supposed to answer with an isomorphism between G_σ and G').

- **Prover's second step** (P2): If the message σ received from the verifier equals 2, then the prover sends π to the verifier. Otherwise (i.e. $\sigma \neq 2$ treated by the prover as $\sigma = 1$), the prover sends $\pi \circ \phi$ to the verifier.

- **Verifier's second step** (V2): If the message, denoted ψ, received from the prover is an isomorphism between G_σ and G', then the verifier outputs 1. Otherwise it outputs 0.

Let us denote the prover's program P_{GI}.

Both the prover's and verifier's programs are easily implemented in probabilistic polynomial time. We will now show that this pair of interactive machines constitutes a zero-knowledge interactive proof system for the language *GI*.

Proposition 4.5. *The language GI has a perfect zero-knowledge interactive proof system. Furthermore, the programs previously specified satisfy the following:*

1. *If G_1 and G_2 are isomorphic (i.e. $(G_1, G_2) \in GI$), then the verifier always accepts (when interacting with P_{GI}.*
2. *If G_1 and G_2 are not isomorphic (i.e. $(G_1, G_2) \notin GI$), then no matter with which machine the verifier interacts, it will reject the input with probability at least $\frac{1}{2}$.*
3. *The prover (i.e. P_{GI}) is perfect zero-knowledge. Namely, for every probabilistic polynomial-time interactive Turing machine V^*, there exists a probabilistic polynomial-time algorithm M^* outputting $\perp$ with probability at most $\frac{1}{2}$, so that for every $x \stackrel{def}{=} (G_1.G_2) \in GI$, the following two random variables are identically distributed:*
   - *$view_{V^*}^{P_{GI}}(x)$ (i.e. the view of V^* after interacting with P_{GI}, on common input x)*
   - *$m^*(x)$ (i.e. the output of machine M^*, on input x, conditioned on not being $\perp$).*

As always, we can reduce the error probability (only in the soundness condition) to 2^{-k} by repeating the protocol a number of k times. It is important that at every repetition, both prover and verifier use independent and new internal coin tosses.

We will proof the Proposition 4.5 in several steps. First, we will proof items 1 and 2. Then we will focus on item 3.

Proof (1). If the input graphs G_1 and G_2 are isomorphic, then the graph G' constructed in Step P1 will be isomorphic to both of them. Thus, if each party follows its prescribed program, then the verifier will always output 1. □

Proof (2). If the input graphs G_1 and G_2 are not isomorphic, then no graph presented to the prover can be isomorphic to both G_1 and G_2. It follows that, no

matter how the prover (cheating or not) constructs G', there exists $\sigma \in \{1,2\}$ such that G' and G_σ are *not* isomorphic. Thus, if the verifier follows its program, then it will output 0 with probability at least $\frac{1}{2}$. □

Before tackling the last part of the proof, we will need the next result:

Lemma 4.6. *Suppose that the graphs G_1 and G_2 are isomorphic. Let τ be a random variable uniformly distributed in $\{1,2\}$, and let Π be a random variable uniformly distributed over the set of permutations over V_τ. Then for every graph G'' that is isomorphic to G_1 (and G_2), it holds that*

$$\Pr\left[\tau = 1 \mid \Pi(G_\tau) = G''\right] = \Pr\left[\tau = 2 \mid \Pi(G_\tau) = G''\right] = \frac{1}{2}$$

Proof. We first claim that the sets $S_1 \stackrel{def}{=} \{\pi : \pi(G_1) = G''\}$ and $S_2 \stackrel{def}{=} \{\pi : \pi(G_2) = G''\}$ are of the same cardinality (this is a direct consequence of $G_1 \cong G_2$). Thus,

$$\begin{aligned}
\Pr\left[\Pi(G_\tau) = G'' \mid \tau = 1\right] &= \Pr\left[\Pi(G_1) = G''\right] \\
&= \Pr\left[\Pi \in S_1\right] \\
&= \Pr\left[\Pi \in S_2\right] \\
&= \Pr\left[\Pi(G_2) = G''\right] \\
&= \Pr\left[\Pi(G_\tau) = G'' \mid \tau = 2\right]
\end{aligned}$$

Using Bayes' rule (analogous for $\tau = 2$),

$$\begin{aligned}
\Pr\left[\tau = 1 \mid \Pi(G_\tau) = G''\right] &= \frac{\Pr\left[\Pi(G_\tau) = G'' \mid \tau = 1\right] \cdot \Pr\left[\tau = 1\right]}{\Pr\left[\Pi(G_\tau) = G''\right]} \\
&= \Pr\left[\tau = 1\right] = \frac{1}{2}
\end{aligned}$$

□

Let us begin with the last part of the proof. We will construct a simulator and proof it simulates the view of the verifier V^*.

