A Taxonomy of Probabilistic Proof Systems

Mark Bebawy
Vrije Universiteit Amsterdam, The Netherlands
markbebawy0@gmail.com

ABSTRACT

Many probabilistic proof systems are currently being studied independently. We create a comprehensive overview of probabilistic proof systems, by creating a taxonomy of their structure, and by analyzing the *zero-knowledge* property of proof systems, which roughly states that provers do not reveal information beyond the fact that a statement is true. Essentially, a proof systems consists of a prover that wants to prove a statement to a verifier. Our taxonomy categorizes proof systems based on three variables: whether they are interactive, whether the verifier has full or bounded (oracle) proof access, and whether the verifier has full or bounded (oracle) input access. These three variables fully categorize many probabilistic proof systems that appear in peer-reviewed studies in this field. Furthermore, we found that zero-knowledge has been studied explicitly for all proof systems in our taxonomy, except for *interactive oracle proofs of proximity* (an interactive proof system where the verifier has bounded proof and bounded input access).

KEYWORDS

Probabilistic Proof System, NP, MA, Interactive Proof System, PCP, IOP, MAP, IPP, PCPP, IOPP, Zero-Knowledge Proof

1 INTRODUCTION

This survey focuses on what is currently known about (probabilistic) proof systems. The concept of a proof has changed throughout the years. Traditionally, proofs were static objects written by a prover that could be verified by anyone wanting to read the proof. With the emergence of randomized and interactive computations, and with the emergence of cryptography (where one often wants to prove that a computation has been done correctly without revealing secrets involved in the computation), the concept of a proof has significantly broadened [32]. Many proof systems were developed, and the goal of proof systems shifted from revealing as much knowledge about a statement as possible, to proving as much as possible with as little communication as possible. These proof systems have been developed independently, and are scattered across many different surveys and papers. This work aims to unify all this knowledge into one taxonomy that provides insight on the reasons these proof systems have been developed.

Our approach involved a search for peer-reviewed literature on proof systems, starting at surveys and extending our search with more specific queries as we got deeper into the field. The main contributions of this survey are a taxonomy of the structure of proof systems (Section 6), and the analysis of the so-called *zero-knowledge* property that proof systems may have (Section 7). Our taxonomy classifies proof systems according to their structure, where the three main variables at play are interactivity, bounding proof access, and bounding input access. For zero-knowledge, which roughly states that provers do not reveal information beyond the fact that the statement at hand is true, we found that for all proof systems except for one zero-knowledge has been studied. The *interactive oracle proof of proximity* (an interactive proof system with bounded proof and bounded input access) is the only exception.

This study serves as an introduction to the field of probabilistic proof systems. Furthermore, for researchers in the field of proof systems, this study may shine more light on the different design principles that are at the root of the proof systems, and may allow for a more structured overview of the field.

2 RELATED WORK

Probabilistic proof systems have been studied for multiple decades [22]. Throughout these years, several surveys have been written about different probabilistic proof systems [28–30, 32, 42]. These surveys mostly cover $\mathcal{NP}$-proofs, interactive proof systems, probabilistically checkable proofs (PCPs), and (non)-interactive zero-knowledge proofs. However, more recently other types of probabilistic proof systems have been developed, such as interactive oracle proofs (IOPs) [12], Merlin-Arthur proofs of proximity (MAPs) [35], interactive proofs of proximity (IPPs) [45], probabilistically checkable proofs of proximity (PCPPs) [14], and interactive oracle proofs of proximity (IOPPs) [11]. For most of these proof systems, the zero-knowledge variant has also been studied [15, 16, 19, 30, 37, 38, 40].

In this survey, we will create a taxonomy of both the proof systems that have been studied in the literature, and some design principles that have been employed more recently.

3 STUDY DESIGN

3.1 Research Goal

The main goal of this study is to give a comprehensive overview of the literature on probabilistic proof systems by creating a taxonomy of their different structures. Such an overview can function as an introduction to the field of probabilistic proof systems, as it will unify the systems that are currently scattered across different surveys and studies.

A second (but equally important) goal is to understand the design principles that are at the foundation of the different proof systems. Understanding these design principles can help future researchers in understanding the techniques that are used in proof systems, and to determine whether techniques used in one proof system are useful in another.

3.2 Research Questions

To achieve our research goals, we first need to discuss the different probabilistic proof systems that have been developed. We focus on the design choices that are at their foundation, and we discuss the trade-offs of each proof system. We also compare them and classify them. In summary, we need to answer the following research questions.

- Which type of probabilistic proof systems exist, and what design principles are at the basis of these proof systems?
- What is the power of the different probabilistic proof systems, and in what way can they be classified?

3.3 Initial search

Initially, we used Google Scholar and ACM Digital Library to search for surveys on probabilistic and interactive proofs (using the query `(Probabilistic OR Interactive) proof`). We mainly used Google Scholar, as it points to many different digital libraries. We also used ACM Digital Library, as many studies in this field have appeared in ACM Symposiums. We read the abstracts and contents of the surveys to find more specific search queries for the different proof systems.

For each more specific search query, we were interested in peer-reviewed primary studies. The search queries that we obtained from our initial search were: `Interactive proof`, `Interactive Oracle Proofs`, `Probabilistically Checkable Proofs`, `Interactive * Proofs of Proximity`, `* Sigma Protocol zero knowledge`, and `Fiat-Shamir`. We applied a simple snowballing approach to find more primary studies on these topics, which we discuss in Section 3.5. Finally, as we were interested in cryptographic applications, we searched for the `Zero-Knowledge proof` variant of each proof system.

3.4 Application of selection criteria

In this section, we present the selection criteria that we used during our search for literature on probabilistic proof systems. We mainly selected peer-reviewed papers and surveys on one or more proof systems and/or zero-knowledge. We also selected a couple of textbooks that we mostly used for the preliminaries (Section 4). For the primary studies, we only included papers that satisfied all inclusion criteria and none of the exclusion criteria.

I1- Studies focusing on one or more probabilistic proof systems and/or on zero-knowledge. The aim of this inclusion criterion is to narrow down the topic of the studies to those relevant for our research goal. Textbooks are allowed if they explain preliminary topics needed to understand the primary studies on probabilistic proof systems.

I2- Studies that have been published in journals or as preprints of conferences, in order to ensure that the included studies are peer-reviewed. This way, we minimize the chance of including results that contain mistakes in their proofs.

E1- Studies in blog posts or tutorials, as they miss the required rigour and level of detail that is needed for a representative taxonomy.

E2- Studies of which the full text is not available, as we cannot inspect them.

3.5 Snowballing

We applied a very simple snowballing approach in our search for literature. Whenever we encountered a survey or a paper on previously developed proof systems, we looked into the references of those papers to find the original study on the proof system. Furthermore, whenever we found studies that applied certain proof systems, we also looked in the references for the original paper. Lastly, we also did 'reverse snowballing' to search for applications of proof systems. We did this by clicking on 'Cited by' on Google Scholar for the original studies of the proof systems.

3.6 Study Replicability

Our study mainly consisted of searching for surveys and studies on proof systems and zero-knowledge. We selected the studies to base our taxonomy on by reading the abstracts of the papers and focusing on those that discuss the proof systems in detail. Our selection resulted in many primary studies [2–15, 18–27, 33–41, 43, 45, 46], some surveys [16, 28–30, 32, 42, 47], two courses [17, 44], and some textbooks [1, 31] to clarify some preliminaries needed for the discussion on proof systems.

We read the literature and mostly focused on the formal definitions of the proof systems. We classified the proof systems according to three questions:

(1) Is the proof system interactive?
(2) Does the proof system provide full or bounded proof access to the verifier?
(3) Does the proof system provide full or bounded input access to the verifier?

After this classification, we found a 'gap' in our taxonomy: a non-interactive proof system was missing that provides full proof access and bounded input access. We searched for such a proof system (using the query `Non-interactive proof of proximity`), and found *Merlin-Arthur proofs of proximity* [35]. For each proof system, we analyzed whether its zero-knowledge variant has been studied as well. There was no data extraction or data synthesis involved.

4 PRELIMINARIES

In this section, we revisit the definitions of the complexity classes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{MA}$, and co$\mathcal{NP}$, as the power of probabilistic proof systems is expressed by the relation between the classes they can prove membership of, and the complexity classes we discuss in this section.

4.1 Complexity Classes

We give a couple of definitions to formalize complexity classes. These definitions follow the book *Computational Complexity* by Arora and Barak [1]. We start by defining the concept of a *language* and a *relation*.

Definition 4.1. Given an $n \in \mathbb{N}$ and a function $f\colon \{0,1\}^n \longrightarrow \{0,1\}$, a *language* L (for f) is the set $L = \{x \in \{0,1\}^n \mid f(x) = 1\}$. We identify the function f with the language L.

Remark 4.2. Although we focus on the set $\{0,1\}^n$ for concreteness, note that any finite object in computer science can be represented as a finite bit-string. We will therefore use finite objects and bit-strings interchangeably throughout this paper.

Remark 4.3. In general, when we refer to a language, we are in fact referring to an infinite family of languages, each for a given bit-length. That is, by a language L we in fact refer to a sequence $(L_{n_i})_{i=1}^{\infty}$ where each $L_{n_i} \subseteq \{0,1\}^{n_i}$ and $n_1 < n_2 < \ldots$ is an infinite monotonically increasing sequence of natural numbers. However, for simplicity we will often omit this more cumbersome notation.

Many languages are obtained from *relations*.

Definition 4.4. A *relation* R is a subset $R \subseteq \{0,1\}^n \times \{0,1\}^m$ for some $n, m \in \mathbb{N}$.

Remark 4.5. In computer science, we often interpret relations as pairs (x, w) of an *instance* from a language and some *witness* for that instance. In fact, given a relation R, we can define the corresponding language

$$L(R) = \{x \mid \exists w, (x, w) \in R\}.$$

As relations and languages are related, we will use them somewhat interchangeably throughout this paper.

For complexity classes, we will be interested in algorithms that decide whether words are in a language. The complexity of these algorithms will be expressed in the number of steps they need as a function of the *size* of the input. We formalize these concepts as follows.

Definition 4.6. Let L be a language and let $x \in L$. We define the *size* of x, denoted $|x|$, as the length of the bit-string representation of x. We write $\mathrm{poly}(|x|)$ (respectively $\exp(|x|)$) for a function that grows at most polynomially (respectively exponentially) in $|x|$.

Definition 4.7. A language L is said to be *decided* by a deterministic algorithm $\mathcal{A}$ if given an input x, $\mathcal{A}(x) = 1 \iff x \in L$.

Definition 4.8. A *complexity class* is a set of languages that are decided by algorithms that adhere to some resource bounds.

Next, we define the classes $\mathcal{P}$ and $\mathcal{N}P$.

Definition 4.9. The class $\mathcal{P}$ *(polynomial-time)* is the set of languages L for which a deterministic algorithm exists that decides L in a number of steps that is polynomial in $|x|$. We say that the algorithm runs in *polynomial-time*.

Definition 4.10. The class $\mathcal{NP}$ *(nondeterministic polynomial-time)* is the set of relations R for which a deterministic algorithm exists that, given an $(x, w) \in R$ with $|w| \le \mathrm{poly}(|x|)$, is able to verify in $\mathrm{poly}(|x|)$ steps that w is a valid witness for $x \in L(R)$.

Intuitively, the class $\mathcal{P}$ consists of all problems that can be solved in polynomial-time, whereas the class $\mathcal{NP}$ consists of all problems for which proposed solutions can be *verified* in polynomial-time. We give an example of a concrete $\mathcal{N}P$ language.

Example 4.11. We say a graph $G = (V, E)$ is 3*-colorable* if all vertices can be colored using only three colors in such a way that each pair of adjacent vertices is colored differently. Suppose f is a function that maps finite graphs G to $\{0,1\}$ such that

$$f(G) = 1 \iff G \text{ is 3-colorable.}$$

Then, $L = \{G \mid f(G) = 1\}$ is a *language*[1] consisting of all graphs that are 3-colorable. This language is in $\mathcal{NP}$, as given a graph G and a coloring w, we can check that w is a 3-coloring of G by checking for each vertex in G that all neighboring vertices have different colors. If G has n vertices, this takes at most $\binom{n}{2}$ comparisons.

In this example, a 3-colorable graph G is an *instance*. A 3-coloring w of G is a *witness*, and the pair (G, w) is an element of the relation of 3-colorable graphs.

Now, we consider algorithms that run in *probabilistic polynomial-time (PPT)*. That is, the algorithm runs in a number of steps that is polynomial in the length of the input, and the algorithm is given access to an infinite unbiased tape of random coin tosses. In other words, at the cost of one time step, the algorithm may toss a random coin. As such algorithms will depend on random coin tosses, there will be some probability for which the algorithm could fail. Adding randomness in this way allows us to extend the class $\mathcal{P}$ to $\mathcal{BPP}$.

Definition 4.12. The class $\mathcal{BPP}$ *(bounded-error probabilistic polynomial time)* is the set of relations R for which a PPT algorithm V exists (whose random coin tosses are determined by a random variable ρ) that given an $x \in L(R)$ outputs 1 with probability at least $\frac{2}{3}$, denoted $\mathbb{P}[V(x;\rho) = 1] \ge \frac{2}{3}$, and if $x \notin L(R)$, then $\mathbb{P}[V(x;\rho) = 1] \le \frac{1}{3}$.

Analogously, we can extend the notion of $\mathcal{NP}$ to $\mathcal{MA}$ by allowing for randomness.

Definition 4.13 ([7])**.** The class $\mathcal{MA}$ *(Merlin-Arthur)* is the set of relations R for which a PPT algorithm V exists such that given an $(x, w) \in R$, we have $\mathbb{P}[V(x, w;\rho) = 1] \ge \frac{2}{3}$, and given an $x \notin L(R)$, $\mathbb{P}[V(x, w;\rho) = 1] \le \frac{1}{3}$.

Finally, we define the complexity class co$\mathcal{NP}$.

Definition 4.14. Given an $n \in \mathbb{N}$, a function $f\colon \{0,1\}^n \to \{0,1\}$, and a language $L = \{x \in \{0,1\}^n \mid f(x) = 1\}$, we define the *complemented language* $\overline{L} = \{0,1\}^n \setminus L = \{x \in \{0,1\}^n \mid f(x) = 0\}$.

Definition 4.15. The class co$\mathcal{NP}$ (complement of $\mathcal{NP}$) is the set of languages L for which a deterministic algorithm exists that given an $x \notin L$ and a witness w is able to verify in $\mathrm{poly}(|x|)$ steps that w indeed proves that $x \notin L$. Equivalently, co$\mathcal{NP} = \{L \mid \overline{L} \in \mathcal{NP}\}$.

Lemma 4.16. *We have $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{MA}$. It is widely believed that $\mathcal{P} \subsetneq \mathcal{NP}$ and $\mathcal{NP} = \mathcal{MA}$, but we have no proofs for these beliefs.*

Note that it is not known how $\mathcal{BPP}$ relates to $\mathcal{NP}$; the former could be a subset of the latter, or the other way around. They could also be equal. The relations between most of the complexity classes discussed here are presented in Figure 1.

