Understanding Zero-Knowledge Proof Systems

Adrian Hamelink

June 2021

Abstract

Over the last few years, a lot of research has been conducted around the study of efficient zero-knowledge proof systems. The ultimate goal is to produce proofs capable of attesting the correct execution of arbitrary programs such that verification time is logarithmic compared to that of running the program directly. Moreover, these proofs should be representable as a short single string of bits which do not reveal anything about the inputs used.

In this report, we provide an overview of notions from complexity theory, information theory, and cryptography. This will provide us with the necessary intuition to understand and compare thee modern proof systems.

Contents

1 Intro **3**

2 Complexity Theory Background **4**
2.1 Proofs 4
2.2 Machines and Algorithms 4
2.3 The NP complexity class & SAT 5
2.4 R1CS 6

3 Proof systems **7**
3.1 Interactive Proofs (IP) 8
3.2 Probabilistically Checkable Proofs (PCP) 12
3.3 Interactive Oracle Proofs (IOP) 16

4 Aurora, Fractal, Marlin **23**
4.1 Aurora 25
4.2 Marlin 27
4.3 Fractal 33
4.4 Evaluation 35

5 Conclusion **36**

References **38**

1 Intro

The last decade has seen an incredible surge in research surrounding "*zk-SNARK*"s, an acronym for ***z**ero-**k**nowledge **S**uccinct **N**on-interactive **AR**gument of **K**nowledge*. Behind these loaded terms is the idea of proving that one knows a fact which confirms that a statement is true. The proof takes the form of a short single string of bits (relative to the size of the original statement) and is verifiable in less time than it would take to naïvely verify the original fact. They should also be unforgeable under standard cryptographic assumptions. Finally, the proof does not leak any information at all about the fact being proved. The statements usually take the form of some computation problem like:

"*Given a public input x and a function F,*
I know a secret w such that $F(x, w)$ returns true."

Blockchains are a very intriguing candidate for applying zkSNARKs, since they can be used to store and verify proofs in an efficient way. This is the case for Zcash [BCG+14] where zkSNARKs are used to encode coin transactions without revealing information about the sender or receiver. In contrast, transactions on classic blockchains are represented as signatures, whose verification requires the identity of the parties involved.

Another interesting application is *verifiable computation*, for checking that a computationally intensive program was executed correctly. The important property here is succinct verification which runs with exponentially less resources. For example, consider the task of running T iterations of a machine learning training loop and delegating to a cloud service provider. By accompanying the resulting model with the appropriate proof, the client could verify this model was correctly computing in only $\log T$ time.

We structure this report in a way that naturally introduces the many "ingredients" that make up some of the most recent proof systems. The main goal is to equip the reader with enough knowledge and curiosity to pursue further research into this subject, and so we will focus more on developing *intuition*, rather than full give details and security proofs.

In Section 2, we recall basic concepts from computational complexity theory and establish a framework for describing proof systems. Section 3 presents the IP, PCP, and IOP proof models, and describes how these constructions can be instantiated in practice. Finally, we explore the Aurora [BCR+19], Marlin [CHM+20], and Fractal [COS20] proof systems in Section 4. This will provide insight as to how modern zkSNARKs are constructed, and how different setups can affect their overall efficiency. We then evaluate these using existing open-source libraries.

Notation For $n \geq 1$, we denote the set $\{1, \ldots, n\}$ as $[n]$. We refer to a field of order q as $\mathbb{F}_q$, but may omit the subscript when the it is clear from the context. The set $\mathbb{F}^{\leq d}[X_1, \ldots, X_n]$ contains all polynomials over $\mathbb{F}$ over n variables where the degree of each monomial is less than d. The letter λ denotes the *security parameter*.

2 Complexity Theory Background

The concepts of *proofs* and *computations* are formalized through the field of complexity theory. They will allow us to reason in more *mathematical way*, mostly though polynomials and linear algebra. After providing the necessary definitions, we will be able to express problems as objects in sets. We also present the *circuit satisfaction problem* which generalizes the type of question we asked in the introduction.

2.1 Proofs

We start by formalizing the notion of *proofs* using terminology from complexity theory. Any expressible object can be written as a sequence of symbols from an *alphabet* which we denote by Σ. Usually, we set Σ to be a field $\mathbb{F}$, or the binary set $\{0,1\}$. The set $\Sigma^\star$ defines all finite sequences of elements in Σ, including the empty sequence ε. For any element $e \in \Sigma^\star$, the *length* of e is represented by $|e|$.

Elements in $\Sigma^\star$ can define *statements* and *proofs*. The validity of a proof w for a statements x depends on whether (x,w) belongs to *relation* $\mathcal{R} \subset \Sigma^\star \times \Sigma^\star$. We can also consider $\mathcal{R}$ as a predicate where $\mathsf{R}(x,w) = \mathsf{true}$ if and only if $(x,w) \in \mathcal{R}$.

A *language* $\mathcal{L}$ is a set of provable statements, which can be defined given an existing relation $\mathcal{R}$:

$$\mathcal{L}_{\mathcal{R}} := \{x \in \Sigma^\star \mid \exists w \in \Sigma^\star, (x,w) \in \mathcal{R}\}.$$

This notation allows us think about proofs and statements in mathematical terms. For example, given a question like "*Is $N \in \mathbb{N}$ the product of two primes?*", we define the relation

$$\mathcal{R}_{\mathsf{COMPOSITE}} := \{(N; (p,q)) \mid N, p, q \in \mathbb{N}, p, q \text{ are prime, and } N = p \cdot q\}.$$

The corresponding language is then

$$\mathcal{L}_{\mathsf{COMPOSITE}} := \{N \in \mathbb{N} \mid \exists \, \text{primes } p, q \text{ such that } N = p \cdot q\}.$$

2.2 Machines and Algorithms

Throughout this report, we will often be referring to *machines* and *algorithms* performing calculations that maps inputs to outputs. The terminology surrounding these can vary a lot in the literature, and delving into the details is beyond the reach of this report. In general, we will refer to an *algorithm* as the set of instructions that must be followed to produce the output. From these instructions, we can construct an abstract computational *machine* performing the same computation. The machine is composed of a *head* which can read the symbol of a cell located on an infinite length memory *tape*. Depending on the value read, the machine performs a lookup in its internal *finite state* which dictate which of the following actions it should perform.

- write a symbol in the current cell

- move to one of the adjacent cells
- repeat the above on the next cell or *halts*

This model is called a *Turing machine*, and is capable of simulating any computation that one might imagine running on a modern-day computer. We denote by $y \leftarrow \mathsf{A}(x)$ the process of an algorithm producing the output y when given x as input. The *run-time* of a machine is the number of iterations of the above loop until it halts. The run time of an algorithm is that of the machine that simulates it. We often think of the run time asymptotically with regards to the length of the input, and note that it is lower bounded by $|x|$ since the machine takes as many steps just to fully read the input.

A *probabilistic* algorithm/machine is an extension of the above construction, where the underlying deterministic machine is given an additional random tape as input. We can think of the randomness as a sequence of random *coin-flips* that the machine is allowed to make at each step. The above process is denoted as $y \leftarrow \mathsf{A}(x; \rho)$ where $\rho \in \{0,1\}^\star$. The length of ρ also imposes a lower bound on the run-time of A. In some cases where the randomness ρ is implicit, we write $y \leftarrow\$\, \mathsf{A}(x)$ to emphasize that y is a sample of some distribution, and we can look at the probability $\Pr_\rho[y \leftarrow \mathsf{A}(x; \rho)]$.

We say that an algorithm A has *oracle access* to another algorithm B if A can call B and receive the output in a single execution step. We denote this by A^{B}. In some cases, we may also write $y \leftarrow \mathsf{O}^B$ to clarify that A is using the oracle.

2.3 The NP complexity class & SAT

The definition of a language is very broad, and it is therefore useful to categorize them in different sets with certain properties. We call these categories *complexity classes*.

The most well known complexity class is NP which stands for *non-deterministic polynomial time*. It represents all statements whose proofs size and verification time is polynomial in the size of the input. Formally, a language $\mathcal{L}$ over an alphabet Σ belongs to NP if there exists a relation predicate R and a polynomial P such that

- $\mathsf{R}(x, w)$ can be evaluated in time $t \leq P(\max\{|x|, |w|\})$
- $\mathcal{L} = \{x \in \Sigma^\star \mid \exists w \in \Sigma^\star, \mathsf{R}(x, w) = \mathsf{true}, |w| \leq P(|x|)\}$

The class P (*deterministic polynomial time*) is a subset of NP, with an additional restriction: given $x \in \mathsf{P}$, there exists a deterministic algorithm for finding w that runs in time $t \leq P(|x|)$. It remains an open question whether $\mathsf{P} = \mathsf{NP}$.

An important problem in this class is the *Boolean satisfiability problem*, or SAT. Given a boolean formula C over boolean variables $x = (x_1, \ldots, x_n)$ which are linked with the AND ($\wedge$), OR ($\vee$), and NOT ($\neg$) operators, does there exist and assignment $w = (w_1, \ldots, w_n) \in \{\mathsf{true}, \mathsf{false}\}^n$ such that $C(w) = \mathsf{true}$? If one is given a candidate assignment w, we can efficiently check that it satisfies C by

simply evaluating $C(w)$. Depending on the way C is written, finding such a w can be either trivial or close to impossible without an exponential amount of ressources (time and/or space).

The Cook-Levin theorem [Coo71, Lev73] states that SAT is NP-Complete, which means that for every problem in NP, there exists a polynomial-time *reduction* to SAT. In other words, every NP problem can be expressed as SAT instance.

Alternatively, we can view boolean circuit in SAT as an *arithmetic circuit* defined by a set of polynomials $p_1, \dots, p_m \in \mathbb{F}^{\leq 2}[X_1, \dots, X_n]$. For each gate in the circuit, we define a quadratic equation over the inputs and output variable which is satisfiable if and only if the polynomial evaluates to 0.

$$\begin{aligned} x_k = x_i \wedge x_j &\iff x_k - x_i \cdot x_j = 0 \\ x_k = x_i \vee x_j &\iff x_k - x_i - x_j + x_i \cdot x_j = 0 \\ x_k = \neg x_i &\iff x_k(1 - x_i) = 0 \end{aligned}$$

This is sufficient if $\mathbb{F} = \mathbb{F}_2$, but if we want to allow other fields, we must add n more constraints of the form $x_i(1 - x_i) = 0,\ \forall i \in [n]$.