Proof (3). Let us construct a probabilistic polynomial-time algorithm M^*. Let $x \stackrel{def}{=} (G_1, G_2)$ be the input. Then,

1. *Setting the random tape of V^**: Let $q(\cdot)$ denote a polynomial bounding the running time of V^*. The simulator M^* starts by uniformly selecting $r \in \{0,1\}^{q(|x|)}$ to be used as the content of the random tape of V^*.

2. *Simulating the prover's first step* (P1): The simulator M^* selects at random , with uniform probability distribution, a "bit" $\tau \in \{1,2\}$ and a permutation ψ from the set of permutations over the vertex set V_τ and edge set

$$F \stackrel{def}{=} \{(\psi(u), \psi(v)) : (u,v) \in E_\tau\},$$

and sets $G'' \stackrel{def}{=} (V_\tau, F)$.

3. *Simulating the verifier's first step* (V1): The simulator M^* initiates an execution of V^* by placing x on V^*'s common input tape, placing r on V^*'s random tape and placing G'' on V^*'s incoming message tape. After executing a polynomial number of steps of V^*, the simulator can read the outgoing message of V^*, denoted σ. To simplify the rest of the description, we normalize σ by setting $\sigma = 1$ if $\sigma \neq 2$ (and leave σ unchanged if $\sigma = 2$).

4. *Simulating the prover's second step* (P2): If $\sigma = \tau$, then the simulator halts with output (x, r, G'', ψ).

5. *Failure of the simulation*: If $\sigma \neq \tau$, then the simulator halts with output $\perp$.

Clearly, if V^* is polynomial-time, then so is M^*. We have shown that M^* outputs $\perp$ with probability at most $\frac{1}{2}$ (follows from Lemma 4.6). It is left to show that, conditioned on not outputting $\perp$, the simulator's output is distributed as the verifier's view in a "real interaction with P_{GI}". Namely,

Lemma 4.7. *Let $x = (G_1, G_2) \in GI$. Then for every string r, graph H, and permutation ψ, it holds that*

$$\Pr\left[view_{V^*}^{P_{GI}}(x) = (x, r, H, \psi)\right] = \Pr\left[M^*(x) = (x, r, H, \psi) \mid M^*(x) \neq \perp\right]$$

The proof of this lemma can be found in Appendix B and is extracted from [Gol06].

This ends the proof of Proposition 4.5. □

4.1.6 Auxiliary Input

When using zero-knowledge proof systems as sub-protocols inside larger protocols, the need of a private local input for each party arises. Though the general results stay the same, it is a generalized approach to add a local *auxiliary-input* tape to the definition of interactive Turing machine (and subsequently change the definitions based on it). Let us redefine some of these concepts.

Definition 4.8 (Interactive Turing Machine, Revisited). *An **interactive Turing machine** is defined as in Definition 3.2.3, except that the machine has an additional read-only tape called **the auxiliary-input tape**. The content of this tape is called **auxiliary input**.*

Definition 4.9 (Complexity of an ITM, Revisited). *The complexity of such an interactive Turing machine is defined as in Definition 3.22 (i.e. still measured as a function of the (common) input length, regardless of the content of its auxiliary input tape).*

Notation: We denote by $\langle A(y), B(z)\rangle(x)$ the random variable representing the (local) output of B when interacting with machine A on common input x, when the random input to each machine is uniformly and independently chosen, and A (resp. B) has auxiliary input y (resp. z).