5 PROTOCOL STRUCTURES

In this section, we discuss the needed knowledge for understanding all components of our taxonomy (see Figure 14). We focus on the *structure* of proof systems. In Section 7, we will discuss the *zero-knowledge* property that proof systems can have.

In Section 5.1, we start by discussing the type of problems and statements that have motivated the emergence of probabilistic proof systems. The subsequent subsections are structured according to our taxonomy. The subsections represent the design choices that underlie the different proof systems. Each paragraph inside a subsection discusses a different proof system.

[1] Note that, as announced in Remark 4.2, here we use graphs interchangeably with any finite bit-string representation of them. Furthermore, as announced in Remark 4.3, the language of 3-colorable graphs is in fact a sequence of languages $(L_{n_i})_{i=1}^{\infty}$, where $n_i = \binom{i}{2}$ and L_{n_i} consists of all 3-colorable graphs that have exactly i vertices. The reason we need n_i bits to encode a graph of i vertices, is that we encode each possible edge with a 1 if it appears in the graph and otherwise with a 0.

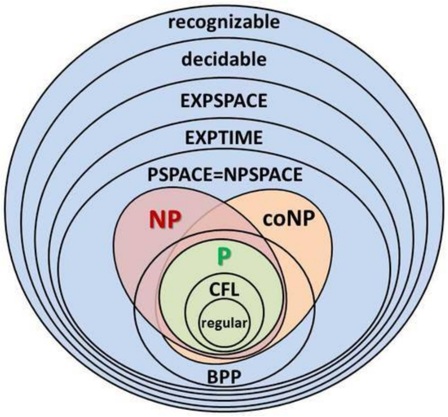


Figure 1: Relationship between the different complexity classes. We only discussed the classes $\mathcal{P}, \mathcal{NP}, \text{co}\mathcal{NP}$, and $\mathcal{BPP}$ inside $\mathcal{PSPACE}$ (see Theorem 5.9 for the definition of $\mathcal{PSPACE}$). Additionally, we discussed $\mathcal{MA}$, which contains $\mathcal{NP}$ and $\mathcal{BPP}$. *Reprinted from the 'Introduction to the Theory of Computation' course at Pennsylvania State University [44].*

The first design choice in proof systems is whether we want interactivity, which is the topic of Subsection 5.2. This choice will introduce the first three proof systems, namely *$\mathcal{NP}$-proofs* and *$\mathcal{MA}$-proofs* in the non-interactive branch, and *interactive proofs ($\mathcal{IP}$)* in the interactive branch. Then, aiming to reduce the amount of communication in a proof system, we consider proof systems where parties have bounded access to the proof (i.e., to the prover's messages) in Subsection 5.3. This will introduce *Probabilistically Checkable Proofs* in the non-interactive branch and *Interactive Oracle Proofs* in the interactive branch. Next, aiming to reduce the time-complexity of a verifier to sublinear time, in Subsection 5.4 we give parties bounded access to the input. This will introduce *Merlin-Arthur proofs of proximity* in the non-interactive branch and *Interactive Proofs of Proximity* in the interactive branch. Finally, we combine the two bounded properties to consider proof systems with both bounded proof and bounded input access. This gives *Probabilistically Checkable Proofs of Proximity* in the non-interactive branch and *Interactive Oracle Proofs of Proximity* in the interactive branch.

Throughout this section, proof systems (paragraphs) that are prepended by an asterisk ($*$) are non-interactive, as opposed to the other proof systems that are interactive.

In Subsection 5.6, we discuss methods to prove that a proof system is *sound* (i.e., that a proof system can prove false statements with very small probability only). Finally, in Subsection 5.7 we discuss the *Fiat-Shamir Transform*, which roughly allows us to translate interactive proof systems to non-interactive proof systems, for the sake of reducing the complexity that interactivity may introduce. This transform requires a strong assumption, namely the so-called *random oracle model (ROM).*

5.1 Type of statements to prove

The goal of designing probabilistic proof systems is to be able to prove more statements than possible with traditional proofs. In this section, we discuss some of the problems for which probabilistic proof systems are being used. This section is based on the book *Computational Complexity* by Arora and Barak [1].

A well-known $\mathcal{NP}$-problem is the *boolean satisfiability problem (SAT)*, where given a boolean formula the question is whether there exists an assignment of the variables such that the formula evaluates to true.

A natural generalization of SAT is the *circuit SAT* problem. Roughly, a *boolean circuit* is a set of gates (representing boolean operations) and wires (representing inputs and outputs) such that each gate gets one or more inputs and produces one output by applying the boolean operation to its inputs. The entire circuit gets one or more inputs and produces one output. Figure 2 shows a simple boolean circuit. The circuit SAT problem then asks whether we can find inputs to the circuit such that its output is true (in Figure 2, inputs $\{p \mapsto 0, q \mapsto 1, r \mapsto 0\}$ make the circuit output true).

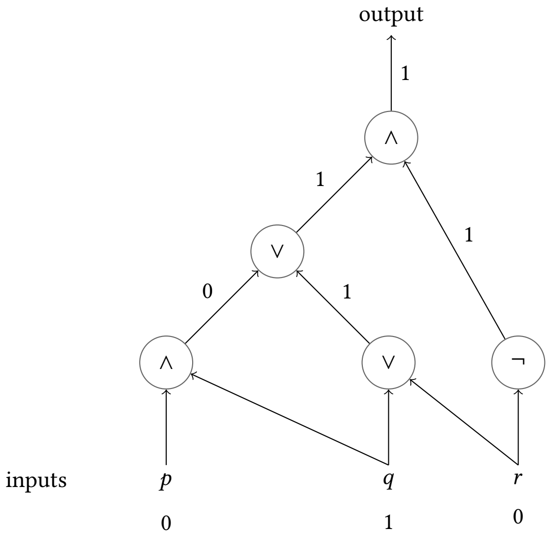


Figure 2: A simple boolean circuit, representing the logical formula $((p \wedge q) \vee (q \vee r)) \wedge (\neg r)$. Given inputs p and r false, and q true, the circuit outputs true.

In cryptographic applications, the main task is often to prove that some computation has been done correctly by one of the parties involved. The circuit SAT problem can be very useful for proving that a computation has been done correctly. A computation can namely be represented by a circuit. Proving that a computation has been done correctly then boils down to proving that one knows inputs for the circuit such that the output is true. In cryptography, we often even desire proof systems where the prover can assert that such inputs exist, without revealing any of the inputs (see Section 7 on *zero-knowledge proofs*).

For some tasks, a more natural variant of the circuit SAT problem is the *arithmetic circuit SAT* problem, where instead of boolean operations the gates represent field operations. Working with an arithmetic field instead of booleans opens the door to algebraic techniques, which often prove useful. The arithmetic circuit SAT problem then asks whether there are inputs to the circuit such that it outputs some chosen value.

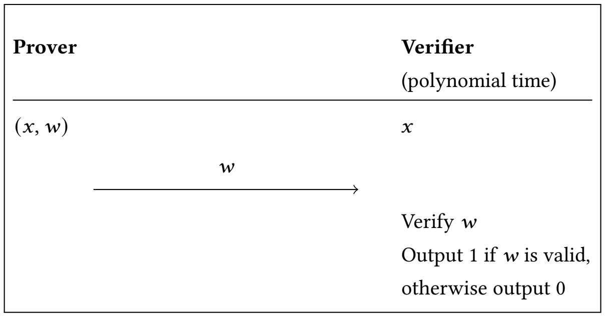
5.2 Interactivity in Proof Systems

The first design principle for proof systems is interactivity. In this section, we discuss the non-interactive $\mathcal{NP}$-proofs and $\mathcal{MA}$-proofs, where we frame the $\mathcal{NP}$-verifier (Definition 4.10) and $\mathcal{MA}$-verifier (Definition 4.13) in a different light, which can naturally be generalized to other proof systems. Then, we aim to prove membership for larger complexity classes by adding interactivity, giving rise to the complexity class $\mathcal{IP}$.

5.2.1 *$\mathcal{NP}$-Proofs and $\mathcal{MA}$-Proofs. The definition of $\mathcal{NP}$ (Definition 4.10) naturally leads to a definition of the $\mathcal{NP}$-proof system, which is a proof system that is able to prove membership of $\mathcal{NP}$-languages. The $\mathcal{NP}$-proof system is represented graphically in Figure 3. The $\mathcal{MA}$-proof system is very similar, with the exception that $\mathcal{MA}$-proof systems have PPT verifiers.

Definition 5.1 ($\mathcal{NP}$-proofs [22]). An $\mathcal{NP}$-proof system for an $\mathcal{NP}$-relation R consists of a *prover* P and a *verifier* V. The prover is given the *common input* x and a purported witness w, with $|w| \leq \text{poly}(|x|)$, and the verifier is given x. The prover sends w to V, after which V verifies w in polynomial-time in $|x|$, satisfying two conditions:

- *Completeness.* If $(x, w) \in R$, then $V(x, w) = 1$ (i.e., V accepts).
- *Soundness.* If $x \notin L(R)$, then $V(x, w') = 0$ for all w'.



Prover | **Verifier** (polynomial time)

(x, w) | x

$\xrightarrow{w}$

Verify w
Output 1 if w is valid,
otherwise output 0

Figure 3: $\mathcal{NP}$-proof for a relation R on common input x and purported witness w for x. By restricting the verifier to run in PPT, this protocol represents an $\mathcal{MA}$-proof.

By definition, the set of relations that have an $\mathcal{NP}$-proof is exactly the set $\mathcal{NP}$. We can give the verifier a bit more power by allowing them to toss random coins. This gives rise to *Merlin-Arthur games*, first introduced by Babai [7].

Definition 5.2 ($\mathcal{MA}$-proofs [7]). A *Merlin-Arthur game* or $\mathcal{MA}$-*proof system* for a relation R consists of a *prover* P and a PPT *verifier* V. The prover is given the *common input* x and a purported witness w, with $|w| \leq \text{poly}(|x|)$, and the verifier is given x. The prover sends w to V, after which V verifies w, satisfying two conditions:

- *Completeness.* If $(x, w) \in R$, then V accepts with probability at least $\frac{2}{3}$, denoted $\mathbb{P}[V(x, w; \rho) = 1] \geq \frac{2}{3}$, where ρ is a random variable representing the internal random coin tosses of V.
- *Soundness.* If $x \notin L(R)$, then for any purported witness w', we have $\mathbb{P}[V(x, w'; \rho) = 1] \leq \frac{1}{3}$.

By definition, the set of relations that have an $\mathcal{MA}$-proof is exactly the set $\mathcal{MA}$. Although these proof systems show a very basic form of interaction (where the verifier listens to the prover and verifies the information they receive from the prover), the messages only flow from prover to verifier. In the next paragraph, we introduce two-way interaction between the prover and verifier.

5.2.2 Interactive Proof Systems. In this paragraph, we extend the power of $\mathcal{NP}$-proofs and $\mathcal{MA}$-proofs by introducing (two-way) interactivity. Traditionally, a proof was always considered to be a fixed static object that is verified by reading the full proof. However, since randomized and interactive computations have emerged, it was a natural development to also consider randomized and interactive proofs, where the verifier and the prover interact dynamically [32].

The general idea of an *interactive proof system* is that a computationally unbounded *prover* wants to prove to a computationally bounded (PPT) *verifier* that some proposition is true (usually that a word is a member of some language). The verifier can send *challenges* (questions) to the prover, which the prover has to answer convincingly to assure the verifier that they actually have a witness for the truth of the proposition. There may be one or $\text{poly}(|x|)$ rounds of challenges and responses. The interactive proof system is shown in Figure 4.

To formalize the definition of an interactive proof, we first need to define the notion of a strategy and an interactive protocol.

Definition 5.3. A *strategy* is a function that describes the party's next move in the interactive proof as a function of the proposition to be proved (also called the *common input*), the party's internal random coin tosses, and all messages received so far.

Definition 5.4. A *k-round interactive protocol* is a k-round interactive game between a computationally unbounded prover and a probabilistic verifier. Each round i has the following structure:

(1) The verifier sends a message c_i to the prover.
(2) The prover responds with a message m_i, determined by the common input, all messages that the verifier has sent so far, and internal random coin tosses (determined by a random variable ρ').
(3) The verifier determines the next message, based on all messages previously sent by the prover, and internal random coin tosses (determined by a random variable ρ).

After k such rounds, the verifier outputs 1 if they accept and 0 if they reject.

We formalize the concept of an interactive proof as follows [32, Definition 1.1]. Note that we write $\{0,1\}^*$ for the set of bit-strings of any (finite) length.

Definition 5.5 (Interactive Proof System). A *k-round interactive proof system* for a relation R with *soundness error* $s\colon \{0,1\}^* \to [0,1]$ consists of a *prover* executing a computationally unbounded strategy, and a *verifier* executing a PPT strategy, satisfying two conditions:

- *Completeness.* For every common input $x \in L(R)$, there exist prover and verifier strategies P and V respectively, such that the verifier accepts with probability 1 after engaging with P in the k-round interactive protocol, denoted $\mathbb{P}[V(x, m_1, \dots, m_k; \rho) = 1] = 1$.

- *Soundness.* For every common input $x \notin L(R)$ and every prover strategy, we have $\mathbb{P}[V(x, m'_1, \dots, m'_k; \rho) = 1] \leq s(x)$.

We write $\langle P \leftrightarrow V \rangle$ for the interactive proof system between prover strategy P and verifier strategy V.

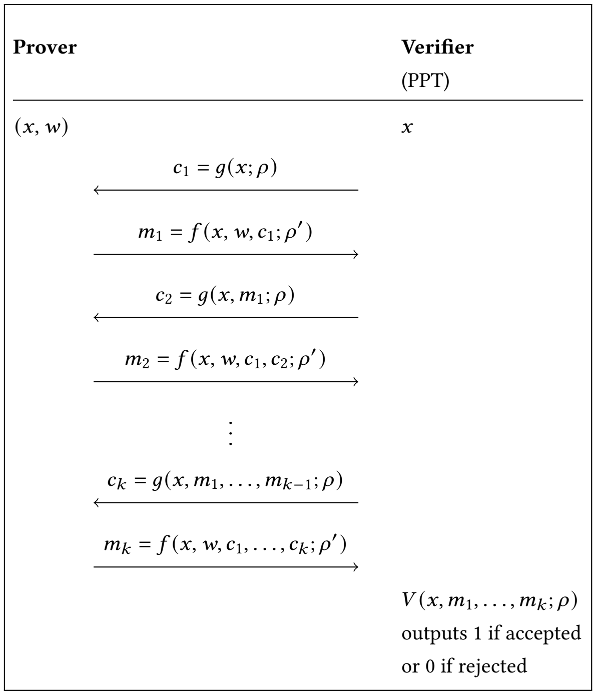

Prover
Verifier
(PPT)
(x, w)
x
$c_1 = g(x; \rho)$
$m_1 = f(x, w, c_1; \rho')$
$c_2 = g(x, m_1; \rho)$
$m_2 = f(x, w, c_1, c_2; \rho')$
$\vdots$
$c_k = g(x, m_1, \dots, m_{k-1}; \rho)$
$m_k = f(x, w, c_1, \dots, c_k; \rho')$
$V(x, m_1, \dots, m_k; \rho)$
outputs 1 if accepted
or 0 if rejected

Figure 4: A k-round interactive proof system with common input x and purported witness w.