This yields an efficient transformation of any circuit C over n variables, to an instance of *quadratic equation satisfiability*, or QESAT, whose relation is defined by:

$$\mathcal{R}_{\mathsf{QESAT}} = \left\{ \big((p_1, \dots, p_m); w\big) \,\middle|\, \begin{array}{l} w \in \mathbb{F}^n \\ p_i \in \mathbb{F}_{\leq 2}[X_1, \dots, X_n] \end{array} \; p_i(w) = 0,\ \forall i \in [m] \right\}.$$

2.4 R1CS

A *Rank-1 Constraint System*, or R1CS, is is yet another way of formulating boolean satisfiability problems. Rather than polynomials, we consider 3 *matrices* $A, B, C \in \mathbb{F}^{m \times n}$ a witness $w \in \mathbb{F}^n$. Each set of rows $\{A_i, B_i, C_i\}_{i=1}^m$ of the matrices then define a polynomial $p_i \in \mathbb{F}^{\leq 2}[X_1, \dots, X_n]$, representing m different contraints the witness should satisfy:

$$p_i(w) = \langle A_i, w \rangle \cdot \langle B_i, w \rangle - \langle C_i, w \rangle = 0, \quad i \in [m].$$

We can also consider a *public input* $v \in \mathbb{F}^k$ to the circuit, by setting $z = (v, w) \in \mathbb{F}^n$ for $w \in \mathbb{F}^{n-k}$. Moreover, we can combine all the equations using the *Hadamard product*, which performs component-wise multiplication between two vectors and is represented by the "$\circ$" symbol. This yields the R1CS relation

$$\mathcal{R}_{\mathsf{R1CS}} := \left\{ \big((A, B, C, v) \,; w\big) \,\middle|\, \begin{array}{l} A, B, C \in \mathbb{F}^{m \times n} \\ v \in \mathbb{F}^k, w \in \mathbb{F}^{n-k} \end{array} \; , Az \circ Bz = Cz, z = (v, w) \right\}.$$

3 Proof systems

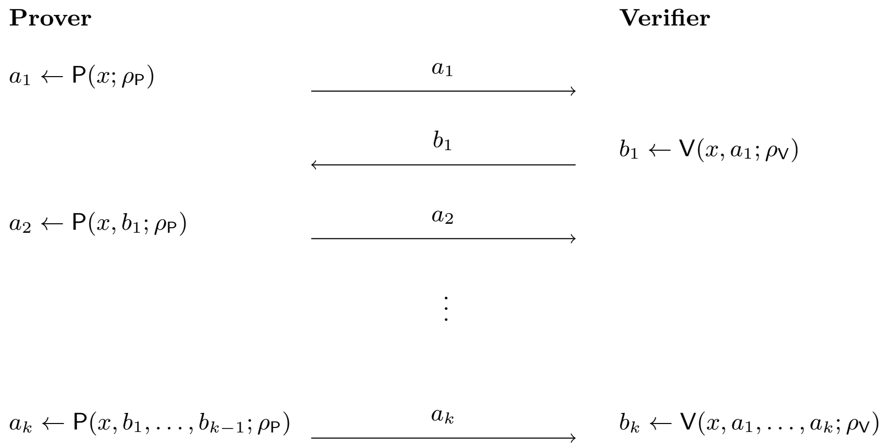
The previous section formalized the notions of statements and proofs. In particular, we saw that we can consider and NP problem in terms of circuits, allowing us to ask questions like:

"Given a circuit C and public input x,
does there exists a witness w such that $C(x; w) = 0$?"

This section focuses on interaction models between a prover P and a verifier V, where P's goal is to convince V that a common instance x does in fact belong to some language $\mathcal{L}$.

Here, we consider P and V as *interactive machines*, an extension to the construction in § 2.2. Both parties have access to an additional *communication tape* where they can write symbols for the other party to read. The machine A (representing either P or V) is modeled as follows: It maps some input x and a list of *incoming messages* $m_1, \ldots, m_n$ to an *outgoing message* $m_{n+1} \leftarrow \mathsf{A}(x, m_1, \ldots, m_n)$. The sequence $(x, m_1, \ldots, m_n)$ is the n-th *partial view* of A. We can think of A as an algorithm that maps an algorithm computes the next message from a partial view.

The above interaction over k rounds is modeled as follows, where we consider P and V as probabilistic algorithms with common input x:



Once the interaction is finished, the *transcript* is the sequence of all messages exchanged during the protocol and is denoted $\langle \mathsf{P} \overset{x}{\leftrightarrow} \mathsf{V} \rangle := (x, a_1, b_1, \ldots, a_k, b_k)$ (the randomness $\rho_\mathsf{P}, \rho_\mathsf{V}$ is implicit). The *view* of V is the set of all messages and variables that V has access to at the end of the interaction. It is defined as $\mathsf{View}_\mathsf{V}(\mathsf{P} \overset{x}{\leftrightarrow} \mathsf{V}) := (x, a_1, \ldots, a_k; \rho_\mathsf{V})$.

Finally, an interactive protocol is called *public-coin*, when all of V's messages are sampled uniformly from some predefined set, and in particular do not depend on any private input. When this is the case, then we can directly consider V's

randomness $\rho_{\mathsf{V}} = (b_1, \ldots, b_{k-1}; r_{\mathsf{V}})$, so that V's output in rounds 1 to $k-1$ are the corresponding symbol in ρ_{V}. We can therefore consider that V's final output b_k is computed deterministically from $\mathsf{View}_{\mathsf{V}}(\mathsf{P} \overset{x}{\leftrightarrow} \mathsf{V})$.

3.1 Interactive Proofs (IP)

Using an interactive protocol to construct proofs of language membership was formalized in [GMR85], using the name *interactive proof.* In this model, both prover P and verifier V are given a common string $x \in \Sigma^\star$ and must decide whether $x \in \mathcal{L}$ or not. We consider both parties as interactive probabilistic machines, but restrict V's run-time to $\mathsf{poly}(n)$, where $n = |x|$. Moreover, we define V's final message b_k as $\mathsf{dec} \leftarrow \langle \mathsf{P} \overset{x}{\leftrightarrow} \mathsf{V} \rangle \in \{\mathsf{acc}, \mathsf{rej}\}$ which reflects V's decision to accept or reject the statement $x \in \mathcal{L}$.

Since we assume both parties to be probabilistic, we need to understand the distribution of V's decision $\Pr_{\rho_{\mathsf{P}}, \rho_{\mathsf{V}}}[\mathsf{dec} \leftarrow \langle \mathsf{P} \overset{x}{\leftrightarrow} \mathsf{V} \rangle]$ rather than one specific outcome of the experiment. In particular, we are interested in measuring the probability of V being *wrong.* This happens either when $x \in \mathcal{L}$ and V outputs rej, or when $x \notin \mathcal{L}$ but V still accepts. Intuitively, using a proof system makes sense when it is hard to prove x, so the first case captures the idea that the protocol description may not always lead to the correct result. On the other hand, if an incorrect proof is accepted, then the prover doesn't actually need to do the necessary work to find a correct one. We therefore consider the probability of this happening when the prover behaves arbitrarily (even maliciously). These properties are respectively referred to as the *completeness* and *soundness.*

Formally, an *interactive proof system* IP for a language $\mathcal{L}$ is an interactive protocol between a prover P and a verifier V such that there exists a polynomial P and constants $\epsilon_c, \epsilon_s \in [0, 1]$ such that:

- The run-time of V until termination is bounded by $P(|x|)$,
- If $x \in \mathcal{L}$, then $\Pr_{\rho_{\mathsf{P}}, \rho_{\mathsf{V}}}[\langle \mathsf{P} \overset{x}{\leftrightarrow} \mathsf{V} \rangle = \mathsf{rej}] \geq 1 - \epsilon_c$,
- If $x \notin \mathcal{L}$, then $\Pr_{\rho_{\tilde{\mathsf{P}}}, \rho_{\mathsf{V}}}[\langle \tilde{\mathsf{P}} \overset{x}{\leftrightarrow} \mathsf{V} \rangle = \mathsf{acc}] \leq \epsilon_s$ for all provers $\tilde{\mathsf{P}}$.

3.1.1 $\mathsf{NP} \subset \mathsf{IP}$

Since we are mainly interested proving results about statements in NP, we must make sure that they are indeed provable using an interactive proof. To understand why, consider the IP protocol in Figure 1.

Since we assume no run-time bound on the prover it is completely valid to require it to find the witness as a first step. The verifier must then evaluate $\mathsf{R}(x; w)$ which it can do by definition in polynomial-time.

Assessing the efficiency of this system is a bit awkward, since the run-time of P is now lower bounded by the time it takes to find w. Therefore, we will often assume that P is implicitly given w as private input and we can then look its computational complexity of the protocol in isolation. This type of proof is called a *proof of knowledge* which we formalize in the next section.

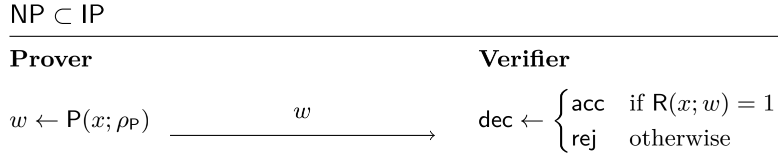


Figure 1: Interactive Proof for any $\mathcal{L}$ in NP

3.1.2 Proof of knowledge

For the class NP, a *proof of knowledge* [GMR85, BG93] is a special type of Interactive Protocol where the prover's goal is to convince a verifier that it does in fact know a witness that satisfies a relation $\mathcal{R}$.

In this type of protocol, we can concentrate on *efficient* provers who run in polynomial-time, rather than unbounded ones since we remove the burden of finding the witness in the first place. Moreover, we now assume that an honest prover is explicitly given a witness as private input. The formal definition is similar to the one for IP but we need to modify the soundness property to account for the different setting. The relation that must validate is defined as $\mathcal{R}_{\mathcal{L}} := \{(x, w) | x \in \mathcal{L}, w \in \mathcal{W}(x)\}$, where $\mathcal{W}(x)$ is the set of all valid witnesses for x.

To make sure the prover actually knows a valid witness, it should be possible to efficiently extract this knowledge from the prover. An *extractor* is a polynomial-time algorithm Ext which is given access to an arbitrary prover $\tilde{\mathsf{P}}$, and returns the witness w. In other words, the probability of extracting the witness from a malicious prover should be at least as high as the probability of $\tilde{\mathsf{P}}$ convincing the verifier. Note that the extractor is allowed to *rewind* the prover. At any point, it can restore $\tilde{\mathsf{P}}$ to one of its previous states, change the remaining unused randomness and obtain a different outcome.

The formal definition of a proof of knowledge is the following:

completeness If $(x, w) \in \mathcal{R}_{\mathcal{L}}$, then $\Pr[\mathsf{acc} \leftarrow \langle \mathsf{P}(w) \overset{x}{\leftrightarrow} \mathsf{V} \rangle] = 1$.

knowledge soundness e There exists a polynomial-time extractor algorithm Ext such that if $(x, w) \notin \mathcal{R}_{\mathcal{L}}$, then

$$\Pr[\mathsf{Ext}^{\tilde{\mathsf{P}}(x)}(x) \in \mathcal{W}(x)] \geq \Pr[\mathsf{acc} \leftarrow \langle \tilde{\mathsf{P}}(x) \leftrightarrow \mathsf{V}(x) \rangle] - e.$$

3.1.3 Zero-Knowledge

When performing a proof of knowledge protocol, it may not be in the prover's best interest to leak any information about the witness itself. Indeed, if we take again the example of the discrete logarithm problem, it is often used to prove knowledge of a secret key which the prover must keep private.

This property is called *zero-knowledge* and is defined through a polynomial-time *simulator* algorithm Sim which can simulate a prover's messages against

an arbitrary verifier $\tilde{\mathsf{V}}$. In particular, the distributions of $\mathsf{Sim}(x; \rho_{\mathsf{Sim}})$ and $\mathsf{View}_{\mathsf{V}}(\mathsf{P} \overset{x}{\leftrightarrow} \mathsf{V})$ should be z-statistically close over the randomness $\rho_{\mathsf{Sim}}, \rho_{\mathsf{P}}, \rho_{\mathsf{V}}$.