Definition 4.10 (Interactive Proof System, Revisited). *A pair of interactive machines (P, V) is called an **interactive proof system** (see Definition 3.23) for a language L if machine V is polynomial-time and the following two conditions hold:*

- ***Completeness**: $\forall x \in L$, there exists a string y such that for every $z \in \{0,1\}^*$,*

$$\Pr\left[\langle P(y), V(z)\rangle(x) = 1\right] \geq \frac{2}{3}$$

- ***Soundness**: $\forall x \notin L$, every interactive Turing machine B and every $y, z \in \{0,1\}^*$,*

$$\Pr\left[\langle B(y), V(z)\rangle(x) = 1\right] \leq \frac{1}{3}$$

These changes lead to a revisited definition of (computational) zero-knowledge interactive proof systems. This is not done as a theoretical exercise, but rather as a practicality. It is common to have *ZKPs* as part of larger protocols and it can happen that the verifier has access to some additional a priori information, that may assist in its attempts to "extract knowledge" from the prover. Thus, we can rethink of an interaction to be zero-knowledge if whatever can be efficiently computed after interacting with the prescribed prover on input x and auxiliary-input z can be computed from x and z.

Definition 4.11 (Auxiliary-Input Zero-Knowledge). *Let (P, V) be an interactive proof for a language L (as in Definition 4.10). Denote by $P_L(x)$ the set of strings y satisfying the completeness condition with respect to $x \in L$ (i.e. $Pr[\langle P(y), V(z)\rangle(x) = 1] \geq \frac{2}{3}$ for every $z \in \{0,1\}^*$). We say that (P, V) is **zero-knowledge with respect to auxiliary input** (or is **auxiliary-input zero-knowledge**) if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic algorithm M^*, running in time polynomial in the length of its first input, such that the following two ensembles are computationally indistinguishable (when the distinguishing gap is considered as a function of $|x|$):*

- $\{\langle P(y_x), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$ *for arbitrary* $y_x \in P_L(x)$
- $\{M^*(x,z)\}_{x\in L, z\in\{0,1\}^*}$

Namely, for every probabilistic algorithm D with running time polynomial in the length of the first input, for every polynomial $p(\cdot)$, and for all sufficiently long $x \in L$, all $y \in P_L(x)$ and $z \in \{0,1\}^*$, it holds that

$$|\Pr[D(x,z,\langle P(y), V^*(z)\rangle(x)) = 1] - \Pr[D(x,z,M^*(x,z)) = 1]| < \frac{1}{p(|x|)}$$

The auxiliary inputs y (i.e. the prover's auxiliary input) and z (i.e. the verifier's auxiliary input) may not be known to the other party. Also, it is important to keep in mind that the prover, the verifier and the simulator all must run in time polynomial in the length of the common input x (and not on time polynomial on the total length of all their inputs). Thus, the verifier V^* reads at most a polynomial-long prefix (from the common input x) of its auxiliary input. A similar convention holds for the simulator M^* (i.e. its running time is polynomial in the length of its first input and consequently it may only read a prefix of the second input) [GO94].

In general, a demonstration of zero-knowledge can be extended to yield zero-knowledge with respect to auxiliary input whenever the simulator used in the original demonstration works by invoking the verifier's program as a black box [Gol06]. All simulation in this work have this property.

4.1.7 Sequential-Composition

We have seen in the Graph Isomorphism example (see Section 4.1.5) that by repeating the protocol, the soundness error quickly diminishes. However, does the *zero-knowledge* property hold? It seems intuitive that if a single repetition of the protocol doesn't yield any knowledge, several repetitions wouldn't as well. This concept is captured by the definition of zero-knowledge with respect to auxiliary input (see Definition 4.11) in the sense that, if an interaction yields no knowledge whatever the auxiliary input may be, then any information passed from repetition to repetition won't yield any knowledge [Hoh07].

Formally speaking, the definition of zero-knowledge interactive proof systems should be closed under sequential composition. It is known that, without the addition of auxiliary input to the *ZKP* definition, this wouldn't hold [Gol06]. This is idea is captured by the *sequential-composition lemma*.

Lemma 4.12 (**Sequential-Composition Lemma**). *Let P be an interactive Turing machine (i.e. a prover) that is zero-knowledge with respect to auxiliary input on some language L. Suppose that the last message sent by P, on input x, bears a special end-of-proof symbol. Let $Q(\cdot)$ be a polynomial, and let P_Q be an interactive Turing machine that, on common input x, proceeds in $Q(|x|)$ phases, each of them consisting of running P on common input x. (We stress that in case P is probabilistic, the interactive machine P_Q uses independent coin tosses for each of the $Q(|x|)$ phases). Then P_Q is zero-knowledge (with respect to auxiliary input) on L. Furthermore, if P is perfect zero-knowledge (with respect to auxiliary input), then so is P_Q.*

The convention concerning the end-of-proof symbol is introduced for technical purposes and every machine P can be easily modified so that its last message will output an appropriate symbol. The lemma doesn't take in account the effect that repetition has on the gap between the acceptance probabilities for inputs inside and outside the language (this aspect is already dealt with in Section 3.2.4). Then, zero-knowledge interactive proof systems with respect to auxiliary input can be repeated sequentially and do preserve the property of zero-knowledge, as well as diminishing the error probability. The proof of this lemma is long and can be found in [Gol06].