Remark 5.6. While in the definition of proof systems we allow for arbitrary soundness error functions $s\colon \{0,1\}^* \to [0,1]$, in the constructions the soundness should be thought of as a constant, say $\frac{1}{2}$, unless explicitly stated otherwise.

Remark 5.7. Note that in the definition above we require *perfect completeness.* One can imagine introducing a parameter $c \in (0,1)$ such that for $x \in L(R)$ we have $\mathbb{P}[V(x, m_1, \dots, m_k; \rho) = 1] \geq c$. However, most proof systems in literature achieve perfect completeness ($c = 1$). In the rest of this survey, we only discuss protocols with perfect completeness.

Definition 5.8. We denote $\mathcal{IP}$ *(interactive polynomial-time)* for the set of languages for which a k-round interactive proof system exists with soundness error $\frac{1}{2}$ for each input x, and with $k = \text{poly}(|x|)$.

Note that the soundness error of $\frac{1}{2}$ is an arbitrary choice. We can always repeat the interactive proof a polynomial number of times to make the soundness error smaller than any chosen constant (or even as small as $\exp(-\text{poly}(n))$, which is important for cryptographic applications). Hence, the definition of $\mathcal{IP}$ does not depend on the choice of soundness error. Note, however, that there is a trade-off between the number of rounds (i.e., the amount of computation), and the soundness error. With more rounds (i.e., more computation), we can reduce the soundness error.

Below, we give an example of an interactive proof system that can be used to show that two graphs are *non-isomorphic* (closely following Goldreich's survey [32]). The graph non-isomorphism problem is known to be in co$\mathcal{NP}$. It is widely believed that this problem is not in $\mathcal{NP}$, as there is no clear witness for the fact that two graphs are non-isomorphic [32]. This shows that interactive proofs can be more powerful than $\mathcal{NP}$-proofs. In fact, the following result holds.

Theorem 5.9 ([46]). *The class $\mathcal{PSPACE}$* (polynomial space) *is the set of languages L for which a deterministic algorithm exists that decides L using an amount of memory that is polynomial in terms of the input size. It holds that $\mathcal{IP} = \mathcal{PSPACE}$.*

Since it is widely believed that $\mathcal{NP} \subsetneq \mathcal{PSPACE}$, this theorem shows that interactive proof systems are indeed powerful.

Before giving the interactive proof for graph non-isomorphism, we introduce notation for uniform random sampling, and we define graph isomorphism.

Notation. We write $c \overset{\$}{\leftarrow} C$ for an element c that is sampled uniformly at random from a finite set C.

Definition 5.10. We say that two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic*, denoted $G_1 \cong G_2$, if there exists a bijective function $\varphi\colon V_1 \to V_2$, such that $\{v_1, v_2\} \in E_1 \iff \{\varphi(v_1), \varphi(v_2)\} \in E_2$. We call φ an *isomorphism.*

Figure 5 presents the interactive proof system that can be used to prove that two graphs are non-isomorphic. The main idea is that the verifier challenges the prover by randomly choosing one of the graphs and then sending a *uniformly* random permutation of the selected graph. The prover then has to tell the verifier whether they received an isomorphism of G_1 or G_2. The prover can only do this reliably if the two graphs are actually non-isomorphic: if the graphs are isomorphic, then the distribution of the transmitted graph is *independent* of the graph the verifier chose, and thus from the prover's point of view, it is equally as likely that the graph they received is a permutation of G_1 or G_2.

Historically, a distinction was made between interactive proof systems where the random coin tosses are public[2] (called *Arthur-Merlin proof systems* [7]), and where the random coin tosses are private [33]. However, Goldwasser and Sipser proved that these two systems are equivalent in power in terms of the complexity classes of which they can prove membership [34]. Therefore, we will not make the distinction of public-coin and private-coin systems in this study.

We will be interested in many special cases of interactive proofs, where additional resource bounds are placed on the prover and/or the verifier, or in which we request additional properties other than completeness and soundness.

[2] If the verifier's random coin tosses are visible to the prover, we say that the random coin tosses are public.

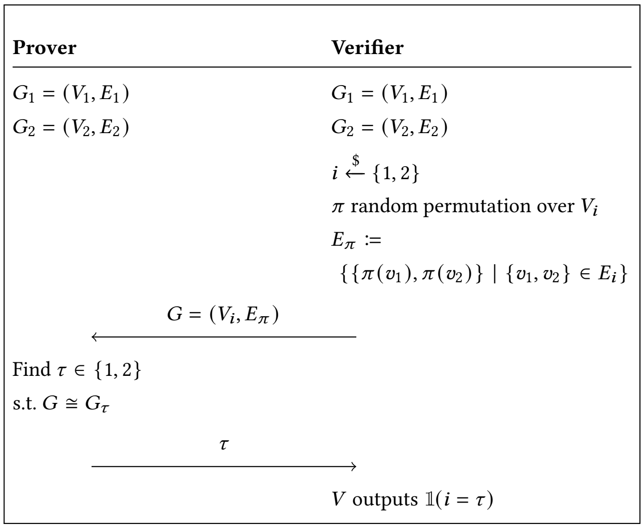


Figure 5: A 1-round interactive proof system where the prover proves that two graphs are non-isomorphic. Note that if $G \cong G_1 \cong G_2$, the prover will send $\tau \overset{\$}{\leftarrow} \{1, 2\}$ to the verifier.

5.3 Bounded Proof Access

An important design principle for proof systems is the construction of *efficient* verifiers. That is, we desire proof systems in which the communication consists of as few bits as possible. To this end, we consider proof systems in which the verifier has *bounded* access to the proof (i.e., the verifier gets an oracle with which they can query a few locations from the proof). This gives rise to *probabilistically checkable proofs* in the non-interactive branch, and *interactive oracle proofs* in the interactive branch.

Note that a *proof* should not be confused with a *witness*; a proof is the set of messages sent by the prover to the verifier, which differs from a witness for the membership of a word in a language.

5.3.1 *Probabilistically Checkable Proofs. In this paragraph, we will discuss *probabilistically checkable proofs (PCPs)*, which are non-interactive proof systems in which the verifier gets oracle access to a proof, to which they only do a few queries to determine whether the proof is valid. Essentially, this means that the verifier can determine whether a purported proof is valid by only reading a few locations from the proof. This is made possible by adding much redundancy to a proof. The PCP system is represented graphically in Figure 6.

For a PCP, we will be interested in the number of locations that need to be queried from the proof, and a measure for the amount of randomness that is involved in the proof system. This measure of randomness is closely related to the length of the proofs.[3]

We formally define PCPs as follows.

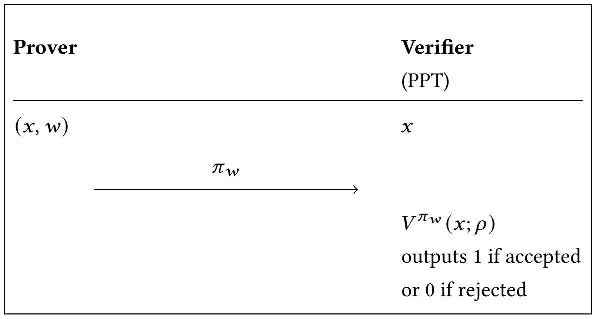


Figure 6: A probabilistically checkable proof with common input x and purported witness w.

Definition 5.11 (Probabilistically Checkable Proof [32]). A *probabilistically checkable proof system (PCP)* for a relation R with *soundness error* $s\colon \{0,1\}^* \to [0,1]$ is a *verifier* V executing a PPT strategy, satisfying two conditions:

- *Completeness.* For every $(x, w) \in R$ (where x is common input), there exists a proof oracle π_w, such that V accepts with probability 1 on input x and access to π_w. We denote this as $\mathbb{P}[V^{\pi_w}(x; \rho) = 1] = 1$, where ρ is the random variable determining the verifier's internal random coin tosses.
- *Soundness.* For all $x \notin L(R)$ and π', $\mathbb{P}[V^{\pi'}(x; \rho) = 1] \leq s(x)$.

As was the case for interactive proof systems (Definition 5.5), we can reduce the soundness error by doing extra rounds of the proving process. Hence, we have a trade-off between the soundness error and the total number of locations that are read from the proof.

Next, we define the parameters of a PCP, which we will need to state results about its power. These definitions closely follow Goldreich's survey [32].

Definition 5.12. The *randomness complexity* $r\colon \mathbb{N} \to \mathbb{N}$ of a PCP expresses the maximum number of random coin tosses by the verifier as a function of the input length.

Definition 5.13. The *query complexity* $q\colon \mathbb{N} \to \mathbb{N}$ of a PCP expresses the maximum number of oracle queries by the verifier as a function of the input length.

Definition 5.14. Let $r, q\colon \mathbb{N} \to \mathbb{N}$ be integer functions. We denote $\mathcal{PCP}(r, q)$ for the set of languages for which a PCP exists with randomness complexity $O(r)$ and query complexity $O(q)$.

PCPs with a polynomial number of random coin tosses and a polynomial number of oracle queries are able to prove membership in a complexity class that is larger than $\mathcal{NP}$ [8].

Theorem 5.15. *The complexity class $\mathcal{NEXP}$ (non-deterministic exponential time) is the set of relations R for which a deterministic algorithm exists that given an $(x, w) \in R$, with $|w| \leq 2^{\mathrm{poly}(|x|)}$, is able to verify in $2^{\mathrm{poly}(|x|)}$ steps that w is a valid witness for $x \in L(R)$. We know that $\mathcal{NP} \subsetneq \mathcal{NEXP}$, and that $\mathcal{NEXP} = \mathcal{PCP}(\mathrm{poly}(|x|), \mathrm{poly}(|x|))$.*

The power of PCPs is further expressed in the *PCP theorem*, which is one of the most celebrated theorems in computer science. The PCP theorem states that for each language in $\mathcal{NP}$, there is a

[3] Suppose the PCP can do q queries and toss r coins. There are 2^r different possible outcomes of the sequence of coin tosses. If for each of these sequences the PCP queries q unique locations of the proof, then the total number of locations that can possibly be queried is $2^r q$. Therefore, $2^r q$ is an upper bound for the effective proof length. Technically, we assume here that the verifier's queries are *non-adaptive* (i.e., the query locations only depend on the verifier's internal random coin tosses, and not on what is read from previous queries). Most PCP constructions are non-adaptive PCPs.

PCP that only needs $O(\log|x|)$ random coin tosses and a *constant number* of oracle queries [2, 3].

Theorem 5.16 (PCP theorem). $\mathcal{NP} = \mathcal{PCP}(\log|x|,\ 1)$.

At first glance, the PCP model might seem a bit unnatural. For example, how can we force a PPT verifier to only read a few locations from the proof? This might not seem like an important detail at this point, but as soon as we consider *zero-knowledge* (where the aim is to limit how much a verifier can learn from an interaction, see Section 7), it will be crucial that the verifier's access to the proof is indeed restricted. Fortunately, cryptographic techniques can indeed implement a PCP system. The basic idea is to have the prover commit to a proof string, after which the verifier responds by sending the desired query locations. The prover then reveals the query locations, together with a proof that they are indeed the values they had committed to. Note that this does cost a bit of interaction between the prover and verifier, but it is quite minimal and can furthermore be removed later via the *Fiat-Shamir transform* (see Section 5.7).

5.3.2 Interactive Oracle Proofs. In this paragraph, we discuss the *interactive oracle proof (IOP)*, which is the interactive analogue of the PCP. Historically, IOPs were designed to combine the power of PCPs (bounded proof access) with the power of interactive proofs (interactivity).

IOPs are (possibly multi-round) interactive proofs, in which the verifier gets *oracle access* to the messages of the prover (i.e., to the proof) and does not have to read them in full. The main trade-off for IOPs is between adding extra rounds and reading less bits per round. IOPs are represented schematically in Figure 7.

To formally define IOPs (and to define IOPs of proximity in Section 5.5.2), we first need to define *interactive oracle protocols*. They are very similar to the interactive protocols (see Definition 5.4), with the exception that in an interactive oracle protocol the prover sends *proof oracles* to the verifier instead of proof messages.

Definition 5.17 ([12]). A *k-round interactive oracle protocol* is a k-round interactive game between a computationally unbounded *prover* and a probabilistic *verifier*. Each round i has the following structure:

(1) The verifier sends a message c_i to the prover.
(2) The prover responds with an oracle π_i, determined by the common input, all messages that the verifier has sent so far, and internal random coin tosses (determined by a random variable ρ').
(3) The verifier determines the next message, based on the common input, all oracles previously sent by the prover, and internal random coin tosses (determined by a random variable ρ).

After k such rounds, the verifier outputs 1 if they accept and 0 if they reject.

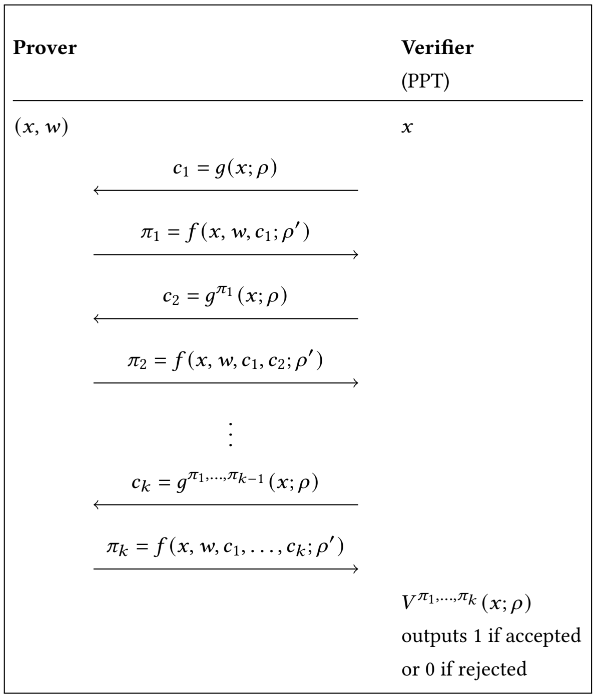

Prover
Verifier
(PPT)
(x, w)
x
$c_1 = g(x;\rho)$
$\pi_1 = f(x, w, c_1; \rho')$
$c_2 = g^{\pi_1}(x;\rho)$
$\pi_2 = f(x, w, c_1, c_2; \rho')$
$\vdots$
$c_k = g^{\pi_1,\ldots,\pi_{k-1}}(x;\rho)$
$\pi_k = f(x, w, c_1, \ldots, c_k; \rho')$
$V^{\pi_1,\ldots,\pi_k}(x;\rho)$
outputs 1 if accepted
or 0 if rejected

Figure 7: A k-round interactive oracle proof system with common input x and purported witness w.