3.1.4 Fiat-Shamir

In many real world scenarios, it is not always practical to run a protocol interactively, since this requires both parties to be online at the same time. For public-coin interactive proofs, a prover can create a single message π which represents the full interaction. This technique first appeared in [FS87] in the context of signature schemes, and is often referred to as the "Fiat-Shamir" transform.

We define the concept of a *random-oracle*, which is an idealized function $\rho : \{0,1\}^\star \to \{0,1\}^\lambda$. Each time it is called on some fresh input x (that is, $\rho(x)$ has not yet been requested), it returns $y \leftarrow\$ \{0,1\}^\lambda$. Whenever $\rho(x)$ is requested after that, the same y is returned. In practice, ρ will be instantiated through some appropriate *collision-resistant hash function* like SHA-2 or BLAKE2. Assuming the existence of a random oracle makes it easier to prove the security of a protocol, and was used in [PS96] to this end for many existing signature schemes.

The idea of the transformation is quite simple: In the original protocol, whenever the verifier is instructed to send a uniformly sampled message b_i, the prover instead obtains this messages by querying ρ with input the instance x and the sequence of messages it has previously sent. That is, $b_i \leftarrow \rho(x, a_1, \dots, a_{k-1})$. The randomness r_{V} used in V's final decision message is set to $\rho(x, a_1, \dots, a_k)$.

The final proof π is then set to $(a_1, \dots, a_k)$. To verify it, a verifier simulates the original protocol by recomputing all the $b_1, \dots, b_{k-1}$ as well as r_{V} and setting $\rho_{\mathsf{V}} = (b_1, \dots, b_{k-1}, r_{\mathsf{V}})$. Finally, it verifies that $\mathsf{V}(x; \rho_{\mathsf{V}})$ returns acc.

Note that when a hash-function is used, we call the protocol an *argument of knowledge* rather than a proof of knowledge. This distinction is due to the fact that, theoretically, a computationally unbounded prover would be able to find collisions allowing it to cheat.

3.1.5 Sum-Check Protocol

The Sum-Check protocol introduced by Lund et al. [LFKN90] was used to prove that co-NP $\subset$ IP [1]. Without diving into the details of this inclusion, this section describes one of the main ideas behind this proof [2]. It involves proving the following statement about summing evaluations of a polynomial over some set:

Suppose that we are given some polynomial $p \in \mathbb{F}^{\leq d}[X_1, \dots, X_n]$ and an integer K, does the following hold?

$$K = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \dots, x_n). \qquad (1)$$

Notice that verifying this sum directly requires the evaluation of p over 2^n different inputs. Using the Sum-Check protocol, a verifier can be convinced with

[1] The class co-NP is the *complement* of NP: $\mathcal{L} \in \text{co-NP} \iff \Sigma^\star \setminus \mathcal{L} \in \text{NP}$

[2] This exposition is inspired from Justin Thaler's 2017 lecture notes [Tha17].

overwhelming probability that the sum claimed by the prover is correct, with the following efficiency (we assume for simplicity that the degree of p in X_j is d for all $j \in [n]$, and that the cost of evaluating p is T):

communication	rounds	V run-time	P run-time	ϵ_c	ϵ_s
$\mathcal{O}(n \cdot d)$	n	$\mathcal{O}(n \cdot d) + T$	$\mathcal{O}(2^n \cdot T)$	0	$\leq n \cdot d / \lvert\mathbb{F}\rvert$

The n round protocol is described in Figure 2, where we assume that the polynomials $p_j(X_j)$ sent by P are represented by their $d+1$ coefficients. The verifier can then evaluate p_j using $\mathcal{O}(n)$ $\mathbb{F}$-multiplications.

At the end of the first round, V receives p_1 and verifies that $p_1(0)+p_1(1) = K$. Unfortunately, a cheating prover could easily find a different $\tilde{p}_1$ which would pass this test, so the verifier must now make sure that

$$p_1(X_1) \equiv \sum_{x_2,\ldots,x_n \in \{0,1\}} p(X_1, x_2, \ldots, x_n).$$

Since both sides of this equation are degree d polynomials over $\mathbb{F}$, the Schwartz-Zippel lemma can be used to obtain a probabilistic guarantee that the claim holds:

$$\Pr_{r_1 \leftarrow\$\, \mathbb{F}}\left[\tilde{p}_1(r_1) = \sum_{x_2,\ldots,x_n \in \{0,1\}} p(r_1, x_2, \ldots, x_n) \mid \tilde{p}_1 \neq p_1\right] \leq \frac{d}{|\mathbb{F}|}. \quad (2)$$

The verifier therefore responds with with $r_1 \leftarrow\$\, \mathbb{F}$. Letting $K' := p_1(r_1)$ and $p'(X_2, \ldots, X_n) := p(r_1, X_2, \ldots, X_n)$, we obtain another instance of Equation 1 in only $n-1$ variables:

$$K' = \sum_{x_2 \in \{0,1\}} \sum_{x_3 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p'(x_2, \ldots, x_n).$$

The parties repeat these steps n times, until P sends $p_n(X_n)$ for which V verifies that $p_{n-1}(r_{n-1}) = p_n(0) + p_n(1)$. Rather than responding with $r_n \leftarrow\$\, \mathbb{F}$, V instead performs its only evaluation of p at $(r_1, \ldots, r_n)$ which it compares with $p_n(r_n)$. The verifier accepts if all checks pass.

The soundness error is upper bounded by $(d \cdot n)/|\mathbb{F}|$. For $n > 1$, we can consider rounds $2, \ldots, n$ of $\mathsf{Sum\text{-}Check}(K, p(X_1, \ldots, X_n))$ as the parties performing the protocol $\mathsf{Sum\text{-}Check}(p_1(r_1), p(r_1, X_2, \ldots, X_n))$. By induction, the soundness error for the reduced instance is $\leq (d \cdot (n-1))/|\mathbb{F}|$, to which we add the error probability obtained in Equation 2.

To understand why we might be interested in such a problem, consider some boolean circuit $C : \{0,1\}^n \to \{0,1\}$. Rather than asking if *there exists* a valid assignment for C (SAT), we can also wonder *how many* satisfy C ($\#\mathsf{SAT}$) Recall from § 2.3 that we can represent C as some n-variate polynomial p over $\mathbb{F}$ of maximal individual degree d, such that $C(x) \equiv p(x)$ for all $x \in \{0,1\}^n$. Therefore, verifying that H is the number of such assignments, reduces to checking Equation 1.

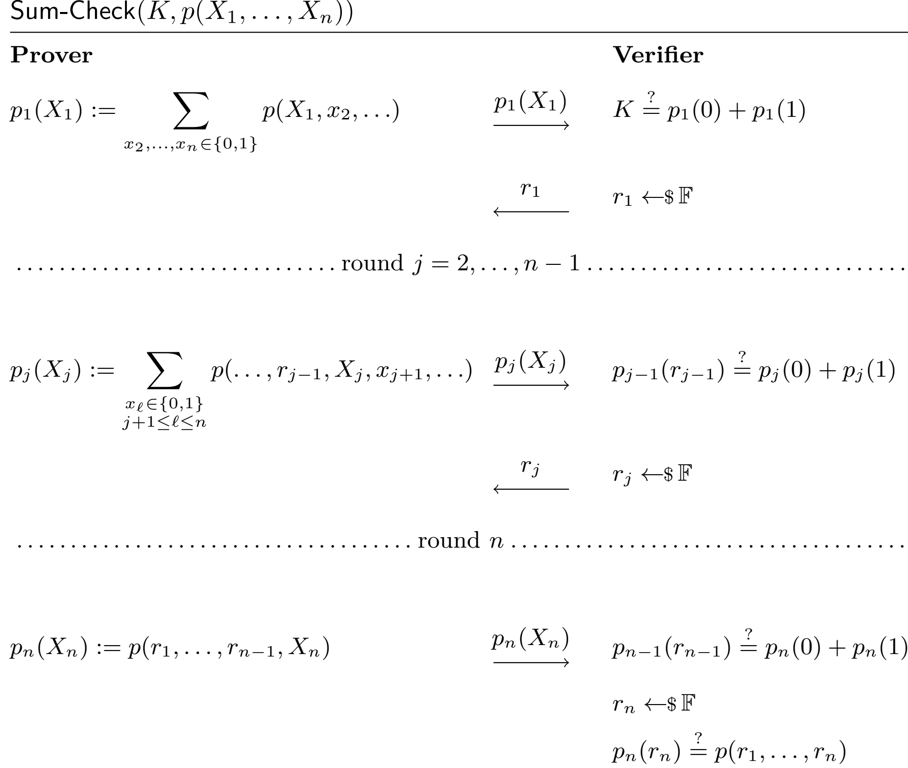


Figure 2: Sum-Check protocol

Beyond proving theoretical results though, one very powerful property of Sum-Check is that the verifier's run-time is *logarithmic* compared to the naive verification. It essentially allows us to delegate most of the work to some prover. This technique was used in [GKR08] to construct an interactive protocol for proving the evaluation of a layered arithmetic circuit $C : \mathbb{F}^n \to \mathbb{F}$ of constant depth d and size $S(n)$. While the prover runs in time $\mathsf{poly}(S(n))$, the verification is only $\tilde{\mathcal{O}}(n + d \log S(n))$ ($\tilde{\mathcal{O}}$ hides a polynomial in $\log n$)

3.2 Probabilistically Checkable Proofs (PCP)

One big limitation of IP is that the length of prover's messages sets a lower bound on the run time of V. Indeed, the verifier must read the message to be able to make its final decision.

Let us consider a slightly different computation model, where the prover produces some proof string $\Pi \in \Sigma^\ell$ and the verifier is then given *oracle access* to it via an oracle O^Π. The verifier is given a random string $\rho \in \{0,1\}^r$ that is used to compute q queries $i_1, \ldots, i_q$. It then sends these queries to the oracle, which responds with a sequence $(\Pi[i_1], \ldots, \Pi[i_q])$, where $\Pi[i]$ is the i-th symbol

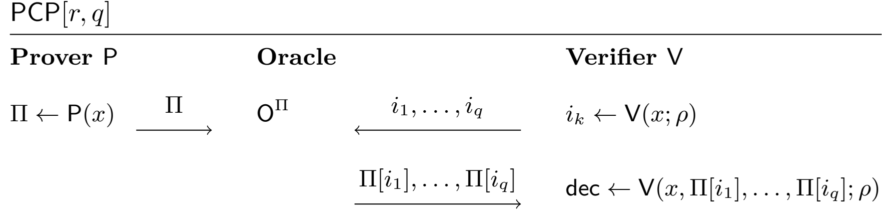


Figure 3: PCP proof model

of Π. Given this response, V decides whether or not $x \in \mathcal{L}$.

Formally, we can think of V as having access to an extra communication tape with the oracle. To read a symbol, it first writes an index i of Π on the tape, and the oracle will then write $\Pi[i]$ in the cell next to it. Figure 3 gives some more intuition about this interaction, where we consider that V interacts with a third party representing the oracle for Π.