Note on Parallel Composition: Parallel composition does reduce the error probability in interactive proof systems, while maintaining the number of rounds. This is known to NOT preserve the property of zero-knowledge, as shown in [GK94].

4.1.8 Further Topics

Zero-knowledge proofs have many different uses and this work could expand in different possible directions. A very natural next step would be to study *Non-Interactive Zero-Knowledge Proof Systems* (*NIZKPs*), which are proofs that don't require interaction between the prover and the verifier. Their main advantage is their efficiency (due to the lack of interaction, the proof can be verified with fewer computational resources) and they are well suited for use in decentralized systems with few (if any) trusted parties (e.g. blockchain networks). Their main disadvantages are their lack of flexibility (compared to interactive *ZKPs*) and have a limited scope to specific types of information.

From *NIZKPs*, one could research some of its most famous protocols like the Succinct Non-Interactive Arguments of Knowledge (or *SNARKs*) and the Zero-Knowledge Scalable Transparent Arguments of Knowledge (or *STARKs*), which are currently at the front of research and application in the field.

Chapter 5

Conclusions

In the course of this work, we have established the theoretical foundations to *zero-knowledge interactive proof systems* through probability and computation theory, as well as a number of examples to depict its utility in real-world problems.

From the definition of Turing machine, we laid down the ideas of simulation and computation power to solve decision problems, captured by different complexity classes. We focused on probabilistic algorithms where decision problems are allowed to fail with a certain error. We have introduced the necessary concepts of probability underlying the idea of *repetition* to decrease the aforementioned error. We continued by familiarizing ourselves with the notion of *interaction* between (interacting) Turing machines, leading to the definition of interactive proof systems and their use cases. From these building blocks, we constructed the idea of interactive proof systems being *zero-knowledge* and saw some variations of this definition. Finally, we captured all these notions with the example of *Graph Isomorphism* and showed that sequential repetition holds the property of *zero-knowledge*.

Bibliography

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* (1989), pp. 186–208.

[GK94] Oded Goldreich and Hugo Krawczyk. "On the Composition of Zero-Knowledge Proof Systems". In: (1994).

[GO94] Oded Goldreich and Yair Oren. "Definitions and Properties of Zero-Knowledge Proof Systems". In: *Journal of Cryptology* (1994).

[Koz97] Dexter C. Kozen. "Automata and Computability". In: Springer New York, NY, 1997. Chap. 3.

[Gol06] Oded Goldreich. *Foundations of Cryptography Basic Tools*. Cambridge University Press, 2006.

[Hoh07] Susan Hohenberger. *Lecture 4: Zero-Knowledge Proofs II*. `https://www.cs.jhu.edu/~susan/600.641/scribes/lecture4.pdf`. 2007.

[KL07] Joanthan Katz and Yehuda Lindell. "Introduction to Modern Cryptography". In: Boca Raton : CRC PRESS, 2007. Chap. 10.

[AB09] Sanjeev Arora and Boaz Barak. "Computational Complexity: A Modern Approach". In: Cambridge University Press, 2009. Chap. 8.

[Vad12] Salil P. Vadhan. "Pseudorandomness". In: *Foundations and Trends in Theoretical Computer Science* (2012), pp. 1–336.

[Gas14] William Gasarch. *Lectures Notes on Turing Machines and P*. `https://www.cs.umd.edu/~gasarch/COURSES/452/F14/p.pdf`. 2014.

[Tre15] Luca Trevisan. *Notes on Zero Knowledge*. `https://lucatrevisan.github.io/teaching/cs172-15/notezk.pdf`. 2015.

[Rub17] Ronitt Rubinfeld. *Lectures Notes on Randomness and Computation*. `https://people.csail.mit.edu/ronitt/COURSE/F17/NOTES/lec6-scribe.pdf`. 2017.