Definition 5.18 (Interactive Oracle Proof [12]). An *interactive oracle proof* (IOP) for a relation R with *round complexity* $k\colon \{0,1\}^* \to \mathbb{N}$ and *soundness error* $s\colon \{0,1\}^* \to [0,1]$ consists of a computationally unbounded prover and a PPT verifier, satisfying two conditions:

- *Completeness.* For every $(x, w) \in R$, there exist probabilistic prover and verifier strategies P and V respectively, such that the verifier accepts with probability 1 after engaging with P in the $k(x)$-round interactive oracle protocol, denoted $\mathbb{P}[V^{\pi_1,\ldots,\pi_{k(x)}}(x;\rho)] = 1$.
- *Soundness.* If $x \notin L(R)$, then for all probabilistic prover strategies, $\mathbb{P}[V^{\pi'_1,\ldots,\pi'_{k(x)}}(x;\rho)] \le s(x)$.

Besides the round complexity and soundness error, we are also interested in the following complexity measures.

Definition 5.19. We define the *proof length* of an IOP as a function $p\colon \{0,1\}^* \to \mathbb{N}$ that maps an input x to the number of bits that the prover communicates to the verifier.

Definition 5.20. We define the *query complexity* of an IOP as a function $q\colon \{0,1\}^* \to \mathbb{N}$ that maps an input x to the number of queries that the verifier asks any of the oracles in the $k(x)$ rounds.

IOPs with *linear* proof length and *polylogarithmic* query complexity in N have been used to prove the satisfiability of arithmetic circuits consisting of N gates over any field (containing at least $O(N)$ elements). The prover runs in linear-time, and the verifier runs in polylogarithmic time. Furthermore, this IOP leaks no information about the inputs of the circuit [19].

It is an open problem whether a PCP exists that can also prove satisfiability of arithmetic circuits in linear prover time and polylogarithmic verifier time [19], which shows that IOPs could be more powerful than PCPs for such problems.

5.4 Bounded Input Access (Proximity)

In the proof systems we have seen so far, the verifier has to read the full input, which takes linear time. That means that none of these proof systems run in sublinear-time. In order to decrease the time complexity and facilitate sublinear-time verification, we give the verifier *bounded* access to the input by giving them an input oracle that can be queried, such that the verifier does not have to read the full input. In that case, the verifier can verify membership for inputs that are in the language, but can only verify non-membership for inputs that are *far* from the language.

Bounded input access (and full proof access) gives rise to *non-interactive proofs of proximity* and *interactive proofs of proximity*.

As was the case for bounded proof access (see Section 5.3.1), we mention that practical implementations of bounded input access is non-trivial as well. One way to construct a proof system with bounded input access, is by using a special data structure called a *Merkle tree* [39]. We omit the details of this implementation.

5.4.1 *Merlin-Arthur proofs of proximity.

In this paragraph, we discuss *Merlin-Arthur proofs of proximity (MAPs)*, which are non-interactive proof systems in which the verifier gets oracle access to the input, to which they only do a few queries to determine whether the input is *close* to a language. The goal of this proof system is to achieve sublinear time complexity in a non-interactive proof system, as proof systems where the input is read in full must have at least linear time complexity. Note that the verifier reads the (sublinear length) proof in full. We will be interested in the *proof length* and *query complexity* of a MAP.

We first define a distance measure on strings (the number of coordinates in which the strings differ), after which we can formally define MAPs. MAPs are represented schematically in Figure 8.

Definition 5.21. Given two strings $x, y \in \{0,1\}^n$, we define the *Hamming distance*

$$\mathrm{d}(x, y) = \frac{1}{n} \cdot |\{i \in \{1, \ldots, n\} \mid x_i \neq y_i\}|.$$

Given a string $x \in \{0,1\}^n$ and a set $L \subseteq \{0,1\}^n$, we define the distance of x to L as

$$\mathrm{d}(x, L) = \min_{y \in L} \mathrm{d}(x, y).$$

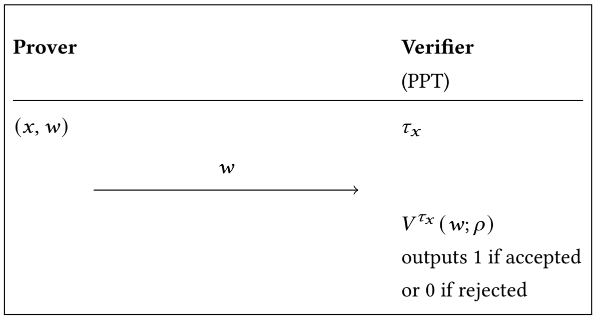


Figure 8: A Merlin-Arthur proof of proximity with input x and purported witness w. The verifier only gets oracle access to x through τ_x.

Definition 5.22 (Merlin-Arthur proofs of proximity [35])**.** A *Merlin-Arthur proof of proximity (MAP)* for a relation R with *proximity parameter* $\delta \in (0,1)$ and *soundness error* $s\colon \{0,1\}^* \to [0,1]$ is a verifier V executing a PPT strategy, satisfying two conditions:

- *Completeness.* For every $x \in L(R)$, there exists a proof w and an oracle τ_x (providing oracle access to x), such that $\mathbb{P}[V^{\tau_x}(w; \rho) = 1] = 1$, where ρ is the random variable determining the internal random coin tosses of V.
- *Soundness.* For every input x with $\mathrm{d}(x, L(R)) > \delta$, every oracle τ, and every w', we have $\mathbb{P}[V^{\tau}(w'; \rho) = 1] \leq s(x)$.

MAPs are more powerful than so-called *property testers*[4], but are less powerful than *interactive proofs of proximity* [35], which is the topic of the following paragraph. Property testers are randomized algorithms with bounded input access that aim to decide a language (with some error probability). A MAP is essentially to a property tester as $\mathcal{NP}$ is to $\mathcal{P}$: a property tester tries to decide a language, a MAP verifies purported witnesses.

5.4.2 Interactive Proofs of Proximity.

In this paragraph, we discuss the *interactive proof of proximity (IPP)*, which is the interactive counterpart of the MAP. As was the case for MAPs, the goal of IPPs is to reduce the time complexity of a proof system to sublinear time, by giving the verifier bounded input access. In that case, the verifier can verify membership for inputs that are in the language, but can only verify non-membership for inputs that are *far* from the language (as was the case for MAPs). IPPs are presented schematically in Figure 9.

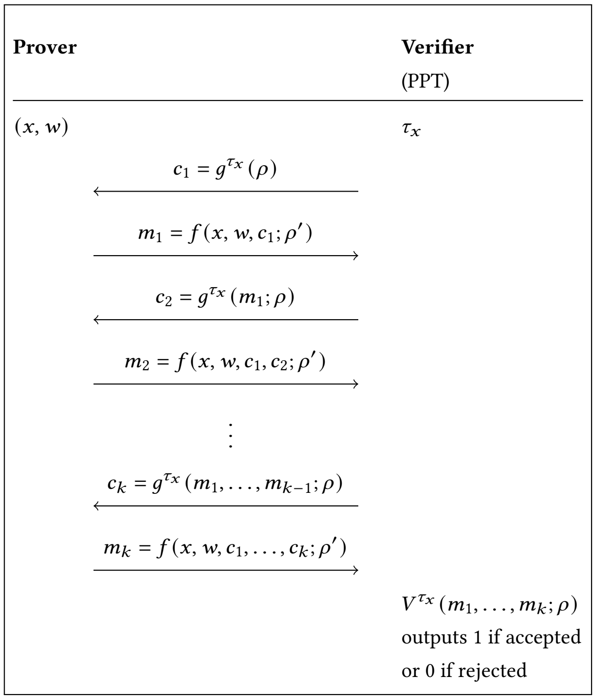

Prover
Verifier
(PPT)
(x, w)
τ_x
$c_1 = g^{\tau_x}(\rho)$
$m_1 = f(x, w, c_1; \rho')$
$c_2 = g^{\tau_x}(m_1; \rho)$
$m_2 = f(x, w, c_1, c_2; \rho')$
$\vdots$
$c_k = g^{\tau_x}(m_1, \ldots, m_{k-1}; \rho)$
$m_k = f(x, w, c_1, \ldots, c_k; \rho')$
$V^{\tau_x}(m_1, \ldots, m_k; \rho)$
outputs 1 if accepted
or 0 if rejected

Figure 9: A k-round interactive proof of proximity with input x and purported witness w. The verifier only gets oracle access to x through τ_x.

[4] Roughly, property testers are a generalization of $\mathcal{P}$, where the decider algorithm is randomized and only has bounded input access. Property testers are, however, outside the scope of this paper.

Definition 5.23 (Interactive Proof of Proximity [45]). An *interactive proof of proximity (IPP)* for a relation R with *proximity parameter* $\delta \in (0,1)$, *soundness error* $s\colon \{0,1\}^* \longrightarrow [0,1]$, and *round complexity* $k\colon \{0,1\}^* \longrightarrow \mathbb{N}$ consists of a prover P executing a computationally unbounded strategy, and a verifier V executing a PPT strategy, satisfying two conditions:

- *Completeness.* For every $x \in L(R)$, there exists an oracle τ_x (providing oracle access to x), such that the verifier accepts with probability 1 after engaging with P in the $k(x)$-round interactive protocol (with τ_x as common input), denoted $\mathbb{P}[V^{\tau_x}(m_1, \ldots, m_{k(x)}; \rho) = 1] = 1$.
- *Soundness.* For every x with $\mathrm{d}(x, L(R)) > \delta$, every oracle τ, and every prover, $\mathbb{P}[V^{\tau}(m'_1, \ldots, m'_{k(x)}; \rho) = 1] \leq s(x)$.

We define a couple of parameters that are used to analyze IPPs.

Definition 5.24. We define the *query complexity* of an IPP as a function $q\colon \{0,1\}^* \longrightarrow \mathbb{N}$ that maps an input x to the number of queries that the verifier asks the input oracle.

Definition 5.25. We define the *communication complexity* of an IPP as a function $c\colon \{0,1\}^* \longrightarrow \mathbb{N}$ that maps an input x to the total number of bits that is communicated between the prover and the verifier.

It turns out that for every language that can be decided by boolean circuits where the longest path from input to output is *small*, there exists a sublinear-time IPP with query and communication complexity $O(\sqrt{|x|})$, and with polylogarithmic round complexity [45]. In general, there is a trade-off between the query and communication complexities and the round complexity. With more rounds, we can decrease the query and communication complexities, allowing the full proof execution to have sublinear time complexity. However, for constant-round IPPs, the query or communication complexity must be polynomial, meaning we get (at least) linear time complexity and we lose the advantage of proximity [45].

5.5 Bounded Proof and Input Access

In the previous two sections, we introduced proof systems with bounded proof access (to minimize the amount of communication) and with bounded input access (to reduce the time-complexity to sublinear). In this section, we combine the two properties to construct proof systems with both bounded proof and bounded input access, such that the verification can run in sublinear time, while still reducing the amount of communication. Bounded proof and bounded input access gives rise to *probabilistically checkable proofs of proximity* in the non-interactive branch, and *interactive oracle proofs of proximity* in the interactive branch.

5.5.1 *Probabilistically Checkable Proofs of Proximity. In this section, we introduce *probabilistically checkable proofs of proximity (PCPPs)*, which are non-interactive proof systems in which the verifier gets oracle access to the proof and to the input. The goal is to obtain a proof system with as little communication as possible that runs in sublinear time. To achieve this, we need to relax our verification power, accepting that we can only prove non-membership for inputs that are far from a language (as we did in Section 5.4). PCPPs are presented schematically in Figure 10.

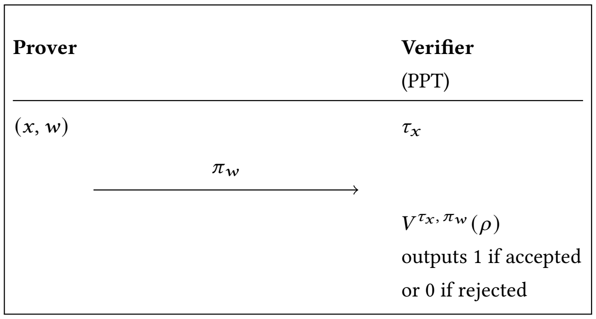


Figure 10: A probabilistically checkable proof of proximity with input x and purported witness w. The verifier only gets oracle access to x through τ_x.

Definition 5.26 (Probabilistically Checkable Proof of Proximity [14]). A *probabilistically checkable proof of proximity (PCPP)* for a relation R with *soundness error* $s\colon \{0,1\}^* \longrightarrow [0,1]$ and *proximity parameter* $\delta \in (0,1)$ is a *verifier* V executing a PPT strategy, satisfying two conditions:

- *Completeness.* For every input $(x, w) \in R$, there exist two oracles τ_x (providing oracle access to x) and π_w, such that $\mathbb{P}[V^{\tau_x, \pi_w}(\rho) = 1] = 1$, where ρ is the random variable for the internal random coin tosses of V.
- *Soundness.* For every x with $\mathrm{d}(x, L(R)) > \delta$ and every pair of oracles τ and π, $\mathbb{P}[V^{\tau, \pi}(\rho) = 1] \leq s(x)$.

The randomness complexity of a PCPP is the same as that of a PCP (see Definition 5.12). The query complexity is also very similar (Definition 5.13), with the only difference that queries to both oracles are counted towards the query complexity of a PCPP.

As stated above, PCPPs are powerful in the sense that they allow proofs to be done in sublinear time. In fact, any $R \in \mathcal{NP}$ has a sublinear (polylogarithmic) time PCPP with logarithmic randomness complexity and constant query complexity [13, 24].

5.5.2 Interactive Oracle Proofs of Proximity. Finally, we introduce the *interactive oracle proof of proximity (IOPP)*, which is the interactive counterpart of the PCPP. Similar to PCPPs, the goal is to design an interactive proof system that runs in sublinear time with as little communication as possible. To this end, the verifier is given a proof oracle and an input oracle. IOPPs are presented schematically in Figure 11.

Definition 5.27 (Interactive Oracle Proof of Proximity [11]). An *interactive oracle proof of proximity (IOPP)* for a relation R with *proximity parameter* $\delta \in (0,1)$, *soundness error* $s\colon \{0,1\}^* \longrightarrow [0,1]$, and *round complexity* $k\colon \{0,1\}^* \longrightarrow \mathbb{N}$ consists of a prover and a PPT verifier, satisfying two conditions:

- *Completeness.* For every $(x, w) \in R$, there exists an oracle τ_x (providing oracle access to x), such that for the verifier in the $k(x)$-round interactive oracle protocol we have $\mathbb{P}[V^{\tau_x, \pi_1, \ldots, \pi_{k(x)}}(\rho) = 1] = 1$.
- *Soundness.* For every x with $\mathrm{d}(x, L(R)) > \delta$, every oracle τ, and every prover, $\mathbb{P}[V^{\tau, \pi'_1, \ldots, \pi'_{k(x)}}(\rho) = 1] \leq s(x)$.