This proof model is called a *probabilistically checkable proof (PCP)* [ALM+92, AS92]. It defines a complexity class $\mathsf{PCP}[r, q]$, where $r = r(n)$ is the number of random bits used by V and $q = q(n)$ is the number of symbols of Π that V reads. The verifier is modeled as a probabilistic polynomial time oracle algorithm V^{Π}. The system satisfies the following properties:

completeness If $x \in \mathcal{L}$, and $\Pi \leftarrow \mathsf{P}(x)$, then $\Pr_{\rho \leftarrow \{0,1\}^r}[\mathsf{V}^{\Pi}(x; \rho)] \geq 1 - \epsilon_c$

soundness If $x \notin \mathcal{L}$, for all $\tilde{\Pi} \in \Sigma^{\ell}$ then $\Pr_{\rho \leftarrow \{0,1\}^r}[\mathsf{V}^{\tilde{\Pi}}(x; \rho)] \leq \epsilon_s$

To get some intuition of what these parameters mean, we can consider different values for r, q and see what complexity classes they represent.

- $\mathsf{PCP}[r = \mathcal{O}(\log n), q = 0] = \mathsf{P}$: V has no information about Π and we can define a deterministic polynomial time algorithm $\mathsf{A}(x)$ which runs $\mathsf{V}(x; \rho)$ for all $\rho \in \{0, 1\}^r$.

- $\mathsf{PCP}[r = 0, q = \mathcal{O}(\log n)] = \mathsf{P}$: V obtains a deterministic fixed witness $w \in \Sigma^q$. A deterministic polynomial time algorithm $\mathsf{A}(x)$ could simulate V over all possible witnesses of size r.

- $\mathsf{PCP}[r = 0, q = \mathsf{poly}(n)] = \mathsf{NP}$: V is a deterministic algorithm that reads a witness w in full, and decides whether $x \in \mathcal{L}$. This characterizes NP.

A more complicated result is the *PCP theorem* [ALM+92] which states:

$$\mathsf{PCP}[\mathcal{O}(\log n), \mathcal{O}(1)] = \mathsf{NP}.$$

Essentially, a verifier who reads only a constant number of symbols from some proof can be convinced with high probability that the proof is valid. For an informal comparison, imagine a long mathematical proof written in a book with

many pages such that reading and verifying this proof would not actually be practical. Instead, a verifier could choose to verify only 10 pages randomly. It these are all consistent, then it would be convinced with high probability that the full proof is correct.

From PCPs to Interactive Agruments In practice, it is not clear how one could actually instantiate the oracle paradigm between a prover and a verifier. We might consider simplifying this exchange by letting the prover act as the oracle, so that when the verifier requests a symbol of Π it is actually the prover who responds. Unfortunately, a malicious prover could adapt their responses to make the verifier always accept. In our informal example, when we request a specific page, the prover could alter its content to make sure everything on it was correct.

We explore a technique due to Kilian and Micali [Mic94, Kil92] which uses the random-oracle model to transform a PCP into an Interactive Proof. When the random-oracle is modeled though a collision resistant hash function, we refer to it as a computationally sound interactive argument, for the same reasons detailed in § 3.1.4.

The main idea is to have the prover first commit to the proof Π in such the commitment is short, and it is possible to extract a decommitment string for each symbol of Π. Once V receives this the commitments, it sends back its list of queries for Π. The prover responds with those selected locations, along with a decommitment string for each symbol. Finally, the verifier ensures that they are consistent with the previous commitment and runs the original PCP verifier. It simulates the proof oracle using the now-verified symbols from P.

The most straightforward way to instantiate this commitment scheme is to use *Merkle Trees* [Mer88]. We start by taking some random oracle ρ whose output is in $\{0,1\}^\lambda$. To build a Merkle tree from $\Pi \in \Sigma^l$ (we assume for simplicity that l is some power of 2), we start by hashing each symbol of Π which we consider as the *leaves* of the tree. We then build up the tree by setting each node's value to be the hash of the concatenation of its two child nodes. The height of this tree is $\log(l)$, and we define its *root* as rt, which can be thought of the commitment to Π. Whenever the prover sends a value $\Pi[i]$, it also provides an *authentication path* ap_i consisting of the values of all sibling nodes on the path to the root of the tree. Upon reception, the receiver has all the necessary information to recompute rt, without needing access to any other symbol of Π. It accepts the value if the computed root matches the one previously committed.

A description of a Merkle tree with depth 4 is given in Figure 4. For example, the authentication path for $\Pi[4]$ would be (h_3, h_{12}), and the receiver computes $\mathsf{rt}' = \rho(h_{12} \| \rho(h_3 \| \rho(\Pi[4])))$.

With this technique in mind, the following Protocol 5 shows the transformation in action. We note that in this protocol, we have two sets of parties (P, V) and $(\mathsf{P}_{\mathsf{PCP}}, \mathsf{V}_{\mathsf{PCP}})$. The latter are those from the original PCP system, and are essentially simulated by the IP parties.

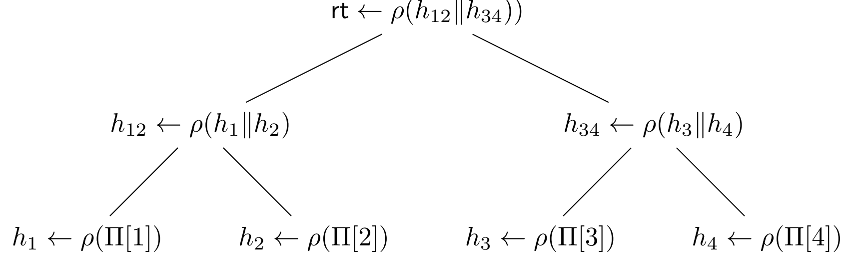


Figure 4: Merkle tree for $\Pi \in \Sigma^4$

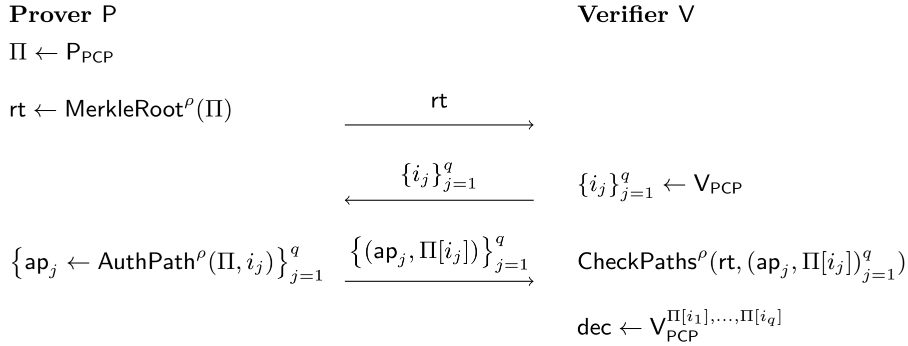


Figure 5: PCP Interactive Argument

Linear PCPs

A *Linear* PCP (LPCP) [IKO07, BCI+13, PHGR13, BCTV14] is special type of PCP where the prover is restricted to only sending a *linear function* of the form $f : \mathbb{F}^n \to \mathbb{F}, x \mapsto \langle x, y \rangle$ for some $y \in \mathbb{F}$. This function is encoded as a string Π, which is the concatenation of all evaluations of f over $\mathbb{F}^n$. To query f at x, the verifier reads index x of Π.

Without the LPCP abstraction, the prover could send y in full (increasing the communication complexity by n field elements), and have the verifier compute $\langle x, y \rangle$ itself (at the cost of $\mathcal{O}(n)$ field operations). Instead with an LPCP, all this work is shifted to prover, resulting in better performance for the verifier.

Unfortunately, the verifier cannot know for certain that Π encodes f correctly. A malicious prover could very well have sent an arbitrary function $\tilde{f} : \mathbb{F}^n \to \mathbb{F}$. Blum et al. [BLR93] came up with a probabilistic test which can be used to check that f is at least δ-*close* to some linear function. Here, "closeness" is defined by the *Hamming* distance between two functions $f, g : \mathbb{F}^n \to \mathbb{F}$ $\Delta(f, g) = \Pr_{x \leftarrow \mathbb{F}}[f(x) \neq g(x)]$, and we take δ to be the minimum of $\Delta(f, g)$ over all linear functions g.

To perform the test, V queries $f(x), f(y), f(x + y)$ for randomly sampled

$x, y \leftarrow\!\$\, \mathbb{F}$ and checks that $f(x) + f(y) = f(x + y)$. If f is δ-far from any linear function, the probability that the check passes is $\leq 1 - \min\{1/6, \delta/2\}$.

We can therefore *compile* a LPCP into an ordinary PCP by wrapping the inner LPCP with the additional linearity test. The overall soundness error is that of the LPCP plus the probability of accepting a non-linear function (though this test can be repeated to reduce it to acceptable levels).

3.3 Interactive Oracle Proofs (IOP)

After describing IP and PCP separately, one might wonder what kind of system is possible when we try "combining" them together. Kalai and Raz [KR08] introduced *Interactive* PCP*s*, but only considered the case where an IP verifier has oracle access to the prover's first message. This model was later generalized by Ben-Sasson et al. as *Interactive Oracle Proofs (IOP)* [BCS16], where *all* of the prover's messages can be considered as oracles.

3.3.1 Interactive Oracle Protocol

We start by considering the *oracle* variant of Interactive Protocols § 3.1. A k-round *interactive oracle protocol* between two probabilistic algorithms P and V is defined as follows: For $i \in [k]$, the i-th round is initiated V who sends $m_i \leftarrow \mathsf{V}^{f_1,\dots,f_{i-1}}(m_1, \dots, m_{i-1}; \rho_{\mathsf{V}}) \in \{0,1\}^{u_i}$. The prover P responds with $f_i \leftarrow \mathsf{P}(m_1, \dots, m_i; \rho_{\mathsf{P}}) \in \{0,1\}^{l_i}$ to which V then has oracle access. V's final message m_{k+1} is the decisions $\mathsf{dec} \in \{\mathsf{acc}, \mathsf{rej}\}$ denoted $\langle \mathsf{P} \leftrightarrow \mathsf{V} \rangle$.

The protocol is *public-coin* when all of V's messages m_i do not depend on any private input. We can assume without generality that they are sampled uniformly from $\{0,1\}^{u_i}$, and that any queries to P's messages are postponed until the last round when V makes its decision (we can thus view ρ_{V} as $(m_1, \dots, m_k; r_{\mathsf{V}})$ where r_{V} is some randomness used for V's decision). Such a protocol is depicted in Figure 6.

3.3.2 Interactive Oracle Proofs

An *Interactive Oracle Proof System* is k-round interactive oracle protocol defined by some relation $\mathcal{R}$ and some soundness parameter $\epsilon_s \in [0, 1)$. In addition it must satisfy the following properties:

completeness For all $(x, w) \in \mathcal{R}$, $\Pr[\mathsf{acc} \leftarrow \langle \mathsf{P}(x, w) \leftrightarrow \mathsf{V}(x) \rangle] = 1$.

soundness For all $x \notin \mathcal{L}_{\mathcal{R}}$ and all $\tilde{\mathsf{P}}$, $\Pr\big[\mathsf{acc} \leftarrow \langle \tilde{\mathsf{P}} \leftrightarrow \mathsf{V}(x) \rangle\big] \leq \epsilon_s$.