[Sch24] Berry Schoenmakers. *Lectures Notes on Cryptographic Protocols*. `https://www.win.tue.nl/~berry/CryptographicProtocols/LectureNotes.pdf`. 2024.

Appendix A

Quadratic Residuosity Protocol

In this annex we will present a small program written in *c* that replicates the *quadratic residuosity protocol* from section 4.1.1. We will be using a very simple set of numbers: $N = 11 \cdot 13 = 143$, $r = 82$ and $x = 15$ since $x^2 \equiv r \mod N$. We will also review the output to show that the theoretical results match the practical computations. Let us first introduce the program. It has two main functionalities: compute the protocol with a single interaction (for 10.000 iterations) and compute the protocol with $1 \leq k \leq 10$ interactions (for 10.000 iterations each k). There are 6 different functions:

- `main`: declares and initializes the input values for the protocol and executes it for both aforementioned functionalities.

- `randRange`: used to balance the function `rand()`. When we use variables with value `rand ()% N;`, we don't get a uniform distribution so we ignore the last numbers in $[$`0, RAND_MAX`$]$ (i.e. we ignore the numbers in (`RAND_MAX - ( RAND_MAX % N), RAND_MAX`$]$.

- `smart_prover`: a single run of the protocol in the case that the prover does provide a valid proof.

- `dumb_prover`: a single run of the protocol in the case that the prover does not provide a valid proof.

- `single_interaction`: computes the single interaction protocol 10.000 times for both `smart_prover` and `dumb_prover` and writes the average success probability in two different files. The results are two columns of the form

$$(\textit{iterations}, \textit{success probability})$$

- `multiple_interaction`: computes the k interactions protocol 10.000 times for both `smart_prover` and `dumb_prover` and writes the average success probability in two different files. The results are two columns of the form

$$(k, \textit{success probability})$$

We now present the results by means of two plots (one for a single interaction and one for multiple interactions).

In Figure A.1, we can clearly see that a "smart" prover (i.e. having a valid proof) is successful with a probability of 1 in every case. However, a "dumb" prover (i.e. not having a valid proof) fails with probability $\approx \frac{1}{2}$. The graph for a "dumb" prover shows some fluctuation for lower iterations, but it stabilizes to its theoretical probability by the law of large numbers.

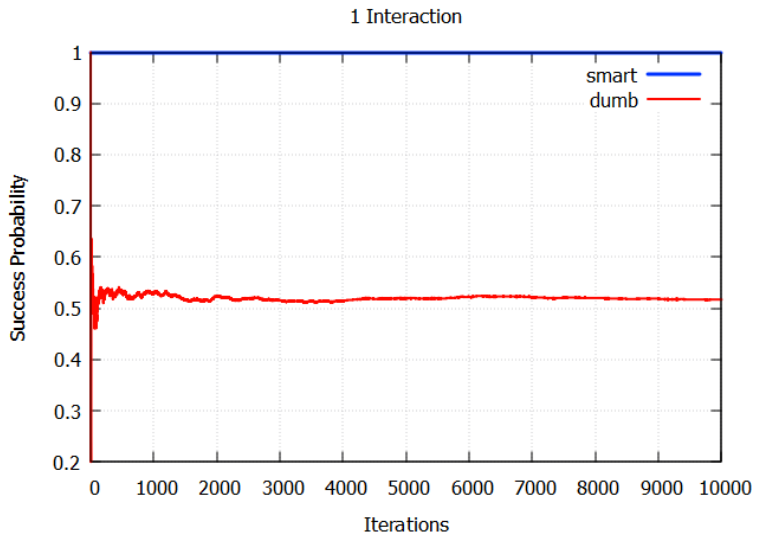

Figure A.1: Success probability for 1 interaction as a "smart" and "dumb" prover

In Figure A.2, the X-axis is the number of k interactions between prover and verifier. We can see an identical result for the "smart" prover (since the probability is always 1 when having a valid proof), but the success probability for the "dumb" prover has an almost exponential decrease as interactions increase. This is also in line with our theoretical results, where we showed that the success probability of a cheating prover should decrease with a rate of 2^{-k}. Indeed, for 1 interaction we get $\approx \frac{1}{2}$ success probability, for 2 interactions $\approx \frac{1}{4}$ success probability, for 3 interactions $\approx \frac{1}{8}$ success probability, etc.