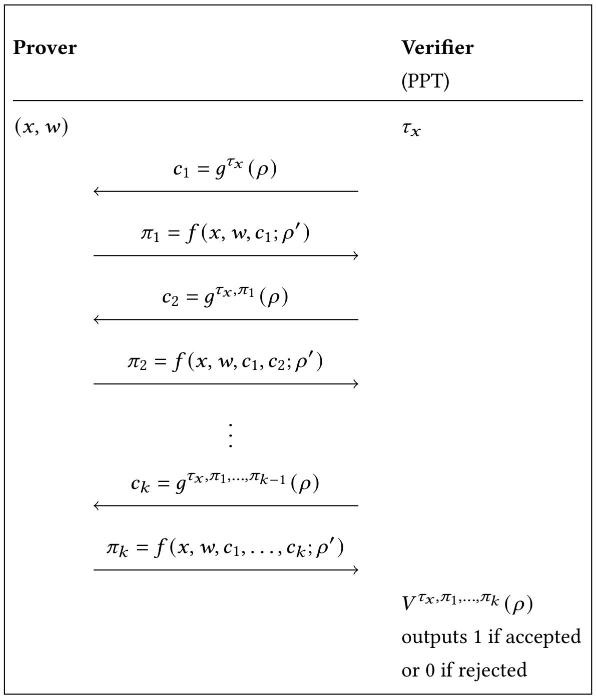

Prover
Verifier
(PPT)
(x, w)
τ_x
$c_1 = g^{\tau_x}(\rho)$
$\pi_1 = f(x, w, c_1; \rho')$
$c_2 = g^{\tau_x, \pi_1}(\rho)$
$\pi_2 = f(x, w, c_1, c_2; \rho')$
$\vdots$
$c_k = g^{\tau_x, \pi_1, \ldots, \pi_{k-1}}(\rho)$
$\pi_k = f(x, w, c_1, \ldots, c_k; \rho')$
$V^{\tau_x, \pi_1, \ldots, \pi_k}(\rho)$
outputs 1 if accepted
or 0 if rejected

Figure 11: A k-round interactive oracle proof of proximity with input x and purported witness w. The verifier only gets oracle access to x through τ_x.

An important construction of an IOPP is the *fast Reed-Solomon IOPP (FRI)*, which is an IOPP that can prove in linear time whether a vector is close to a codeword in a *Reed-Solomon code*[5] over an 'FFT-friendly' finite field (i.e., a finite field with some nice properties that open the door to the *Fast Fourier Transform*). The query complexity is logarithmic and the soundness is constant, which beats PCPPs for Reed-Solomon codes that had super-linear proving time for polynomial query complexity [9]. This major improvement is made possible by IOPPs. The FRI was later generalized to an IOPP for *algebraic geometry codes* [20], and to an IOPP for all (sufficiently large) finite fields [10].

5.6 Proving Soundness: extractor analysis

Throughout this section, we have seen that there are two properties that need to hold for each proof system, namely *completeness* (all propositions from a predefined set of true statements have a proof), and *soundness* (a malicious prover only has a small probability of convincing the verifier that a false statement is true). Completeness is usually easy to prove, as it mostly involves tying together the different steps of the proof system and applying some calculations. We will see an example in Section 7 for which completeness is indeed easy to prove (Example 7.15). Soundness, however, is much harder to prove when only working with the definition. Therefore, *extractor analysis* has been introduced to be able to prove stronger statements that are easier to address than soundness, but imply soundness. In this section, we discuss three extractor analysis techniques, namely *proofs of knowledge, special soundness (statistical soundness)* and *arguments of knowledge (computational soundness).*

5.6.1 Statistical soundness: proof of knowledge.

Statistical soundness is the definition of soundness we have used in most of the proof systems above: a proof system for a relation R is *statistically sound* if for every $x \notin L(R)$ and every (possibly malicious and computationally unbounded) prover P', the probability that the verifier accepts is bounded by the *soundness error.*

Proofs of knowledge give us a stronger condition that implies statistical soundness. A *proof of knowledge* is essentially a proof in which the prover convinces the verifier that they have *knowledge* of some property (i.e., the prover proves that they have a witness for some $x \in L(R)$, instead of just proving that such a witness exists). To formally define a proof of knowledge, we first need to define what it means to know something. In the context of a proof of knowledge, we say intuitively that a prover *knows* an object if that object can be outputted by an efficient algorithm that uses that prover as a black-box. Such an algorithm is called a *knowledge extractor* and must satisfy the following property[6]: if P' is a (possibly malicious and computationally unbounded) prover for which V accepts with high probability after interacting with P' on input x, then there exists a probabilistic *knowledge extractor E* with oracle access to P' that outputs a witness w for x in expected polynomial time[7] [31, 32].

Note that if the knowledge extractor is able to output a witness w for x, then indeed $x \in L(R)$ (otherwise such a witness would not exist). Hence, if no knowledge extractor exists and $x \notin L(R)$, then for every (possibly malicious and computationally unbounded) prover P', the verifier will only accept with small probability. In other words, proof of knowledge (or *knowledge soundness*) implies statistical soundness. This implication is useful, as it is sometimes easier to construct a knowledge extractor, than to prove directly that a proof system is statistically sound.

5.6.2 Statistical soundness: special soundness.

Similar to (but slightly simpler than) knowledge extractors is the concept of *special soundness.* A 3-message (public-coin) proof system for a relation R has the *special soundness* property if, given two distinct accepting proof transcripts with the same first message (m, c_1, z_1) and (m, c_2, z_2) for the same input x, there exists a PPT *extractor E* that outputs a witness w for x [23]. Note that we assume here that in the 3-message proof system the prover first sends a message, after which the verifier responds with a challenge, to which the prover sends an answer. Such proof systems are called Σ-protocols and are discussed further in Section 7.4.

We see that for a proof system that has special soundness, if $x \notin L(R)$ (i.e., if there exists no witness for x), intuitively there exists at most one challenge c that the verifier can send to the prover while still being convinced that $x \in L(R)$. If there were multiple such

[5] A *Reed-Solomon code* is a type of *error-correcting codes*, which are ways to encode strings into larger codewords, such that bit-flips in the codewords can be detected and corrected during their decoding.

[6] Although in practice proofs of knowledge are mostly studied in the context of efficient provers (an unbounded prover may find a witness through an exhaustive search), we define proofs of knowledge for general provers, as the definition does not depend on the runtime of the prover.

[7] A probabilistic algorithm runs in *expected* polynomial time if the expectation of the runtime (taken over the randomness of the algorithm) is polynomial in the input size.

challenges, then special soundness would not hold. Hence, special soundness implies statistical soundness with soundness error $\frac{1}{|C|}$, where C is the (finite) set of possible challenges the verifier can send. Special soundness even implies knowledge soundness, which we introduced in the previous section. Note that it requires much work to formalize this intuition.

The implication from special soundness to statistical soundness is extremely useful, as special soundness can be easier to prove than statistical soundness: one has to define an extractor that extracts witnesses from two proof transcripts. This task can be much easier than proving soundness from the definition. In Section 7, we will discuss an example where it is easy to prove special soundness (Example 7.15).

Now, special soundness can be generalized as follows. Let $\mu \in \mathbb{N}$ and $k_1, \ldots, k_\mu \in \mathbb{N}$ be given. A $(2\mu + 1)$-message public-coin proof system for a relation R has the $(k_1, \ldots, k_\mu)$*-special soundness* property if, given a tree of distinct accepting proof transcripts as shown in Figure 12, there exists a PPT *extractor* E that outputs a witness w for x [5].

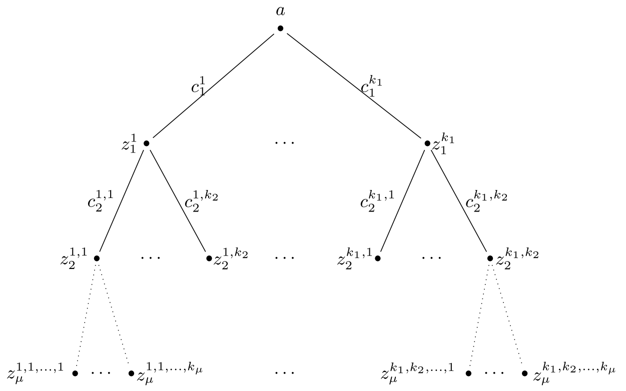


Figure 12: This tree represents distinct proof transcripts for the same input in a $(2\mu + 1)$-round public-coin proof system. Namely, each path from the root to a leaf represents a single proof transcript, where each node on the path is a prover message, and each edge on the path is a verifier message. *Reprinted from Attema, Cramer, and Kohl [5].*

Note that special soundness as defined above is a special case of $(k_1, \ldots, k_\mu)$-special soundness. Namely, special soundness is 2-special soundness.

As was the case for special soundness, $(k_1, \ldots, k_\mu)$-special soundness also implies statistical soundness with a certain soundness error [5, Theorem 1].

5.6.3 Computational soundness: argument of knowledge.

We can relax the notion of statistical soundness by requiring only that it be *computationally unfeasible* for a prover to convince a verifier that a false statement is true (whereas for statistical soundness, we do not take the computational power of the prover into account). We call this relaxation *computational soundness* [31]. Thus, a proof system for a relation R is computationally sound if for every $x \notin L(R)$ *and every PPT prover* P', the probability that the verifier accepts after interacting with P' on input x is small. In this context, we replace the term *proof of knowledge* by *argument of knowledge*. We use *argument* instead of *proof* to highlight the fact that we require a less strong notion of soundness: essentially, arguments are computationally sound proofs. As we have weakened the prover, it can be easier to prove computational soundness than statistical soundness. Nonetheless, computational soundness is sufficient for many applications [31].

5.7 Fiat-Shamir Transform: From interactive to non-interactive

Throughout this survey, we kept the interactive branch separated from the non-interactive branch of proof systems. We saw that interaction allows us to challenge the prover, thereby increasing the power of proof systems, allowing us to test more concepts (e.g., Figure 5 presented an interactive protocol for testing graph non-isomorphism, which is a relation that is not believed to be in $\mathcal{NP}$). Interactivity can, however, be costly in terms of added communication complexity. Furthermore, interactivity can be problematic in terms of latency, as both the prover and the verifier need to be online during the entire proof. The *Fiat-Shamir Transform* [25] is a method that translates interactive proofs into non-interactive proofs, thereby reducing the communication complexity while preserving the power of interactivity.

Any interactive public-coin interactive proof can be translated into a non-interactive proof by applying the *Fiat-Shamir Transform*. Intuitively, this follows from the fact that in the public-coin setting the prover can see the tape of random coin tosses of the verifier. Therefore, instead of receiving a challenge from the verifier, the prover can execute the computation that the verifier would have done to compute the challenge.

To understand the Fiat-Shamir transform, we need to define *cryptographic hash functions*.

Definition 5.28 ([47]). A *cryptographic hash function* is a function $H\colon \{0,1\}^* \to \{0,1\}^n$ for some $n \in \mathbb{N}$ that satisfies the following properties:

- *One-way.* Given an $x \in \{0,1\}^*$, it is easy to compute $H(x)$. However, given $H(x)$ it is hard to find x.
- *Collision-resistant.* It is hard to find distinct x_1, $x_2 \in \{0,1\}^*$ such that $H(x_1) = H(x_2)$.

Remark. The word 'hard' in the above definition could be made more precise, but for our discussion this informal definition suffices.

The idea is that a cryptographic hash function produces *hashes* for their inputs that look like random bit-strings.

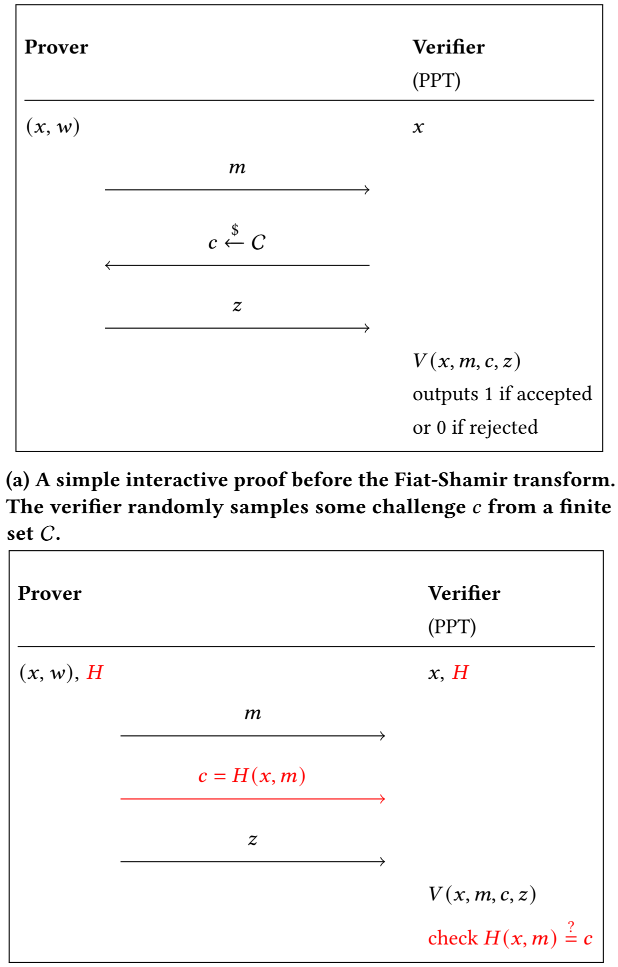
The properties of the Fiat-Shamir transform depend on the *Random Oracle Model (ROM)*. Currently, there is no consensus on the validity of the ROM.

Definition 5.29. In the *Random Oracle Model (ROM)*, all parties have access to a *perfect* hash function $H\colon \{0,1\}^* \to \{0,1\}^n$, meaning:

- for distinct $x_1, x_2 \in \{0,1\}^*$, $H(x_1)$ and $H(x_2)$ are independent and uniformly random;
- H is a function (i.e., it produces the same output for the same input);

- H is interpreted as a black-box random oracle (i.e., we have no knowledge of the internal state of H).

The interaction in an interactive proof consists of random challenges that the verifier poses to the prover, to which the prover gives convincing answers. The main idea of the Fiat-Shamir transform is to replace these random challenges by hashes of the prover's messages. That means the prover will act as if they receive a challenge c from the verifier, whereas in reality c is a hash of the prover's previous message(s). Any verifier can check the proof by calculating the hashes of the prover's messages and checking whether they agree with the challenges calculated by the prover. If the hash function is modeled as a random oracle, the transform will be sound and complete (assuming the interactive proof system is sound and complete). After the transform, all messages go in one direction (from prover to verifier), and thus can be sent as one large message. The Fiat-Shamir transform is shown schematically in Figure 13.