In addition, we can also require that the IOP system is a *proof of knowledge* and satisfies *honest-verifier zero-knowledge*:

proof of knowledge e There exists a probabilistic polynomial-time *extractor* algorithm Ext which may interact with an arbitrary prover $\tilde{\mathsf{P}}$ via rewinding (at any moment, $\tilde{\mathsf{P}}$ can be restored to a previous state), such that $\Pr[(x, \mathsf{Ext}^{\tilde{\mathsf{P}}}(x)) \in \mathcal{L}_{\mathcal{R}}] \geq \Pr[\mathsf{acc} \leftarrow \langle \tilde{\mathsf{P}} \leftrightarrow \mathsf{V}(x) \rangle] - e$ for all x.

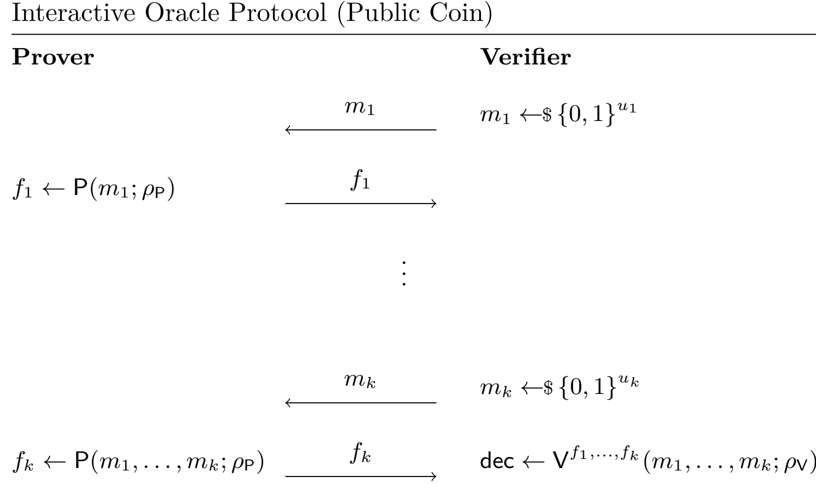


Figure 6: Interactive Oracle Protocol

honest verifier z-statistical zero-knowledge There exists a probabilistic polynomial-time *simulator* algorithm Sim such that for every $(x, w) \in \mathcal{L}_\mathcal{R}$, the distribution of $\mathsf{Sim}(x)$ and the set of query responses by V in $\mathsf{P}(x, w) \leftrightarrow \mathsf{V}$ are z-close.

The intuition behind these properties are the same as in § 3.1.

The complexity of the system is evaluated through the following quantities:

verifier time t_P the number of steps for which the prover P runs over the course of the whole interaction.

verifier time t_V the number of steps for which the verifier V runs over the course of the whole interaction.

proof length p the total number of bits communicated by the prover to V.

query complexity q number of bits queried by V to P's messages $f_1, \ldots, f_k$

We notice a few differences between the above definition, and the one presented in § 3.1. For starters, we directly assume that an honest prover is given the witness as private input. This more closely resembles real-world scenarios where the goal is to prove the correct execution of a circuit run with some private input. Moreover, we also set the completeness probability to 1, since all the protocols we present achieve this requirement. The definitions here are taken from [BCS16], and we refer the reader to it for a more complete description of IOP.

3.3.3 Non-interactive Random Oracle Proofs for IOP

Transforming an IOP into an non-interactive proof in the random oracle model follows a similar pattern to the one we saw for PCPs 3.2. In the general case, we can apply the transformation from [BCS16] which generalizes [Mic94]. We note that it only works when the underlying IOP is public-coin, since we can then simulate V's messages using the random oracle.

Let ρ_1, ρ_2 be two random oracles (used for V's messages and the Merkle tree respectively), and set $\sigma_0 \leftarrow \rho_2(x)$. Each round i for $i = 1, \ldots, k$ proceeds as follows:

- sample V's messages $m_i \leftarrow \rho_1(x \| \sigma_{i-1})$,
- send m_i to P and obtain f_i,
- compute the Merkle-tree root rt_i of f_i using ρ_2,
- set $\sigma_i \leftarrow \rho_2(\mathsf{rt}_i \| \sigma_{i-1})$.

Once the interaction is finished, set V's randomness ρ_V to $(m_1, \ldots, m_k; \rho_1(x \| \sigma_k))$ and simulate $\mathsf{V}^{f_1,\ldots,f_k}(x; \rho_\mathsf{V})$. When V makes a query to f_j and receives a_j, save the authentication path ap_{a_j} for a_j. The final proof is

$$\pi = \Big((\mathsf{rt}_1, \ldots, \mathsf{rt}_k), (\mathsf{ap}_{a_1}, \ldots, \mathsf{ap}_{a_q}), \sigma_k\Big).$$

To verify π, the verifier recomputes $(m_1, \ldots, m_k)$ and $(\sigma_1, \ldots, \sigma_k)$ and checks that the computed σ_k matches the provided one. It then computes ρ_V the same way and simulates V with the queries provided in the proof. If verification passes, then it checks the authentication paths and accepts π if these pass too.

The efficiency of the resulting non-interactive proof π is given in the following table, where the notation $\mathcal{O}_\lambda(\cdot)$ hides any powers of the security parameter λ. This gives us a lot more intuition about how we want to optimize the parameters in the original IOP. Keeping the number of rounds k low yields better performance overall. Verification time and proof length are a lot more dependant on the number of queries, which is why we will try to keep these to a minimum.

$\lvert\pi\rvert$	t'_P	t'_V	ϵ'_s
$\mathcal{O}_\lambda(k + q \cdot \log p)$	$\mathcal{O}_\lambda(k + p) + t_\mathsf{P} + t_\mathsf{V}$	$\mathcal{O}_\lambda(k + q) + t_\mathsf{V}$	$\epsilon_s + \mathsf{negl}(\lambda)$

3.3.4 Algebraic IOPs

Most often, adding more capabilities to a proof system will make it more expressive. We can wonder though if there is anything to gain from *restricting* the proof system. This idea was already explored in § 3.2, where we considered PCPs where the prover only sends linear functions. However, this approach is not efficient for the prover who must then compute the full evaluation table for $f : \mathbb{F}^n \to \mathbb{F}$.

A similar type of restriction can be established for IOPs which we will refer to as *Algebraic* IOP*s*. This model enforces the prover's messages to be *polynomials*, which the verifier can then query at any point it desires. The verifier is then able to learn a lot more information with only a single query.

To transform an algebraic IOP into a standard IOP, we need to find an efficient encoding for polynomials, as well as an appropriate test for it. In the $\mathsf{Sum\text{-}Check}$ protocol from § 3.1.5, we simply considered that the prover sent polynomials as the full list of coefficients, which the verifier then used to obtain evaluations.

If a prover needs to send a vector $v \in \mathbb{F}^n$, it must find an appropriate representation of v as a polynomial $f_v(X)$. A simple approach would be to simply set $f_v(X) = \sum_{i=0}^{n-1} v_{i-1} X^i$, but this is not useful since it is not clear how the verifier could recover some v_i with few calls to f_v. Instead, the prover computes the *low-degree extension* of v, which is a degree n polynomial such that $f_v(i) = v_i$ for all $i \in [n]$.

In what follows, we first describe *Reed-Solomon Codes* which we will use to encode polynomials. We then present FRI, an efficient IOP for testing the correctness of these encodings. Finally, we take a deeper look at how these techniques work for low-degree extensions.

Reed-Solomon Codes A *Reed-Solomon Code* is defined by a *rate* $\rho \in [0, 1]$ and an evaluation domain $L \subseteq \mathbb{F}$. It defines a mapping from the set of polynomials of degree less than $\rho|L|$ to *codewords* in the set $\mathsf{RS}[\rho, L]$. The codeword of a polynomial $f(X)$ is described by a function $c_f : L \to \mathbb{F}$ which represents the evaluation of f over L (that is, $c_f(\ell) = f(\ell)$ for all $\ell \in L$). When the prover sends c_f, it actually sends a list of all $\{f(\ell)\}_{\ell \in L}$, where the order of the elements in L is agreed upon beforehand. We can then consider the *encoding function* for $\mathsf{RS}[\rho, L]$ as $c_f \leftarrow \mathsf{Enc}(f(X)) = \{f(\ell)\}_{\ell \in L}$, for all $f \in \mathbb{F}^{<\rho|L|}[X]$.

The distance $\Delta(c_f, c_g)$ between two Reed-Solomon codewords is taken as their relative Hamming distance

$$\Delta(c_f, c_g) = \frac{|\{\ell \in L | c_f(\ell) \neq c_g(\ell)\}|}{|L|}.$$

The codewords have the additional property that $\Delta(c_f, c_g) > 1 - \rho$ whenever $c_f \neq c_g$. More importantly though, for any $\tilde{c} : L \to \mathbb{F}$ such that $\Delta(\tilde{c}, c_f) < \frac{1-\rho}{2}$, we can uniquely decode $\tilde{c}$ as f. This property makes it an *error correcting* code, as the original polynomial can always be recovered as long as $\tilde{c}$ agrees with with some correct c_f in at least $(1 + \rho)|L|/2$ points.

An issue arrises though when we consider the BCS transformation from § 3.3.3, since the root of the Merkle tree only contains commitments to the values $f(L)$. If the verifier queries $y := f(z)$ for some $z \in \mathbb{F} \setminus L$, the prover cannot provide any authentication path for y. To solve this issue, the verifier constructs the simulated polynomial $g(X) := \frac{f(X)-y}{X-z}$ and corresponding codeword $c_g(\ell) := \frac{c_f(\ell)-y}{\ell-z}, \forall \ell \in L$. If y was wrong, then $(X - z)$ does not divide $f(X)$ and therefore $c_g \notin \mathsf{RS}[\rho, L]$ since g is a rational function. Therefore, the verifier must test that

both c_f and c_g are at least δ-close to actual codewords in $\mathsf{RS}[\rho, L]$ using the FRI protocol we present next.

3.3.5 Low-Degree Testing with FRI

The *Fast Reed-Solomon IOP of proximity (FRI)* [BBHR17] is a special type of IOP protocol, which provides a probabilistic guarantee that a given oracle $c_f : L \to \mathbb{F}$ is at most δ-far away from some codeword in $\mathsf{RS}[\delta, L]$. Following a similar construction as with Linear PCPs, this test can be run alongside an algebraic IOP to test the correctness of the oracles. In particular, if we define $D := \lceil \rho |L| \rceil$, the verifier is convinced the answers it received from the oracle were indeed evaluation of a polynomial f with degree $d < D$. For simplicity, we will assume that ρ is such that $D = \rho|L| \in \mathbb{N}$.

Before explaining the details of FRI, we can describe a naïve way the test could be performed. The verifier starts by queries c_f at D randomly chosen points in L, and can reconstruct a candidate $\tilde{f}$ using polynomial interpolation (explained in the next section). It could then request additional points of c_f and test whether if matches with $\tilde{f}$. Through the Schwartz-Zippel lemma, the probability of passing this test is $\leq (D-1)/(|L| - D)$, since $\deg(\tilde{f}) < D$, and there are $|L| - D$ remaining points to sample. This may be acceptable both in terms of number of queries and error probability when D is small and L is large, but for the general case FRI will be a lot better. In particular, it requires only $\mathcal{O}(\log d)$ oracle queries.