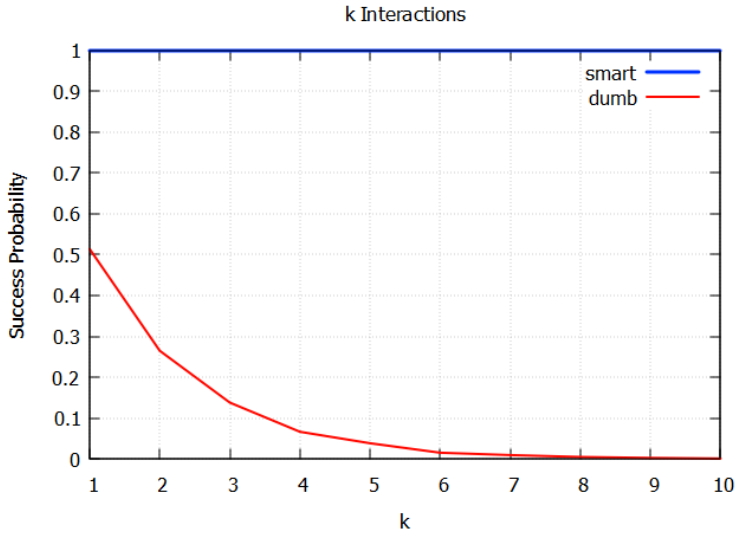

Figure A.2: Success probability for k interactions as a "smart" and "dumb" prover

We now present the entire code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define ITER 10000

void single_interaction(int N, int r, int x);
void multiple_interaction(int N, int r, int x, int k_max);
int smart_prover(int N, int r, int x);
int dumb_prover(int N, int r);
int randRange(int N);

int main()
{
  int N, r, x, k_max;

  srand(time(NULL));

  // Common input
  N = 143;
  r = 82;

  // Prover's private input
```

```
24    x = 15;
25
26    // Maximum interactions
27    k_max = 10;
28
29    single_interaction(N, r, x);
30    multiple_interaction(N, r, x, k_max);
31
32    return 0;
33  }
34
35  void single_interaction(int N, int r, int x)
36  {
37    int i;
38    double smart_sum, dumb_sum;
39    char smart_file[30] = "smart_single.txt", dumb_file[30] = "
        dumb_single.txt";
40    FILE *f;
41
42    f = fopen(smart_file, "w");
43    if (f == NULL)
44    {
45      printf("Error opening %s!\n", smart_file);
46      exit(1);
47    }
48
49    smart_sum = 0.;
50    for (i = 1; i <= ITER; i++)
51    {
52      smart_sum += smart_prover(N, r, x);
53      fprintf(f, "%d %5.3lf\n", i, smart_sum / i);
54    }
55
56    fclose(f);
57    f = fopen(dumb_file, "w");
58    if (f == NULL)
59    {
60      printf("Error opening %s!\n", dumb_file);
61      exit(1);
62    }
63
64    dumb_sum = 0.;
65    for (i = 1; i <= ITER; i++)
66    {
```

```
67      dumb_sum += dumb_prover(N, r);
68      fprintf(f, "%d %5.3lf\n", i, dumb_sum / i);
69    }
70
71    fclose(f);
72
73    return;
74  }
75
76  void multiple_interaction(int N, int r, int x, int k_max)
77  {
78    int i, j, k, pass;
79    double smart_sum, dumb_sum;
80    char smart_file[30] = "smart_multiple.txt", dumb_file[30] = "
        dumb_multiple.txt";
81    FILE *f;
82
83    f = fopen(smart_file, "w");
84    if (f == NULL)
85    {
86      printf("Error opening %s!\n", smart_file);
87      exit(1);
88    }
89
90    for (k = 1; k <= k_max; k++)
91    {
92      smart_sum = 0.;
93      for (i = 1; i <= ITER; i++)
94      {
95        pass = smart_prover(N, r, x);
96        for (j = 1; j < k && pass == 1; j++)
97          pass = smart_prover(N, r, x);
98        smart_sum += pass;
99      }
100     fprintf(f, "%d %5.3lf\n", k, smart_sum / ITER);
101   }
102
103   fclose(f);
104   f = fopen(dumb_file, "w");
105   if (f == NULL)
106   {
107     printf("Error opening %s!\n", dumb_file);
108     exit(1);
109   }
```