(a) A simple interactive proof before the Fiat-Shamir transform. The verifier randomly samples some challenge c from a finite set C.

(b) The result of applying the Fiat-Shamir transform to a simple interactive proof. Differences with the interactive proof are highlighted in red.

Figure 13: The Fiat-Shamir transform.

6 TAXONOMY

In this section, we discuss the main contribution of this survey, namely a taxonomy of the structure of currently studied proof systems. Additional to the structure of proof systems, a special property called *zero-knowledge* is studied in the literature as well. This property says, roughly, that a prover does not reveal any information beyond the fact that a statement is true during the proof process. As this property does not depend on the underlying proof system, we omit zero-knowledge from our taxonomy and discuss it separately in Section 7.

We present our taxonomy of proof systems that are currently studied in Figure 14. The structure of Section 5 already revealed most parts of the taxonomy; as a reminder to the reader, we discuss the main ideas again in this section. Our taxonomy consists of two main branches, namely *non-interactive* proof systems (where messages only go from prover to verifier), and *interactive* proof systems. Both branches contain four proof systems. A proof system where the verifier has full proof access[8] and full input access, one where they have bounded proof access (BPA) and full input access, one where they have full proof access and bounded input access (BIA), and one where they have both BPA and BIA.

For the non-interactive branch, we discussed $\mathcal{NP}$-proofs (Section 5.2.1), in which the verifier has full proof and input access. We grouped $\mathcal{MA}$-proofs under the same category as $\mathcal{NP}$-proofs, as it is widely believed that $\mathcal{NP} = \mathcal{MA}$, and as they have the same structure, with the only difference that the verifier in an $\mathcal{MA}$-proof is probabilistic, while they are deterministic in $\mathcal{NP}$-proofs. By bounding the proof access of the verifier, we get PCPs (Section 5.3.1). Full proof access and bounded input access gives us MAPs (Section 5.4.1), and both bounded proof and bounded input access gives us PCPPs (Section 5.5.1).

Similarly, in the non-interactive branch, we discussed $\mathcal{IP}$-proofs (Section 5.2.2) where verifiers have full proof and input access. We discussed IOPs (Section 5.3.2) for bounded proof and full input access, we studied IPPs (Section 5.4.2) for full proof and bounded input access, and we discussed IOPPs (Section 5.5.2) for both bounded proof and bounded input access.

7 ZERO-KNOWLEDGE

Fascinatingly, proof systems exist that do not leak more information about a proposition than the fact that it is true. Such proofs are called *zero-knowledge proofs*. They have numerous applications. For example, zero-knowledge proofs can allow inspectors to confirm whether an object is a nuclear weapon, without gaining knowledge about the internal workings of the possible weapon [27]. This application is particularly useful for nuclear disarmament. Zero-knowledge proofs are also desirable in cryptographic settings. They can be used for authentication schemes, where users prove their identity without revealing any information [26]. Finally, zero-knowledge proofs can be applied to blockchains: for example, a user may wish to prove they have sufficient funds for a purchase, without revealing their account balance [21, 41].

However, the concept of a zero-knowledge proof feels somehow paradoxical: how can we prove a statement without revealing any

[8] For proof systems consisting of multiple rounds, proof access means access to the prover's messages.

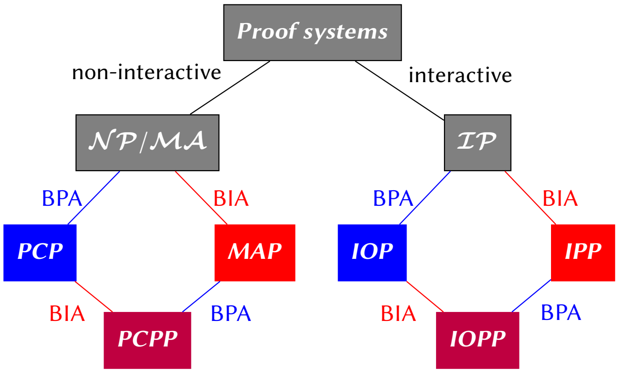


Figure 14: A taxonomy of proof systems. BPA stands for *bounded proof access*. BIA stands for *bounded input access*. Purple represents proof systems that have both BPA and BIA.

information? We saw a zero-knowledge proof in Example 7.15, although the zero-knowledge property might feel a bit abstract in that example. The following toy-example, inspired by the *Zero-Knowledge Proofs MOOC* [17], gives a sense of how zero-knowledge could be possible.

Example 7.1. Suppose Paul has two candies and wants to prove to Vanessa that the candies have different colors, without revealing the colors of those candies. Paul gets two non-transparent cups, hides each candy under one of the cups, and puts the cups in front of Vanessa. Paul looks away, after which Vanessa tosses a random coin. If the coin lands on heads, she switches the cups. Otherwise, she leaves the cups as they are. After she is finished, Paul will take a peak under the cups and tell Vanessa whether she switched the cups or not.

We see that if the two candies indeed have different colors, Paul will know with certainty whether Vanessa switched the cups. If, however, the candies have the same color, Paul has a probability of $\frac{1}{2}$ of guessing correctly whether the cups were switched. If Paul and Vanessa repeat this game k times, the probability that Paul guesses correctly each time drops to $\frac{1}{2^k}$.

This example shows that Vanessa can be convinced that Paul indeed has two different colored candies, without learning the colors of the candies. Furthermore, Paul can trick Vanessa with only a small probability as the number of rounds increases.

To formally define zero-knowledge proofs, we need a definition of what it means for the verifier to learn nothing beyond the fact that the assertion is true. We follow the simulation paradigm as described in Goldreich's book *Foundations of Cryptography* [31, Chapter 4]. Intuitively, the verifier learns nothing if what they can compute before the interaction is the same as what they can compute after the interaction. This is certainly true if the verifier can simulate each possible proof transcript without interacting with the prover: anything they could compute after the interaction can be computed without talking to the prover by generating a transcript using the simulator.

We model the simulation by comparing two random variables, namely the *proof transcript random variable* (depending on the common input and witness), and a *simulator random variable* (depending only on the verifier and common input). If these random variables are *close* to each other, the proof system is *zero-knowledge.* There are multiple ways to define this closeness, which gives rise to different definitions of zero-knowledge. In the following subsections, we discuss three definitions of zero-knowledge. In Section 7.1, we discuss *perfect zero-knowledge*, where we require the two random variables to be *identically distributed.* In Section 7.2, we discuss *statistical zero-knowledge*, where the random variables have to be indistinguishable by *any* algorithm, and we discuss *computational zero-knowledge*, which is similar to statistical zero-knowledge, except for the fact that the algorithm has to be PPT. Then, in Section 7.3 we apply these definitions to the proof systems that have been discussed in Section 5 and discuss their power. Finally, in Section 7.4 we define Σ*-protocols*, which is a special class of zero-knowledge proofs, consisting of three messages.

First, we define the three definitions of closeness for arbitrary random variables.

Definition 7.2. Let X and Y be random variables. We say that X and Y are *identically distributed*, denoted $X \equiv Y$, if they have the same probability distribution.

Definition 7.3. Let $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ be families of random variables. We say that X and Y are *statistically indistinguishable*, denoted $X \approx_s Y$, if given any algorithm $\mathcal{A}$, any polynomial p, and sufficiently large $\lambda \in \mathbb{N}$, we have

$$|\mathbb{P}[\mathcal{A}(X_\lambda) = 1] - \mathbb{P}[\mathcal{A}(Y_\lambda) = 1]| \leq \frac{1}{p(\lambda)}.$$

Note that $\mathcal{A}$ can be interpreted as an algorithm that tries to distinguish two distributions. The above definition says that any such algorithm can only distinguish between the random variables with very small probability.

The definition of *computational indistinguishability* is very similar, with the exception that $\mathcal{A}$ is required to be a PPT algorithm.

Definition 7.4. Let $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ be families of random variables. We say that X and Y are *computationally indistinguishable*, denoted $X \approx_c Y$, if given a *PPT* algorithm $\mathcal{A}$, any polynomial p, and sufficiently large $\lambda \in \mathbb{N}$, we have

$$|\mathbb{P}[\mathcal{A}(X_\lambda) = 1] - \mathbb{P}[\mathcal{A}(Y_\lambda) = 1]| \leq \frac{1}{p(\lambda)}.$$

Finally, we define the *proof transcript* and *simulator* random variables that need to be compared when analyzing zero-knowledge.

Definition 7.5. Let P and V be a prover and verifier in a proof system.

- The *proof transcript* $\Pi\langle P \leftrightarrow V\rangle(x; w)$ is a random variable over proof transcripts that depends on the common input x, a witness w, and the internal random coin tosses of P and V.
- The *simulator* or *simulated transcript* $\texttt{Sim}(x)$ is a random variable that only depends on the common input x, and the internal random coin tosses of V.

Note that the sample space of these random variables consists of possible transcripts of the proof system (including the outcome of the coin tosses).

7.1 Perfect Zero-knowledge

Formally, perfect zero-knowledge for a verifier that adheres to the underlying proof system is defined as follows.

Definition 7.6 (Honest-verifier perfect zero-knowledge [31]). A prover in a proof system $\langle P \leftrightarrow V \rangle$ for a relation R is *honest-verifier perfect zero-knowledge* if there exists a PPT simulator Sim such that for each $(x, w) \in R$, we have

$$\Pi\langle P \leftrightarrow V \rangle(x; w) \equiv \mathtt{Sim}(x)$$

The definition above assumes that the verifier is honest (i.e., that the verifier adheres to the rules of the proof system). We can generalize this definition to include all PPT verifier strategies as follows.

Definition 7.7 (Malicious-verifier perfect zero-knowledge [31]). A prover in a proof system $\langle P \leftrightarrow V \rangle$ for a relation R is *malicious-verifier perfect zero-knowledge* if for every PPT verifier strategy V' and $(x, w) \in R$, there exists a PPT simulator Sim such that

$$\Pi\langle P \leftrightarrow V' \rangle(x; w) \equiv \mathtt{Sim}(x, V').$$

7.1.1 Non-interactive Perfect Zero-knowledge.

We give a definition of a special type of zero-knowledge proof, namely *non-interactive* zero-knowledge proofs. We follow Goldreich's survey on zero-knowledge proofs [30], although non-interactive zero-knowledge was originally introduced by Blum, Feldman, and Micali [16]. We follow the so-called *common reference string model*, which assumes the existence of a trusted third-party that provides a reference string following a publicly known distribution to the prover and verifier.

Definition 7.8 (Non-interactive zero-knowledge). A *non-interactive zero-knowledge proof* consists of a prover strategy P, a verifier strategy V, and a reference string s (provided by a trusted third-party). P and V can read s and do additional random coin tosses. P can send one message m to V, after which V either accepts or rejects the proof. Such a model is *zero-knowledge* if the following two random variables are identically distributed:

- *Proof transcript.* (m, s) is a random variable consisting of the proof message and the reference string.
- *Simulator.* The simulator $\mathtt{Sim}(s)$ is a random variable that only depends on the reference string s.

It turns out that by sharing a common, short, random string, any interactive zero-knowledge proof can be replaced by a non-interactive zero-knowledge proof [16]. Furthermore, the Fiat-Shamir transform can be proved secure in the Random Oracle Model [43]. Specifically, the Fiat-Shamir transform preserves zero-knowledge (i.e., if the original proof system is zero-knowledge, then the transform will also be zero-knowledge), assuming the hash function does not leak any information.

7.2 Statistical and Computational Zero-knowledge

Formally, statistical and computational zero-knowledge for a verifier that adheres to the underlying proof system are defined as follows.

Definition 7.9 (Honest-verifier (statistical/computational) zero–knowledge [31]). A prover in a proof system $\langle P \leftrightarrow V \rangle$ for a relation R is *honest-verifier statistical zero-knowledge* if there exists a PPT simulator Sim such that for each $(x, w) \in R$, we have

$$\Pi\langle P \leftrightarrow V \rangle(x; w) \approx_s \mathtt{Sim}(x)$$

Similarly, such a prover is *honest-verifier computational zero-knowledge* if

$$\Pi\langle P \leftrightarrow V \rangle(x; w) \approx_c \mathtt{Sim}(x)$$

The definition above assumes that the verifier is honest (i.e., that the verifier adheres to the rules of the proof system). We can generalize this definition to cheating verifiers (compare Definition 7.6 and Definition 7.7), but we omit this to avoid repetitiveness.

In practice, perfect zero-knowledge is hard to achieve, but statistical and computational zero-knowledge suffice for most (cryptographic) applications. Note that, in particular, computational zero-knowledge is the most used definition in literature, as it is the easiest to construct.

The set $\mathcal{CZK}$ consists of all relations for which a computational zero-knowledge interactive proof system exists. The set of relations for which a statistical zero-knowledge interactive proof system exists is called $\mathcal{SZK}$. It has been proven that $\mathcal{SZK}$ is contained in a complexity class[9] which is not believed to contain $\mathcal{NP}$. That means, for example, that there is no statistical zero-knowledge interactive proof for 3-SAT[10]. The following theorem shows that we can prove membership of far more relations with (mild) cryptographic assumptions, namely by assuming the existence of *one-way functions*. Roughly speaking, a one-way function is a function that is easy to evaluate, but hard to invert. Although there is no rigorous proof for the existence of one-way functions, it is widely believed that they exist. In fact, much of cryptography is based on their existence.

Theorem 7.10 ([32]). *If non-uniformly hard*[11] *one-way functions exist, then* $\mathcal{CZK} = \mathcal{IP}$.

7.3 Zero-knowledge proof systems

In this section, we discuss some results on explicit zero-knowledge proof systems.

7.3.1 Zero-knowledge proof of proximity.

First, we can combine interactive proofs of proximity with zero-knowledge proofs to construct *zero-knowledge proofs of proximity (ZKPP)* [15]. In essence, a ZKPP is an IPP that does not leak any information beyond the fact that the input is in the language (or far from the language) and the few locations that are read from the input.

The power of ZKPPs is expressed in the following theorem [15, Corollary 6.10]. Note that a *collision-resistant* hash function is essentially a cryptographic hash function without necessarily having the one-way property (see Definition 5.28).

[9] We know $\mathcal{SZK} \subseteq \mathcal{AM} \cap \text{co}\mathcal{AM}$, and it is believed that $\mathcal{NP} \not\subseteq \mathcal{AM} \cap \text{co}\mathcal{AM}$. The class $\mathcal{AM}$ consists of the relations for which a public-coin 1-round interactive proof exists. The class co$\mathcal{AM}$ consists of the relations whose complement is in $\mathcal{AM}$.

[10] Given a logical formula that is a finite conjunction of disjunctions that contain three literals: $(a_1 \vee b_1 \vee c_1) \wedge \cdots \wedge (a_n \vee b_n \vee c_n)$, the 3-SAT problem asks whether an assignment of the variables in the formula exists such that the formula evaluates to true. The relation consisting of such formulas with satisfying assignments is in $\mathcal{NP}$.