To understand how FRI works, we will consider $c_f : L \to \mathbb{F}$, which supposedly corresponds to the evaluation over L of a polynomial f of degree $d < D$. FRI requires that L be either an affine subspace of $\mathbb{F}$ or a multiplicative subgroup of $\mathbb{F}^\star$. We will restrict ourselves to the latter case and define $L := \langle w \rangle$, where the order of w is a power of 2, say 2^k. We can then define the sequence of domains $\{L_i\}$ such that $L_0 = L$, $L_i = L_{i-1}^2 = \langle w^{2^i} \rangle$ for $i > 0$ (and $L_i = \{1\}$ for all $i \geq k$). Note that $|L_i| = |L_{i-1}|/2 = 2^{k-i}$ and $L_i \subset L_{i-1}$ for $i \geq 1$.

The main idea behind the FRI protocol is reduce $c_f \in \mathsf{RS}[\rho, L]$ to $c'_f \in \mathsf{RS}[\rho, L']$ where $|L'| \ll |L|$, making it possible to test c'_f at all points in L'. Note that since the rate is the same, the resulting c'_f would be the codeword of a polynomial with maximum possible degree $d' = \rho|L'| \ll D$. The full protocol is described in Figure 7

In each round i of the protocol, the verifier has oracle access to $c_{i-1} : L_{i-1} \to \mathbb{F}$, and wants to check that it is a codeword in $\mathsf{RS}[\rho, L_{i-1}]$ (we set $f_0 = f$, $c_0 = c_f$ and $L_0 = L$ for the first round). V start by sending $\alpha_i \leftarrow\$ \mathbb{F}$ to P. The prover responds with $c_i : L_i \to \mathbb{F}$ which it computes as

$$c_i(\ell^2) := \frac{c_{i-1}(\ell) + c_{i-1}(-\ell)}{2} + \alpha_i \frac{c_{i-1}(\ell) - c_{i-1}(-\ell)}{2\ell}, \quad \forall\, \ell^2 \in L_i.$$

If c_{i-1} was the evaluation of a polynomial f_{i-1} with degree d_{i-1}, then c_i would be the codeword of some f_i with degree $d_i \leq d_{i-1}/2$ such that

$$f_i(X) = \mathsf{even}(f_{i-1})(X) + \alpha_i \cdot \mathsf{odd}(f_{i-1})(X).$$

The odd and even functions return a polynomial of degree at most half the original one, taken by considering only the odd/even coefficients respectively. When the prover performs this step correctly, the rate ρ is preserved since $d_i \leq \frac{d_{i-1}}{2} < \frac{1}{2}\rho|L_{i-1}| = \rho|L_i|$. The verifier is then left to check that $c_i \in \mathsf{RS}[\rho, L_i]$ which it does by repeating the above procedure.

After performing $\log(d) = r$ rounds, the resulting c_r would be constant which the verifier can easily check. It must also ensure however that the sequence of $\{c_i\}_{i=0}^r$ are consistent with each other. To do so, V samples $\mu \leftarrow\!\$\, L$ and checks

$$c_{i+1}(\mu^{2^{i+1}}) \stackrel{?}{=} \frac{c_i(\mu^{2^i}) + c_i(-\mu^{2^i})}{2} + \alpha_i \frac{c_i(\mu^{2^i}) - c_i(-\mu^{2^i})}{2\mu^{2^i}} \quad \forall\, i = 1, \ldots, r.$$

By using powers of μ, the verifier reuses the computed $c_i(\mu^{2^i})$ and only queries $c_i(-\mu^{2^i})$, setting the total number of queries for this check to $2 \cdot r$.

The total number of queries is thus $\mathcal{O}(\log d)$, and the communication complexity is $|L_1| + \cdots + |L_r| = \sum_{i=1}^r \frac{|L|}{2^i} = \mathcal{O}(|L|)$ field elements.

Given a relative distance parameter δ, the soundness error of FRI is upper bounded by $1-\delta$. This bound is conjectured [BBHR17] to be $1-\delta+(|L|/|\mathbb{F}|)^{\mathcal{O}(1)}$, but careful analyses [BGKS20, BSKR10] have only been able to show loose bounds. The soundness error can always be reduced invoking multiple iteration in parallel, in exchange for worse efficiency.

Finally, we look at the run-time of both parties. The prover computes each c_i by combining evaluations of c_{i-1} over all of L_{i-1}, for a total of $\mathcal{O}(|L|)$ operations. In each round, the verifier only performs a constant number of operations, making its total $\mathcal{O}(\log |L|)$. Together with previous figures, we obtain the following efficiency table.

rounds	queries	communication	t_{V}	t_{P}	ϵ_s
$\mathcal{O}(\log d)$	$\mathcal{O}(\log d)$	$\mathcal{O}(\lvert L\rvert)$	$\mathcal{O}(\log d)$	$\mathcal{O}(\lvert L\rvert)$	"$1-\delta$"

Low-degree Extensions With the power of FRI in hand, we take a slight detour to explain how ordinary vectors $v \in \mathbb{F}^n$ can be encoded with polynomials.

We start by considering a slightly different representation of v by viewing it as a function $v : H \to \mathbb{F}$ with $H \subseteq \mathbb{F}$ and $|H| = n$. With a given bijection $\mathsf{int} : H \to [n]$, we have $v(a) = v_{\mathsf{int}(a)}$ for all $a \in H$. For clarity, we will identify $\mathbb{F}^H$ with $\mathbb{F}^n$ and for $v \in \mathbb{F}^H$, $v_a = v_{\mathsf{int}(a)}$, for $a \in H$.

We define the *low-degree extension* of $v \in \mathbb{F}^H$ as the unique polynomial $f_v(X) \in \mathbb{F}^{<|H|}[X]$ such that $f_v(a) = v_a$, for all $a \in H$. It can be explicitly represented as

$$f_v(X) = \sum_{a \in H} L_a(X) \cdot v_a, \quad \text{where } L_{H,a}(X) = \prod_{b \in H \setminus \{a\}} \frac{X - b}{a - b}.$$

The set $\{L_{H,a}(X)\}$ is referred to as a *Lagrange basis* for H and has the property that $L_{H,a}(a) = 1$ and $L_{H,a}(b) = 0$ for all $a \neq b \in H$.

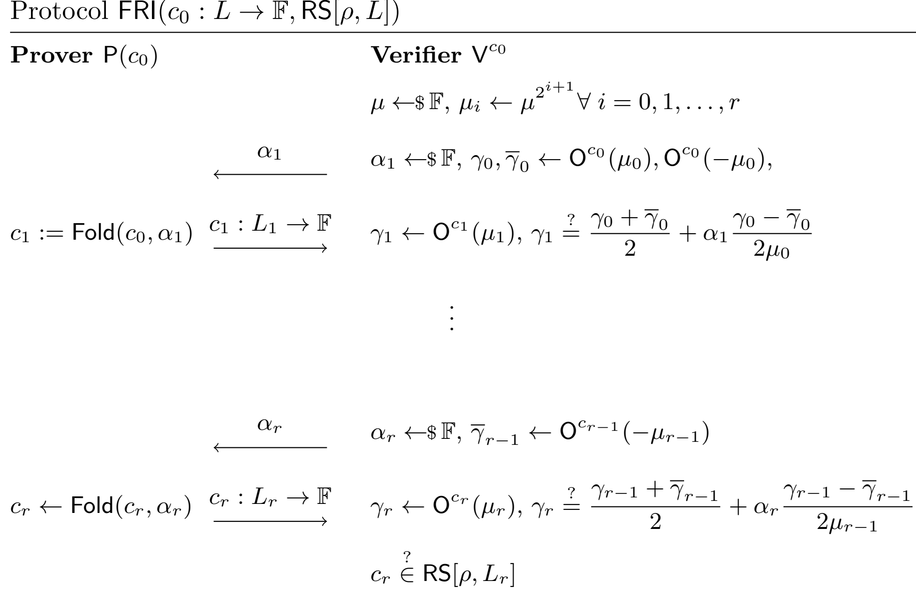


Figure 7: FRI Protocol

When we take $H = \langle w \rangle \subset \mathbb{F}^\star$ such that the order of w is a power of 2, we can evaluate $\{L_a(\tau)\}$ for any τ in $\mathcal{O}(n)$ field operations. Taking the linear combination with $\{v_a\}$ yields $f_v(\tau)$. We refer the reader to [BCG+13, Appendix E.2] for more details about this optimization.

When the prover sends $f_v(X)$ as an oracle, it must evaluate f_v over the full domain L. Since both H and L are cyclic groups whose order is a power of 2 (for L this was a requirement for FRI), we can exploit the Fast Fourier Transform.

Applying $\mathsf{FFT}_L(f)$ to a polynomial f of degree $< |L|$ returns the list of evaluations of f over L, performing the same operation as Enc for Reed-Solomon encoding. Moreover, due to the special field structure, it requires only $\mathcal{O}(|L| \log |L|)$ field operations.

The reverse operation, or *Inverse Fast Fourier Transform* is denoted IFFT_H. Given a set of points indexed by H, it returns the coefficients of a polynomial of degree $< |H|$ which interpolates these points. It also runs in $\mathcal{O}(|H| \log |H|)$.

For a vector $v \in \mathbb{F}^H$, the prover can compute its low-degree extension $f_v(X) \leftarrow \mathsf{IFFT}_H(v)$ and the corresponding RS codeword $c_{f_v} \leftarrow \mathsf{FFT}_L(f_v(X))$ in time $\mathcal{O}(|H| \log |H| + |L| \log |L|)$.

4 Aurora, Fractal, Marlin

In the two previous sections, we spent a lot of time explaining many different building blocks that can be used to construct a zkSNARK. The first step requires arithmetizing the statement being proved, yielding for example and R1CS relation. Then one must decide on what type of computation model we want between the prover and the verifier. Many recent proof systems are based on the IOP model, as it is very expressive and through appropriate tools like the BSC transform § 3.3.3 can easily be made non-interactive with minimal overhead. Moreover, the FRI protocol § 3.3.5 allows us to work in the algebraic IOP model, where polynomial evaluation is efficient for the verifier.

While there are many variations on the above "recipe", we chose to present 3 different protocols which follow these ideas. They are respectively named *Aurora* [BCR+19], *Marlin* [CHM+20], and *Fractal* [COS20], and all follow a very similar construction.

Aurora and Fractal have The rather unique property among recent proof systems of being plausible post-quantum, since they do not rely on any cryptographic assumption beyond ordinary hash function. Moreover, they are *universal*, in the sense that they do not depend on any existing trusted setup. Marlin on the other hand depends on pairings which are used to replace the Reed-Solomon/FRI paradigm.

We will start by describing Aurora which both other proof systems build upon. For Marlin, we present the concepts of *holography* and *polynomial commitments* and see how they can greatly increase performance. Finally, *proof composition* is described in Fractal, which can be thought of a "proofs of proof verification".