```
110
111   for (k = 1; k <= k_max; k++)
112   {
113     dumb_sum = 0;
114     for (i = 1; i <= ITER; i++)
115     {
116       pass = dumb_prover(N, r);
117       for (j = 1; j < k && pass == 1; j++)
118       {
119         pass = dumb_prover(N, r);
120       }
121       dumb_sum += pass;
122     }
123     fprintf(f, "%d %7.5lf\n", k, dumb_sum / ITER);
124   }
125
126   fclose(f);
127
128   return;
129 }
130
131 int smart_prover(int N, int r, int x)
132 {
133   int y, a, b, c;
134
135   // Prover's first step: choose a random y < N and calculate a
          = y*y (mod N)
136   y = randRange(N);
137   a = y * y % N;
138
139   // Verifier's first step: flipping a coin b.
140   b = randRange(2);
141
142   // Prover's second step: c = y if b = 0 or c = y*x (mod N) if
          b = 1
143   if (b == 0)
144     c = y;
145   else
146     c = y * x % N;
147
148   // Verifier\'s second step: calculating the output.
149   if (b == 0)
150     return ((c * c % N) == a);
151
```

```
152   return ((c * c % N) == (a * r % N));
153 }
154
155 int dumb_prover(int N, int r)
156 {
157   int y, a, b, c;
158
159   // Prover's first step: choose a random y < N and calculate a
          = y*y (mod N)
160   y = randRange(N);
161   a = y * y % N;
162
163   // Verifier's first step: flipping a coin b.
164   b = randRange(2);
165
166   // Prover's second step: c = y if b = 0 or choose a RANDOM c
          < N if b = 1
167   if (b == 0)
168     c = y;
169   else
170     c = randRange(N);
171
172   // Verifier\'s second step: calculating the output.
173   if (b == 0)
174     return ((c * c % N) == a);
175
176   return ((c * c % N) == (a * r % N));
177 }
178
179 int randRange(int N)
180 {
181   int limit, r;
182
183   limit = RAND_MAX - (RAND_MAX % N);
184
185   while ((r = rand()) >= limit)
186     ;
187
188   return r % N;
189 }
```

Appendix B

Graph Isomorphism Proof

Lemma B.1. *Let $x = (G_1, G_2) \in GI$. Then for every string r, graph H, and permutation ψ, it holds that*

$$\Pr\left[view_{V^*}^{P_{GI}}(x) = (x, r, H, \psi)\right] = \Pr\left[M^*(x) = (x, r, H, \psi) \mid M^*(x) \neq \perp\right]$$

Proof. Let $m^*(x)$ describe $M^*(x)$ conditioned on its output not being $\perp$. We first observe that both $m^*(x)$ and $view_{V^*}^{P_{GI}}(x)$ are distributed over quadruples of the form $(x, r, \cdot, \cdot)$, with uniformly distributed $r \in \{0,1\}^{q(|x|)}$. Let $\nu(x, r)$ be a random variable describing the last two elements of $view_{V^*}^{P_{GI}}(x)$ conditioned on the second element equaling r. Similarly, let $\mu(x, r)$ describe the last two elements of $m^*(x)$ (conditioned on the second element equaling r). We need to show that $\nu(x, r)$ and $\mu(x, r)$ are identically distributed for every x and r.

Observe that once r is fixed, the message sent by V^*, on common input x, random tape r, and incoming message H, is uniquely defined. Let us denote this message by $\nu^*(x, r, H)$. We show that both $\nu(x, r)$ and $\mu(x, r)$ are uniformly distributed over the set

$$C_{x,r} \stackrel{\text{def}}{=} \left\{(H, \psi) : H = \psi(G_{\nu^*(x,r,H)})\right\}$$

where (again) $\psi(G)$ denotes the graph obtained from G by relabeling the vertices using the permutation ψ (i.e. if $G = (V, E)$, then $\psi(G) = (V, F)$, so that $(u, v) \in R$ iff $(\psi(u), \psi(v)) \in F$).