[11] A one-way function is *non-uniformly hard* if it is hard to invert, even for a non-uniform algorithm. That is, even for probabilistic polynomial-time algorithms whose behavior may be different for different input sizes.

Theorem 7.11. *If collision-resistant hash functions exist, then for every $\mathcal{NP}$-relation R and every $\delta \in (0,1)$, there exists a constant-round statistical zero-knowledge proof of proximity with proximity parameter δ, with a polylogarithmic time verifier, and with a polynomial time prover that proves membership in R.*

Hence, we see that for any $\mathcal{NP}$-relation R we can construct an interactive proof system that verifies that a word is close to $L(R)$ in sublinear time without revealing any additional information. Although this shows that ZKPPs can be valuable, there exist languages for which an IPP, but no ZKPP, exists [15, Theorem 5.1].

7.3.2 Zero-Knowledge Probabilistically Checkable Proofs.

Zero-knowledge PCPs (ZKPCPs) are essentially PCPs that do not leak any information. They were first introduced by Kilian, Petrank, and Tardos [40], and were later revisited by Ishai, Mahmoody, and Sahai [37].

The power of ZKPCPs is expressed in the following theorem.

Theorem 7.12 ([40])**.** *Let $Q = \text{poly}(|x|)$ be an upper bound for the number of queries that a (malicious) verifier can make to its proof oracle. For every $\mathcal{NP}$-relation R, there exists a statistical ZKPCP for which the oracle size is polynomial in the input size and the query complexity is polylogarithmic. However, this ZKPCP is only zero-knowledge against (malicious) verifiers making at most Q queries.*

Note that the above theorem implies that polynomial-time malicious verifiers making more than Q queries could potentially gain information from the ZKPCP. In fact, it has been proven that such verifiers exist if a widely believed conjecture is true.[12] [37].

7.3.3 Zero-Knowledge Probabilistically Checkable Proofs of Proximity.

Zero-knowledge PCPPs (ZKPCPPs) are PCPPs that do not leak any information. The power of ZKPCPPs is expressed in the following theorem.

Theorem 7.13 ([38])**.** *Every $\mathcal{NP}$-relation has a statistical ZKPCPP with polynomial proof length and polylogarithmic query complexity.*

7.3.4 Zero-Knowledge Interactive Oracle Proofs.

Similar to zero-knowledge for the proof systems discussed above, we can define *zero-knowledge interactive oracle proofs (ZKIOPs)* as IOPs for which the proof transcript can be simulated by a polynomial-time simulator [12].

ZKIOPs can be used in the Random Oracle Model to obtain zero-knowledge proofs with quasilinear[13] prover time and polylogarithmic verifier time [12].

It is noteworthy to mention that of all proof systems we discussed, IOPPs are the only proof systems for which no zero-knowledge variant has been studied. Our taxonomy therefore shines light upon systems that have not been studied before. For future research, it would be interesting to study whether zero-knowledge IOPPs could be useful. Note, however, that methods have also been studied that give us zero-knowledge, independently of the underlying proof system [39]. The basic idea is that at the beginning of a proof, the common input is ‘randomized’ (in some sense encrypted) in a clever way, after which the regular proof systems is executed on the randomized input. That way, we get zero-knowledge independently of the underlying proof system. It seems that in a lot of PCP and IOP(P) constructions, this method of obtaining zero-knowledge is employed, which could explain why zero-knowledge IOPPs have not been studied explicitly. It would, however, still be interesting to study zero-knowledge IOPPs explicitly, in the hope of finding new applications of them.

7.4 Σ-Protocols

We often encounter zero-knowledge interactive proofs consisting of three messages. Therefore, such proofs are given a special name, namely Σ-protocols. Σ-protocols are a special class of interactive zero-knowledge proofs, formally defined as follows.

Definition 7.14 (Σ-protocol [36])**.** Let R be a relation such that each $x \in L(R)$ has a witness w which has length at most $\text{poly}(|x|)$. Let C be a finite set, called the *challenge space*. Let $x \in L(R)$ and let w be a polynomial length witness for x. A Σ*-protocol* for R is a 3-message interactive proof between a prover P and a verifier V, both executing PPT strategies, where the prover is given (x, w) as input and the verifier is given x, satisfying the following conditions:

- *Three rounds.* There are three rounds with the following structure. First, P sends a *commitment* message m to V. Second, V responds with a uniformly randomly chosen *challenge* $c \in C$. Third, P sends an *opening* message z to V. The verifier either accepts or rejects (based on a *deterministic* computation on x and the transcript (m, c, z)).
- *Completeness.* For every common input $x \in L(R)$, there exist PPT prover and verifier strategies P and V such that the verifier V accepts after executing the 3-round protocol with P on input x.
- *Special soundness.* There exists a probabilistic polynomial-time algorithm, called an *extractor*, that outputs a witness w for an $x \in L(R)$ when given two accepting transcripts (m, c_1, z_1) and (m, c_2, z_2) for x, with $c_1 \neq c_2$.
- *Honest-verifier zero-knowledge.* There exists a PPT simulator $\texttt{Sim}$ that outputs a transcript (m, c, z) for an input $x \in L(R)$ that is identically distributed to the proof transcript for honest players executing the 3-round protocol on input x. In other words, $\Pi\langle P \leftrightarrow V\rangle(x; w) \equiv \texttt{Sim}(x)$.

We secretly already introduced Σ-protocols in Figure 13a.

Recall from Section 5.6.2 that special soundness implies soundness with soundness error $\frac{1}{|C|}$ [23]. With special soundness, it becomes easier to design and analyze Σ-protocols, as special soundness is much easier to prove than (statistical) soundness. Typically, for Σ-protocols the most challenging part is formally proving (honest-verifier) zero-knowledge.

7.5 Example Σ-Protocol: Schnorr’s identification scheme

We give an example of an interactive proof for a *discrete logarithm* in $\mathbb{Z}_q^*$ for a prime q.

Example 7.15. Let q be a prime number and $\mathbb{Z}_q^*$ the multiplicative group of integers modulo q. The common input consists of a

[12] The widely believed conjecture is that the *polynomial time hierarchy* does not collapse. This hierarchy consists of a set of complexity classes that roughly capture problems that are harder than $\mathcal{NP} \cup \text{co}\mathcal{NP}$. If two of these classes turn out to be equal (which is unproven but believed unlikely), then the whole hierarchy collapses to that class.

[13] We say that a function is *quasilinear* for input size n if it is of order $O(n\,\text{poly}(\log^c(n)))$, with $c = O(1)$.

generator $g \in \mathbb{Z}_q^*$, and a group element $x = g^w \in \mathbb{Z}_q^*$. Paul wants to prove to Vanessa that he knows w without revealing w. He does this through the following interactive zero-knowledge proof:

(1) Paul picks a $v \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ uniformly at random and sends $m = g^v$ to Vanessa.
(2) Vanessa picks a challenge $c \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ uniformly at random and sends c to Paul.
(3) Paul calculates $z = v - cw$ and sends z to Vanessa.
(4) Vanessa checks whether $m = g^z x^c$. She accepts if this is the case and rejects otherwise.

Completeness follows from the fact that

$$g^z x^c = g^z g^{wc} = g^{v-cw+wc} = g^v = m.$$

Soundness follows from special soundness: given two accepting proof transcripts $(m = g^v, c_1, z_1 = v - c_1 w)$ and $(m = g^v, c_2, z_2 = v - c_2 w)$ (with $c_1 \neq c_2$), we can extract the witness w by calculating

$$\frac{z_2 - z_1}{c_1 - c_2} = \frac{-c_2 w + c_1 w}{c_1 - c_2} = w.$$

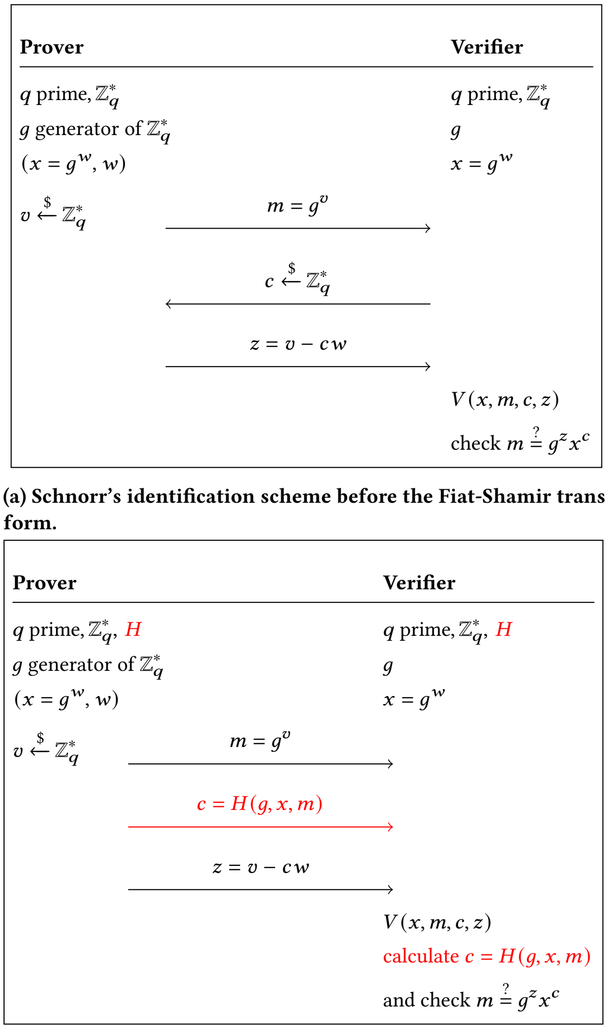
This shows that special soundness holds, and since the challenge space C in this case is $\mathbb{Z}_q^*$, this proof system is sound with soundness error $\frac{1}{|\mathbb{Z}_q^*|} = \frac{1}{q-1}$.

The proof system is zero-knowledge, as Paul does not reveal any information about w. We can construct a simple simulator algorithm `Sim`: we start by selecting a z and c uniformly at random ($c \overset{\$}{\leftarrow} \mathbb{Z}_q^*, z \overset{\$}{\leftarrow} \mathbb{Z}_q^*$), after which we set $m = g^z x^c$. Then, (m, c, z) is a valid proof transcript, as $m = g^{z+cw} = g^v$ with $v := z + cw$. Note that by the DLOG assumption,[14] the input $x = g^w$ reveals nothing about the witness w.

Now, we can convert this into a non-interactive zero-knowledge proof by applying the Fiat-Shamir transform. The common input still contains the generator $g \in \mathbb{Z}_q^*$ and a group element $x = g^w \in \mathbb{Z}_q^*$, but here it also contains a cryptographic hash function[15] H. Again, Paul wants to prove that he knows w. Paul executes the following protocol; we highlight the differences with the interactive protocol in red.

(1) Paul picks a $v \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ uniformly at random and calculates $m = g^v$.
(2) Paul calculates $c = H(g, x, m)$, which looks like a random element in $\mathbb{Z}_q^*$ by the properties of a cryptographic hash function.
(3) Paul calculates $z = v - cw$ and sends (m, z) to Vanessa.
(4) Vanessa calculates $c = H(g, x, m)$ and checks whether $m = g^z x^c$. She accepts if this is the case and rejects otherwise.

We see that the interaction is removed by letting Paul create his own challenge. The Fiat-Shamir transform applied to Schnorr's identification scheme is represented schematically in Figure 15.


Prover
Verifier
q prime, $\mathbb{Z}_q^*$
q prime, $\mathbb{Z}_q^*$
g generator of $\mathbb{Z}_q^*$
g
$(x = g^w, w)$
$x = g^w$
$v \overset{\$}{\leftarrow} \mathbb{Z}_q^*$
$m = g^v$
$c \overset{\$}{\leftarrow} \mathbb{Z}_q^*$
$z = v - cw$
$V(x, m, c, z)$
check $m \overset{?}{=} g^z x^c$
Prover
Verifier
q prime, $\mathbb{Z}_q^*$, H
q prime, $\mathbb{Z}_q^*$, H
g generator of $\mathbb{Z}_q^*$
g
$(x = g^w, w)$
$x = g^w$
$v \overset{\$}{\leftarrow} \mathbb{Z}_q^*$
$m = g^v$
$c = H(g, x, m)$
$z = v - cw$
$V(x, m, c, z)$
calculate $c = H(g, x, m)$
and check $m \overset{?}{=} g^z x^c$

(a) Schnorr's identification scheme before the Fiat-Shamir transform.

(b) The result of applying the Fiat-Shamir transform to Schnorr's identification scheme. Differences with the interactive proof are highlighted in red.

Figure 15: The Fiat-Shamir transform applied to Schnorr's identification scheme.

8 TWO STATE-OF-THE-ART CONSTRUCTIONS

In this section, we discuss two important state-of-the-art constructions of probabilistic proof systems, namely *compressed Σ-protocols* and the *Fast Reed-Solomon IOPP (FRI)*. We compare their techniques and observe a similarity in the 'folding' technique. They both trade more rounds for shorter proof lengths by 'folding' their domain spaces: they end up with a logarithmic number of rounds, in which a constant number of bits is communicated, resulting in a logarithmic proof length. Note that in the Random Oracle Model it is not expensive to add more rounds, as the Fiat-Shamir transform can remove the interactivity, resulting in a single shorter proof.

[14] The Discrete Logarithm (DLOG) assumption relative to a group G of order q says that given a $g \in G$ and a $k \in \mathbb{Z}_q^*$, it is hard to determine k from g^k.
[15] Although we defined cryptographic hash functions in terms of bit-strings, here we identify $\mathbb{Z}_q^*$ with any binary representation of it.

8.1 Bulletproofs and Compressed Σ-protocols

In this section, we discuss an efficient *compressed Σ-protocol* for proving the circuit SAT problem (see Section 5.1). Compressed Σ-protocols are a follow-up of so-called *Bulletproofs*, which we discuss first.

Roughly, a Bulletproof is a non-interactive zero-knowledge proof that offers short proofs (logarithmic in witness size). Bulletproofs can be applied to construct zero-knowledge arguments for arithmetic circuit satisfiability [18]. The round and communication complexities are *logarithmic* in the circuit size[16]. Bulletproofs can also be used to prove that a value is inside some range (which we saw is useful for blockchains to prove that one has enough funds for a purchase without revealing their account balance, see Section 7) [21]. Attema and Cramer used similar ideas to Bulletproofs to strengthen the power of Σ-protocols [4], resulting in a simpler zero-knowledge protocol for arithmetic circuit satisfiability for linear relations, achieving the same round and communication complexities as in the version using Bulletproofs.