Algebraic IOPs for R1CS

For what follows, we recall the structure of an R1CS problem defined its instance $x = (\mathbb{F}, m, n, k, (A, B, C), v)$ and witness $w \in \mathbb{F}^{n-k}$ with $A, B, C \in \mathbb{F}^{m\times n}$ and $v \in \mathbb{F}^k$. Together, the pair (x, w) satisfies the following relation:

$$\left(\sum_{j=1}^{n} A_{i,j} z_j\right) \cdot \left(\sum_{j=1}^{n} B_{i,j} z_j\right) = \left(\sum_{j=1}^{n} C_{i,j} z_j\right) \ \forall\, i \in [m],$$

where $z = (v, w) \in \mathbb{F}^n$. Using the Hadamard entry-wise product $\circ$, it can be written more succinctly as $Az \circ Bz = Cz$. This relation can be further broken up by considering $z_M := Mz$ for $M = A, B, C$, so that

$$Az \circ Bz = Cz \iff z_A \circ z_B = z_C \text{ and } z_M = Mz,\ \forall\, M = A, B, C$$

We are now left with two simpler problems:

lincheck Given $z_M \in \mathbb{F}^m, z \in \mathbb{F}^n$, and $M \in \mathbb{F}^{m\times n}$, prove that $z_M = Mz$.

rowcheck Given $z_A, z_B, z_C \in \mathbb{F}^m$, prove that $z_A \circ z_B = z_C$

Before moving further, we make some additional simplifications which ease up notation, and facilitate the exposition. In particular, we set $k = 0$ and $m = n$ so that $v = \emptyset$, $z = w$, and $M \in \mathbb{F}^{n \times n}$ is square.

We recall that in the context of an algebraic IOP the prover can send vectors as polynomials by computing their low-degree extension. Before starting, the parties agree on a common cyclic[3] group $H \subseteq \mathbb{F}^{\star}$. The prover then computes $z, z_A, z_B, z_C \in \mathbb{F}^H$ as described, and sends to V their respective low-degree extensions $f_z, f_{Az}, f_{Bz}, f_{Cz} \in \mathbb{F}^{<n}[X]$. The verifier is now left to check the two following two polynomial relations:

lincheck For $M \in \{A, B, C\}$: $f_{Mz}(a) = \sum_{b \in H} M_{a,b} \cdot f_z(b), \quad \forall\, a \in H$.

rowcheck $f_{Az}(a) \cdot f_{Bz}(a) - f_{Cz}(a) = 0, \quad \forall\, a \in H$.

Once both these facts are checked, the verifier must also ensure the correctness of the polynomial oracles using for example FRI.

Polynomial Tips & Tricks

Before proceeding, we need to recall some basic facts about polynomials.

We define the *vanishing polynomial* for a set $H \subseteq \mathbb{F}$ as the degree $|H|$ polynomial $\mathbb{Z}_H(X)$ such that $\mathbb{Z}_H(a) = 0$ for all $a \in H$. For arbitrary H, it is defined as $\mathbb{Z}_H(X) := \prod_{a \in H}(X - a)$. When we consider H as a cyclic subgroup $H = \langle w \rangle$, $\mathbb{Z}_H$ can be computed in $\mathcal{O}(\log |H|)$ field operations by noticing the following:

$$\mathbb{Z}_H(X) = \prod_{i=0}^{|H|-1} (X - w^i) = X^{|H|} - 1.$$

The following lemma will prove very useful when we explore a variation of Sum-Check.

Lemma. *Let H be a subgroup of $\mathbb{F}^{\star}$, and let $f \in \mathbb{F}^{<|H|}[X]$. Then*

$$\sum_{a \in H} f(a) = |H| \cdot f(0).$$

Proof. Write $f(X)$ as $f(X) = \sum_{i=0}^{|H|-1} c_i X^i$, and assume $H = \langle w \rangle$ for some $w \in H$. We then have

$$\sum_{a \in H} f(a) = \sum_{a \in H} \left(\sum_{i=0}^{|H|-1} c_i a^i \right) = \sum_{i=0}^{|H|-1} c_i \cdot \left(\sum_{a \in H} a^i \right).$$

Looking more closely at the inner sum for each i, we notice

$$\sum_{a \in H} a^i = \sum_{j=0}^{|H|-1} \left(w^j\right)^i = \sum_{j=0}^{|H|-1} \left(w^i\right)^j = \begin{cases} |H|, & \text{if } i \equiv 0 \mod |H| \\ 0, & \text{otherwise.} \end{cases}$$

[3] We can also consider *affine subspaces* of $\mathbb{F}$ as is done in [BCR+19].

Therefore,

$$\sum_{a \in H} f(a) = c_0 \cdot |H| = |H| \cdot f(0).$$

□

We also recall polynomial division: Given $H \subseteq \mathbb{F}$ and a polynomial $f(X)$ of degree $d > |H|$, there exist two polynomials $g(X), h(X)$ such that

$$f(X) = h(X) \cdot \mathbb{Z}_H(X) + g(X), \text{ where } \deg g < |H| \text{ and } \deg h = d - |H|.$$

First, notice that if $f(a) = 0$ for all $a \in H$, then $g \equiv 0$. Therefore, we can state that f vanishes on H *if and only if* there exists a polynomial h of degree $d - |H|$ such that $f(X) = h(X) \cdot \mathbb{Z}_H(X)$.

4.1 Aurora

The Aurora protocol [BCR+19] combines many of the tools we have seen up till now. While we can technically describe it as an algebraic IOP, the authors use the term *Reed-Solomon Encoded* IOP since the techniques used depend heavily on the FRI low-degree test for the oracles. In particular, they directly refer to the polynomials sent by the prover as Reed-Solomon codewords, and most of the checks performed by the verifier are reduced to checking whether the oracle for the polynomial is δ-close to some codeword with predefined rate. In other words, it checks whether the degree of the polynomial is correct. This technique is at the heart of the *univariate* Sum-Check protocol which we now present.

4.1.1 Univariate Sumcheck

The Sum-Check protocol we saw in § 3.1.5 considered multivariate polynomials over $\{0,1\}^n$. Instead, we are interested in proving $\sum_{a \in H} f(a) = \gamma$ for any polynomial $f \in \mathbb{F}[X]$ of degree d, and $\gamma \in \mathbb{F}$.

In order to take advantage of the above lemma, we first decompose f as

$$f(X) = h(X)\mathbb{Z}_H(X) + (Xp(X) + \gamma/|H|) \tag{3}$$

where $\deg p < |H| - 1$ and $\deg h = d - |H|$. If we set $g(X) = Xp(X) + \gamma/|H|$, then this is simply the result of applying the polynomial division of f by $\mathbb{Z}_H$. Moreover, we have

$$\sum_{a \in H} f(a) = \sum_{a \in H} h(a)\mathbb{Z}_H(a) + g(a) = \sum_{a \in H} g(a) = |H| \cdot g(0) = \gamma.$$

In the univariate Sum-Check protocol, the verifier is given f and γ. To prove that Equation 3 holds, the prover sends p, and the verifier can then simulate queries to h by querying f, p and evaluating:

$$h(X) := \frac{f(X) - Xp(X) + \gamma/|H|}{\mathbb{Z}_H(X)}.$$

The verifier must therefore only check that $\deg(h) \leq d - |H|$ and $\deg(p) \leq |H| - 1$, which can be done once again using FRI. If the prover has malicious by either claiming a wrong sum, or submitting inconsistent polynomials, then $\mathbb{Z}_H$ would not divide $(f - Xp + \gamma/|H|)$. The function h would represent a rational function, which would be detected through the low-degree test.

Note: When working with algebraic IOPs, functions that the verifier simulates through previously sent oracles are called *virtual oracle*. To evaluate them, the verifier must simply evaluate the existing oracles and compose the queries together.

4.1.2 Rowcheck

A similar technique can be applied even more effortlessly to solve the "rowcheck" problem. Since the prover sends f_{Az}, f_{Bz}, f_{Cz} at the start of the protocol, the verifier can make sure $f_{Az} \cdot f_{Bz} - f_{Cz} \equiv 0$ by testing the virtual oracle

$$r(X) := \frac{f_{Az}(X) \cdot f_{Bz}(X) - f_{Cz}(X)}{\mathbb{Z}_H(X)}.$$

Since the claimed degrees of f_{Az}, f_{Bz}, f_{Cz} are all $\leq |H| - 1$, the numerator should have degree $\leq 2(|H| - 1)$. If the relation does hold, we would have $\deg(r) \leq |H| - 2$, which is what the verifier will check.

4.1.3 Lincheck

Recall that the "lincheck" problem checks whether $y \stackrel{?}{=} Mx$ for $x, y \in \mathbb{F}^n, M \in \mathbb{F}^{n \times n}$. Rather than checking the relation in full, we could instead verify that a random linear combination of the components of $v := y - Mx$ equals 0. Setting this linear combination to $\left(1, \alpha, \dots, \alpha^{n-1}\right)$, we can define the above problem as a polynomial instead:

$$h(X) = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{n} M_{i,j} x_j \right) X^{i-1}.$$

When the orignal matrix relation holds, h is identically 0. If that is not the case, then the Schwartz-Zippel lemma would fail with high probability, since

$$\Pr_{r \leftarrow\$ \mathbb{F}}[h(r) = 0 \mid y \neq Mx] \leq \frac{n-1}{|F|}.$$

In an IOP protocol, the verifier is given oracle access to the low-degree extension of $x, y \in \mathbb{F}^H$ as $f_x(X), f_y(X)$ which are both of degree $< n$ (V still has full access to $M \in \mathbb{F}^{H \times H}$). V starts by sampling $r \leftarrow\$ \mathbb{F}$ and sends it to P. Both parties can then construct the polynomial p_r defined over H:

$$p_r(X) = r^{\mathsf{int}(X)} \cdot f_y(X) - \left(\sum_{b \in H} r^{\mathsf{int}(b)} \cdot M_{b,X} \right) \cdot f_x(X).$$

Note that for V, p_r is a only a virtual oracle, but this is enough since the parties can now run the univariate Sum-Check protocol from § 4.1.1 to verify $\sum_{a \in H} p_r(a) \equiv 0$.

In the full protocol, 3 separate "linchecks" need to be performed, but though careful optimization it is possible to batch these together.

4.1.4 Adding Zero-Knowledge

So far, we have not yet detailed how the zero-knowledge property is achieved concretely. Most times, it requires only a few modification to an existing protocol, and does not add much overhead. We briefly present some techniques used in Aurora to achieve this goal.

First, the oracles f_{Mz} and f_z sent by the prover should not leak any information about the underlying vectors. Therefore, we forbid the verifier from querying them in any $a \in H$. This is not sufficient, since the full polynomial can always be recovered given $|H| + 1$ arbitrary points, even outside of H. Since we know from the protocol description how much queries V makes to some f_z, the prover actually sends $f'_z(X) := f_z(X) + \mathbb{Z}_H(X) \cdot R(X)$. Here, $R(X)$ is a *random* polynomial of degree b, where b is a bound on the number of queries V makes to f'_z.

The Sum-Check protocol for $\sum_H f = \mu$ can also be enhanced with zero-knowledge with only 2 extra communication rounds.