To prove this last claim, we will use results related to the automorphism group of the graph G_2 (i.e. the set of permutations π for which $\pi(G_2)$ is identical with G_2, not just isomorphic to G_2). For simplicity, consider first the special case in which the automorphism group of G_2 consists of merely the identity permutation.

In this case, $(H,\psi) \in C_{x,r}$ iff H is isomorphic to (both G_1 and) G_2 and ψ is the (unique) isomorphism between H and $G_{v^*(x,r,H)}$. Hence, $C_{x,r}$ contains exactly $|V_2|!$ pairs, each containing a different graph H as the first element. In the general case, $(H,\psi) \in C_{x,r}$ iff H is isomorphic to (both G_1 and) G_2 and ψ is an isomorphism between H and $G_{v^*(x,r,H)}$. We stress that $v^*(x,r,H)$ is the same in all pairs containing H. Let $\mathrm{aut}(G_2)$ denote the size of the automorphism group of G_2. Then each H (isomorphic to G_2) appears in exactly $\mathrm{aut}(G_2)$ pairs of $C_{x,r}$, and each such pair contains a different isomorphism between H and $G_{v^*(x,r,H)}$. The number of different H's that are isomorphic to G_2 is $|V_2|!/\mathrm{aut}(G_2)$, and so $|C_{x,r}| = |V_2|!$ also in the general case.

We first consider the random variable $\mu(x,r)$ (describing the suffix of $m^*(x)$). Recall that $\mu(x,r)$ is defined by the following two-step process:

1. One selects uniformly a pair (τ,ψ), over the set of pairs $(\{1,2\} \times \text{permutation})$, and sets $H = \psi(G_\tau)$.

2. One outputs (i.e. sets $\mu(x,r)$ to) $(\psi(G_\tau),\psi)$ if $v^*(x,r,H) = \tau$ (and ignores the (τ,ψ) pair otherwise).

Hence, each graph H (isomorphic to G_2) is generated, at the first step, by exactly $\mathrm{aut}(G_2)$ different $(1,\cdot)$-pairs (i.e. the pairs $(1,\psi)$ satisfying $H = \psi(G_1)$) and by exactly $\mathrm{aut}(G_2)$ different $(2,\cdot)$-pairs (i.e. the pairs $(2,\psi)$ satisfying $H = \psi(G_2)$). All these $2 \cdot \mathrm{aut}(G_2)$ pairs yield the same graph H and hence lead to the same value of $v^*(x,r,H)$. It follows that out of the $2 \cdot \mathrm{aut}(G_2)$ pairs of the form (τ,ψ) that yield the graph $H = \psi(G_2)$, only the $\mathrm{aut}(G_2)$ pairs satisfying $\tau = v^*(x,r,H)$ lead to an output. Hence, for each H (that is isomorphic to G_2), the probability that $\mu(x,r) = (H,\cdot)$ equals $\mathrm{aut}(G_2)/(|V_2|!)$. Furthermore, for each H (that is isomorphic to G_2),

$$\Pr[\mu(x,r) = (H,\psi)] = \begin{cases} \frac{1}{|V_2|!} & \text{if } H = \psi(G_{v^*(x,r,H)}) \\ 0 & \text{otherwise} \end{cases}$$

Hence $\mu(x,r)$ is uniformly distributed over $C_{x,r}$.

We now consider the random variable $v(x,r)$ (describing the suffix of the verifier's view in a "real interaction" with the prover). Recall that $v(x,r)$ is defined by selecting uniformly a permutation π (over the set V_2) and setting $v(x,r) = (\pi(G_2),\pi)$ if $v^*(x,r,\pi(G_2)) = 2$, and $v(x,r) = (\pi(G_2),\pi \circ \phi)$ otherwise, where ϕ is the isomorphism between G_1 and G_2. Clearly, for each H (that is isomorphic to G_2), the

probability that $\nu(x,r) = (H,\cdot)$ equals $\mathrm{aut}(G_2)/(|V_2|!)$. Furthermore, for each H (that is isomorphic to G_2),

$$\Pr[\nu(x,r) = (H,\psi)] = \begin{cases} \frac{1}{|V_2|!} & \text{if } \psi = \pi \circ \phi^{2-\nu^*(x,r,H)} \\ 0 & \text{otherwise} \end{cases}$$

Observing that $H = \psi(G_{\nu^*(x,r,H)})$ iff $\psi = \pi \circ \phi^{2-\nu^*(x,r,H)}$, we conclude that $\mu(x,r)$ and $\nu(x,r)$ are identically distributed.

The claim follows. □