They start by applying an ingenious folding technique to a zero-knowledge proof for linear form evaluation to *compress* it. Roughly, this folding technique works as follows. They start from a natural zero-knowledge Σ-protocol which proves that a linear form evaluation has been done correctly under the DLOG assumption. However, this basic protocol requires the prover to send n (which is linear in the input size) field elements in the last round (essentially consisting of every masked version of the secret input). The main trick is then to replace this last step by another protocol, in which the prover has to prove the correctness of a linear form evaluation over a domain of size $\frac{n}{2}$. The prover now has to send two messages: the first consists of a constant number of field elements, and the second consists of $\frac{n}{2}$ elements. The folding is then repeated for the message consisting of $\frac{n}{2}$ elements. By recursively applying this protocol $\log n$ times, we get $O(\log n)$ messages, each consisting of a constant number of field elements. Hence, we end up with a compressed Σ-protocol with logarithmic total proof length.

Using techniques from multi-party computation (e.g., *multiplication triples*), Attema and Cramer also extended this technique to construct a compressed Σ-protocol for general arithmetic circuits[17].

8.2 Fast Reed-Solomon Interactive Oracle Proof of Proximity

In Section 5.5.2, we briefly mentioned the *Fast Reed-Solomon Interactive Oracle Proof of Proximity*. We saw that the FRI is an IOPP that can prove in linear time whether a vector is close to a codeword in a Reed-Solomon code. The main idea is that a (possibly malicious) prover gives a Reed-Solomon codeword to a verifier, but as the verifier does not trust the prover, they want to spot-check the codeword and assert whether it is actually (close to) a Reed-Solomon codeword.

Historically, Reed-Solomon codes have played an important role in the construction of ZKPCPs and ZKIOPs [9]. One of the main challenges is that ZKPCPs and ZKIOPs for membership in Reed-Solomon codes usually have large computational complexities. To mitigate this problem, the FRI has been developed [9], which achieves logarithmic query complexity in linear-time. FRI, therefore, beats earlier PCPs and IOPs that only achieved polynomial query complexity in super-linear proving time.

In the FRI protocol, we again see a similar folding technique as the one used for compresse Σ-protocols. We start with a polynomial (whose degree is linear in the input size) over a linear sized domain, which we consider as too large for the verifier to read. The verifier then challenges the prover to construct a related polynomial over a domain that has half the size of the original domain. This folding is repeated a logarithmic number of times, where in each step the verifier checks for consistency in the related polynomials. The result is a protocol consisting of a logarithmic number of rounds, with a constant number of bits communicated per round. Therefore, FRI is able to achieve small proof sizes.

8.3 Applying same techniques across different proof systems

We saw that the compressed Σ-protocol and FRI both use some sort of folding procedure, where the basic idea is roughly to reduce the proof size in each step, to cut down the amount of communication between the prover and the verifier. By repeating these steps a logarithmic number of times, we end up with constant communication complexity per round, which results in a small proof.

Note that in our taxonomy, the FRI falls under *IOPPs*, whereas the compressed Σ-protocol falls under *IPs*, shining light on the fact that the same technique — a folding technique in this case — can be applied to different types of proof systems. This idea is strengthened by another paper, in which techniques based on the Fiat-Shamir Transform (which transforms zero-knowledge interactive Σ-protocols to non-interactive zero-knowledge proofs) are applied to convert public-coin IOPs to non-interactive proofs in the Random Oracle Model [12].

These techniques that have been applied to different proof systems beg the question whether it holds in general that techniques in one proof system can be applied to another. All proof systems discussed throughout this paper have been developed somewhat independently. For future research, it would be interesting to investigate whether a more general theory of proof systems can be developed, where techniques used in one proof system can easily be transferred to another proof system, with the aid of our understanding of the different design principles as discussed in Section 5.

One could start this research by investigating whether a folding technique, similar to the one used for compressed Σ-protocols and FRI, can be applied to analyze the soundness of IOPs and IOPPs. Could it be that a single type of soundness analysis could be effective for many of these independently-developed protocols?

9 DISCUSSION

In this paper, we created a comprehensive overview of the literature on (probabilistic) proof systems. We separated this task into the *structure* of proof systems, with as main result our taxonomy in Section 6, and into *zero-knowledge*, which we discussed in Section 7.

Our taxonomy categorized the proof systems symmetrically around interactivity. In both the non-interactive and the interactive branch, we saw that proof systems have been developed with

[16] The size of a circuit is based on the number of gates and wires in the circuit.

[17] Follow-up works (e.g., [5, 6]) have also designed compressed Σ-protocols based on other computational assumptions.

either full proof and input access, with one of proof or input access bounded, or with both proof and input access bounded. Most of these proof systems have developed independently, but the symmetry in our taxonomy begs the question whether a more general theory of proof systems can be developed, such that techniques applied to one proof system would automatically apply (perhaps differently) to other proof systems. This idea of transferring techniques to other proof systems is reinforced by our discussion in Section 8.3, where we saw multiple examples of very similar techniques that were developed independently of each other. A first step towards the ambitious goal of generalizing the theory of proof systems could be to investigate whether a folding technique, similar to the one used for compressed Σ-protocols and FRI, can be applied to analyze the soundness of IOPs and IOPPs as well.

In our analysis of zero-knowledge in the literature (Section 7), we encountered that zero-knowledge has been studied explicitly for all the proof systems in our taxonomy, *except for zero-knowledge IOPPs*. As IOPPs have mostly been used to prove proximity of a word to some code (be it Reed-Solomon codes or algebraic geometry codes, see Section 5.5.2), perhaps there has not been need yet for zero-knowledge IOPPs. Furthermore, the 'randomization' of the input (as described in Section 7.3.4) already provides an easy way of constructing zero-knowledge IOPPs. Throughout our study, we noticed that proof systems have mostly been developed as a result of some need (e.g., proofs of proximity were developed after there was a need for sublinear time verifiers). To flip this order, it would be interesting to formally define and study zero-knowledge IOPPs, and thereafter analyze whether they can give us more efficient zero-knowledge constructions of IOPPs than those obtained by the 'randomization of input' technique.

10 THREATS TO VALIDITY

In this section, we discuss the most prominent threats to the validity of this study. Fortunately, for theoretical computer science we do not have to worry about statistical validity threats, as no experiments are involved in mathematical reasoning. The only main validity threat is how rigorous the proofs are for the statements that we have included in this study, as we fully rely on their correctness. Aiming to reduce the validity threat emerging from non-rigorous proofs, we only selected peer-reviewed studies. That means that the proofs for all results in this paper have been read and validated by multiple researchers. Especially journal papers are held to very high standards, which allows us to be very confident in the validity and rigorousness of the proofs in those papers. However, we also included preprints of conference publications. Although those papers have been peer-reviewed as well, they are not held to the same standard as journal papers, meaning they are not subjected to complete scrutiny. Hence, it could be possible that we have included results from conference publications that contain mistakes in their proofs which might be corrected at a later point in time. The goal, however, of this study was to give a comprehensive overview of the knowledge on probabilistic proof systems at the time of writing.

11 CONCLUSION

In this paper, we discussed the structure of (probabilistic) proof systems and the zero-knowledge property that provers in such systems may have. We created a taxonomy of the structure of proof systems, which is presented in Section 6. We found that zero-knowledge has been studied extensively as a property of all proof systems in our taxonomy, except for IOPPs.

Our research started with reading surveys on proof systems, which pointed us to interactive proofs, PCPs, and zero-knowledge interactive proofs [28–30, 32, 42]. Then, using search queries for other known proof systems (such as IOPs), we read papers on the other proof systems that ended up in our taxonomy. We applied a snowballing technique to find more papers on the proof systems we studied by going through the references of primary studies and surveys.

Our first research question was which type of probabilistic proof systems exist and what design principles are at their basis. This question is fully answered by our taxonomy in Figure 14. We saw that the design principles boil down to three options:

- interactive or non-interactive;
- full proof access or bounded (oracle) proof access;
- full input access or bounded (oracle) input access.

Our second research question, namely what the power of the different systems is and in what way they can be classified, is also mostly answered by our taxonomy. We saw that interactivity increases the expressiveness of a proof system (i.e., interactive proof systems may prove more statements than their non-interactive counterparts). Bounding proof access mainly resulted in less communication between the prover and verifier, and bounded input access allowed verifiers to run in sublinear time. The classification of proof systems coincides with the design principles that underlie them. One main potential implication of our work is that proof systems that have been studied and developed independently could perhaps be unified into a more generalized theory, allowing us to transfer techniques between different proof systems.

Future research

Throughout this paper, we proposed three topics for future research, which in increasing ambition are:

- Formally define zero-knowledge IOPPs and study possible constructions of them (with the aim of achieving more efficient zero-knowledge IOPPs).
- Transfer the analysis techniques used for compressed Σ-protocols to IOP(P)s to analyze their soundness.
- Generalize the theory of (probabilistic) proof systems, focusing on methods to transfer techniques from one proof system to another, and on developing a single type of soundness analysis that could be effective for many of the independently-developed proof systems.

REFERENCES

[1] Sanjeev Arora and Boaz Barak. 2009. *Computational complexity: a modern approach.* Cambridge University Press.

[2] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. 1998. Proof Verification and the Hardness of Approximation Problems. *J. ACM* 45 (1998), 501–555.

[3] Sanjeev Arora and Shmuel Safra. 1998. Probabilistic Checking of Proofs: A New Characterization of NP. *J. ACM* 45 (1998), 70–122.

[4] Thomas Attema and Ronald Cramer. 2020. Compressed-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020,*

Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III. Springer, 513–543.

[5] Thomas Attema, Ronald Cramer, and Lisa Kohl. 2021. A Compressed Σ-Protocol Theory for Lattices. In *Advances in Cryptology – CRYPTO 2021*. 549–579.

[6] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. 2021. Compressed Σ-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures. In *Advances in Cryptology – ASIACRYPT 2021*. Springer, 526–556.

[7] László Babai. 1985. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. 421–429.

[8] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. 1991. Checking Computations in Polylogarithmic Time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*. 21–32.

[9] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Fast reed-solomon interactive oracle proofs of proximity. In *45th international colloquium on automata, languages, and programming (icalp 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[10] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. 2022. Elliptic Curve Fast Fourier Transform (ECFFT) Part II: Scalable and Transparent Proofs over All Large Fields. (2022).

[11] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. 2016. Interactive Oracle Proofs with Constant Rate and Query Complexity. Cryptology ePrint Archive, Paper 2016/324. https://eprint.iacr.org/2016/324 https://eprint.iacr.org/2016/324.

[12] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. 2016. Interactive Oracle Proofs. In *Theory of Cryptography*, Martin Hirt and Adam Smith (Eds.). Springer Berlin Heidelberg, 31–60.

[13] Eli Ben-Sasson and Madhu Sudan. 2008. Short PCPs with polylog query complexity. *SIAM J. Comput.* 38 (2008), 551–607.

[14] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. 2004. Robust PCPs of Proximity, Shorter PCPs, and Applications to Coding. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*. 1–10.

[15] Itay Berman, Ron D Rothblum, and Vinod Vaikuntanathan. 2017. Zero-knowledge proofs of proximity. *Cryptology ePrint Archive* (2017).

[16] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-Interactive Zero-Knowledge and its Applications. In *20th ACM Symposium on the Theory of Computing*. 103–112.

[17] Dan Boneh, Shafi Goldwasser, Dawn Song, Justin Thaler, and Yupeng Zhang. 2023. *Zero-Knowledge Proofs (MOOC)*. https://zk-learning.org/

[18] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. 2016. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology – EUROCRYPT 2016*. Springer, 327–357.

[19] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. 2022. Zero-knowledge IOPs with linear-time prover and polylogarithmic-time verifier. In *Advances in Cryptology – EUROCRYPT 2022*. Springer, 275–304.

[20] Sarah Bordage and Jade Nardi. 2020. Interactive oracle proofs of proximity to algebraic geometry codes. *arXiv preprint* (2020).

[21] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 315–334.

[22] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC '71)*. 151–158.

[23] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 2001. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology – CRYPTO'94*. Springer, 174–187.

[24] Irit Dinur. 2006. The PCP theorem by gap amplification. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*. 241–250.

[25] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems.. In *Crypto*, Vol. 86. Springer, 186–194.

[26] Uriel Fiege, Amos Fiat, and Adi Shamir. 1987. Zero knowledge proofs of identity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 210–217.

[27] Alexander Glaser, Boaz Barak, and Robert J Goldston. 2014. A zero-knowledge protocol for nuclear warhead verification. *Nature* 510 (2014), 497–502.

[28] Oded Goldreich. 1993. A taxonomy of proof systems. *ACM SIGACT News* 24, 4 (1993), 2–13.

[29] Oded Goldreich. 1999. *Probabilistic proof systems*. Springer.

[30] Oded Goldreich. 2002. Zero-Knowledge twenty years after its invention. *IACR Cryptol. ePrint Arch.* (2002), 186.

[31] Oded Goldreich. 2004. *Foundations of Cryptography*. Cambridge University Press.

[32] Oded Goldreich. 2008. Probabilistic Proof Systems: A Primer. *Foundations and Trends in Theoretical Computer Science* (2008), 1–91.

[33] S Goldwasser, S Micali, and C Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. 291–304.

[34] Shafi Goldwasser and Michael Sipser. 1986. Private coins versus public coins in interactive proof systems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. 59–68.

[35] Tom Gur and Ron D. Rothblum. 2015. Non-Interactive Proofs of Proximity. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*. 133–142.

[36] Carmit Hazay, Yehuda Lindell, Carmit Hazay, and Yehuda Lindell. 2010. Sigma protocols and efficient zero-knowledge. *Efficient Secure Two-Party Protocols: Techniques and Constructions* (2010), 147–175.

[37] Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. 2012. On efficient zero-knowledge PCPs. In *9th Theory of Cryptography Conference*. Springer, 151–168.

[38] Yuval Ishai and Mor Weiss. 2014. Probabilistically Checkable Proofs of Proximity with Zero-Knowledge. In *Theory of Cryptography*, Yehuda Lindell (Ed.). 121–145.

[39] Joe Kilian. 1992. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 723–732.

[40] Joe Kilian, Erez Petrank, and Gábor Tardos. 1997. Probabilistically checkable proofs with zero knowledge. In *Proceedings of the 29th annual ACM symposium on Theory of Computing*. 496–505.

[41] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. 2013. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*. 397–411.

[42] Juha Partala, Tri Hong Nguyen, and Susanna Pirttikangas. 2020. Non-Interactive Zero-Knowledge for Blockchain: A Survey. *IEEE Access* (2020).

[43] David Pointcheval and Jacques Stern. 1996. Security Proofs for Signature Schemes. In *Advances in Cryptology — EUROCRYPT '96*. 387–398.

[44] Sofya Raskhodnikova. 2012. Course Introduction to the Theory of Computation. The Pennsylvania State University.

[45] Guy N. Rothblum, Salil Vadhan, and Avi Wigderson. 2013. Interactive Proofs of Proximity: Delegating Computation in Sublinear Time. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing (STOC '13)*. 793–802.

[46] Adi Shamir. 1992. IP = PSPACE. *J. ACM* 39 (1992), 869–877.

[47] Rajeev Sobti and Ganesan Geetha. 2012. Cryptographic hash functions: a review. *International Journal of Computer Science Issues (IJCSI)* (2012).