- P sends a random polynomial $q \in \mathbb{F}^{\leq |H|}[X]$, along with $\beta := \sum_{a \in H} q(a)$.
- V responds with $c \leftarrow\!\$\, \mathbb{F}$.
- P and V invoke Sum-Check for the claim $\sum_H (c \cdot f + q) = c \cdot \mu + \beta$.

A similar idea is used during the low-degree testing phase in FRI, through linearity of Reed-Solomon codes. When proving that some $f_z \in \mathsf{RS}[L, \rho]$, the prover first sends $r \in \mathsf{RS}[L, \rho]$ and the verifier responds with $s \leftarrow\!\$\, \mathbb{F}$. The test is then performed over $s \cdot f_z + r \in \mathsf{RS}[L, \rho]$.

4.2 Marlin

Marlin [CHM+20] improves upon Aurora in two distinct ways to achieve logarithmic verification time and much smaller practical proof sizes. It achieves this by preprocessing the R1CS instance and uses a different polynomial. Proofs with preprocessing are sometimes referred to as *holographic proofs*, which are proof where the verifier has access to a succinct representation of the input.

4.2.1 Holography

When a proof system is used to prove the correct execution of a specific problem, the task of converting this program into an appropriate R1CS instance can take a lot more time than running the program itself. In situations where the

verification time needs to be fast, the arithmetization can be performed ahead of time, essentially as a preprocessing step of the proof system.

This is for example the case of Zcash [BCG+14], where a specific circuit was designed to verify transactions in a private way. A highly optimized $\mathsf{R1CS}$ instances was crafted, which was used with the proof system due the Groth [Gro16].

In general, this preprocessing step is defined as a deterministic algorithm that produces a *proving* and *verification* key for the circuit. The proving key pk should provide all the information necessary to create a proof for this circuit, whereas the verification key vk is used to attest the proof is consistent with the encoding provided in pk.

In Marlin, this idea is used to speed up the "lincheck" from Aurora, where the preprocessing step involves creating succinct representations of the matrices A, B, C. If we manage to find appropriate low-degree polynomials which accurately describe these matrices, we can set the proving key pk to be these polynomials. At the start of the protocol, the verifier is given oracle access to these polynomials, and may query them in the same way as the prover's ones. The verification key can then be seen as a commitment these polynomials, and is used at the end of the protocol to ensure the that the evaluations it received and the look at correct.

While Marlin itself uses a different way of representing polynomials (which we explore in the next section), we look at the case where pk contains Reed-Solomon codewords. The key vk would then consist of the Merkle root of the tree for the codewords, and when the prover sends symbols from the polynomials in pk, it would then send the corresponding authentication paths.

4.2.2 Matrix Arithmetization

In order to describe $M \in \{A, B, C\}$ as a polynomial, we take advantage of the fact that M is often sparse since the number of non-zero entries will be similar to the number of variables.

We start by indexing the non-zero entries in M by a set $K \subset \mathbb{F}$ and define

$$\langle M \rangle \colon k \in K \mapsto (a, b, M_{a,b}) \in H \times H \times \mathbb{F}.$$

Taking the low-degree extension of each component of $\langle M \rangle(k)$, we obtain 3 polynomials $\mathsf{row}(X), \mathsf{col}(X), \mathsf{val}(X) \in \mathbb{F}^{<|K|}[X]$ such that

$$\langle M \rangle = (\mathsf{row}(k), \mathsf{col}(k), \mathsf{val}(k)) = (a, b, M_{a,b}),\ k \in K.$$

We can now write M as a bivariate polynomial:

$$f_M(X, Y) := \sum_{k \in K} L_{H,\mathsf{row}(k)}(X) \cdot L_{H,\mathsf{col}(k)}(Y) \cdot \mathsf{val}(k).$$

Since we compose L_H with row and col, the degree of the summand in k is $\mathcal{O}(|H||K|)$. Therefore, M is not a low-degree extension, whose degree would then be $< |K|$ in both X and Y. Some extra work is required to get there.

We consider the *unnormalized Lagrange polynomial* [BCG+19] which is defined as $\mathbb{Y}_H(X,Y) := \frac{\mathbb{Z}_H(X)-\mathbb{Z}_H(Y)}{X-Y}$. It has the property that $\mathbb{Y}_H(X,a) = \mathbb{Y}_H(a,X) = L_{H,a}(X) \cdot \mathbb{Y}_H(a,a)$. We use it to obtain a rational expression for $L_{H,\mathsf{row}(k)}(X)$ and $L_{H,\mathsf{col}(k)}(Y)$:

$$L_{H,\mathsf{row}(k)}(X) = \frac{\mathbb{Y}_H(\mathsf{row}(k),X)}{\mathbb{Y}_H(\mathsf{row}(k),\mathsf{row}(k))},\ L_{H,\mathsf{col}(k)}(Y) = \frac{\mathbb{Y}_H(\mathsf{col}(k),Y)}{\mathbb{Y}_H(\mathsf{col}(k),\mathsf{col}(k))}.$$

The normalizing factors $\mathbb{Y}_H(\alpha,\alpha), \mathbb{Y}_H(\beta,\beta)$ can be taken into account by val instead. We therefore consider $\mathsf{row}(X), \mathsf{col}(X), \mathsf{val}(X)$ as the low-degree extensions over K such that for all $k \in K$ and $(a,b,M_{a,b}) := \langle M\rangle(k)$:

$$\mathsf{row}(k) = a, \quad \mathsf{col}(k) = b, \quad \mathsf{val}(k) = \frac{M_{a,b}}{\mathbb{Y}_H(a,a)\cdot\mathbb{Y}_H(b,b)}.$$

We can now write the bivariate low-degree extension $f_M(X,Y)$ of M as:

$$\begin{aligned} f_M(X,Y) &= \sum_{k\in K} \mathbb{Y}_H(\mathsf{row}(k),X)\cdot\mathbb{Y}_H(\mathsf{col}(k),Y)\cdot\mathsf{val}(k) \\ &= \sum_{k\in K} \mathbb{Y}_H(\mathsf{row}(k),X)\cdot\mathbb{Y}_H(\mathsf{col}(k),Y)\cdot\mathsf{val}(k) \\ &= \sum_{k\in K} \frac{\mathbb{Z}_H(\mathsf{row}(k))-\mathbb{Z}_H(X)}{\mathsf{row}(k)-X}\cdot\frac{\mathbb{Z}_H(\mathsf{col}(k))-\mathbb{Z}_H(Y)}{\mathsf{col}(k)-Y}\cdot\mathsf{val}(k) \\ &= \sum_{k\in K} \underbrace{\frac{\mathbb{Z}_H(X)}{X-\mathsf{row}(k)}\cdot\frac{\mathbb{Z}_H(Y)}{Y-\mathsf{col}(k)}\cdot\mathsf{val}(k)}_{h_k(X,Y)}. \end{aligned}$$

Note that in each $h_k(X,Y)$, we have $\mathsf{row}(k), \mathsf{col}(k) \in H$. Therefore, h_k is not rational, and has degree $< |K|$ in both variables. By definition of val, we have $f_M(a,b) = M_{a,b}$, making $f_M(X,Y)$ the low-degree extension of M.

If a verifier is given oracle access to $\mathsf{row}, \mathsf{col}, \mathsf{val}$, it can check $f_M(a,b) \stackrel{?}{=} M_{a,b}$ for $a,b \in H$. We consider the summand of f_M as a rational function h over K, such that $h(k) = h_k(a,b)$. This results in the $\mathsf{Sum\text{-}Check}$ problem

$$\sum_{k\in K} h(k) \stackrel{?}{=} M_{a,b}.$$

Unfortunately, the univariate $\mathsf{Sum\text{-}Check}$ from § 4.1.1 cannot be used directly, since we must adapt it to the case where h is rational.

4.2.3 Rational Sumcheck

In the rational $\mathsf{Sum\text{-}Check}$ protocol, the prover has polynomials p, q (to which the verifier is given oracle access) such that $\deg(p) = d_p, \deg(q) = d_q$ and $q(k) \neq 0, \forall k \in K$. The goal of the prover is to convince the verifier that $\sum_{k\in K} f(k)/g(k) \stackrel{?}{=} \sigma$, for some $\sigma \in \mathbb{F}$.

The prover starts by computing the polynomial $f(X) \in \mathbb{F}^{<|K|}[X]$ which agrees with p/q over K. It then finds $g, h \in \mathbb{F}[X]$ with $\deg(g) \leq |K| - 2$ and $\deg(h) = d_h \leq \max\{d_p, |K| - 1 + d_q\}$, such that

$$f(X) = Xg(X) + \sigma/|K|, \quad p(X) - q(X) \cdot f(X) = h(X) \cdot \mathbb{Z}_K(K).$$

Notice that like in the univariate Sum-Check protocol, we have $\sum_{k \in K} f(k) = \sigma$.

The prover now sends g, h to V who can now compute the virtual oracle $h(X)$

$$h(X) = \frac{p(X) - q(X) \cdot (Xg(X) + \sigma/|K|)}{\mathbb{Z}_K(X)}.$$

It then must check that the degree bounds $\deg(g) \leq |K| - 2$ and $\deg(h) \leq d_h$ are satisfied.

4.2.4 Lincheck

Equipped with the rational Sum-Check protocol, we show how it can be used to to prove the "lincheck" problem when the verifier is given oracle access to $\mathsf{row}, \mathsf{col}, \mathsf{val}$ which characterize $M \in \mathbb{F}^{H \times H}$. Recall that $f_M(X, Y)$ can be defined as

$$f_M(X, Y) = \sum_{k \in K} \frac{\mathbb{Z}_H(X)}{X - \mathsf{row}(k)} \cdot \frac{\mathbb{Z}_H(Y)}{Y - \mathsf{col}(k)} \cdot \mathsf{val}(k).$$

In same spirit as the "lincheck" in Aurora § 4.1.3, the prover starts by sending the low-degree extensions f_x, f_y of $x, y \in \mathbb{F}^H$, where y is supposedly equal to Mx. The verifier then responds with $\alpha \leftarrow\$ \mathbb{F}$.

The parties must now check that the following polynomial sums to 0 over H.

$$q(X) := \mathbb{Y}_K(\alpha, X) \cdot f_y(X) - r_M(\alpha, X) \cdot f_x(X),$$

where $r_M(X, Y) := \sum_{k \in H} \mathbb{Y}_K(X, k) f_M(k, Y)$.

If we ran the univariate Sum-Check protocol here, the verifier would struggle, since evaluating the virtual oracle $r_M(\alpha, X)$ would require $\Omega(|H||K|)$ operations and oracle evaluations. The prover helps the verifier by sending it $t(X) = r_M(\alpha, X) \in \mathbb{F}^{<|H|}[X]$, and they proceed with a univariate Sum-Check for q' over H:

$$q'(X) := \mathbb{Y}_K(\alpha, X) \cdot f_y(X) - t(X) \cdot f_x(X).$$

However, the verifier must also check that the given t is correct, so it asks P for the evaluation of $t(\beta_1)$ for $\beta_1 \leftarrow\$ \mathbb{F} \setminus H$ who responds with γ [4] .

Since $t(\beta) = r_M(\alpha, \beta) = \sum_{k \in H} \mathbb{Y}_K(\alpha, k) f_M(k, \beta)$, another univariate Sum-Check would need to be performed over

$$q''(X) = \mathbb{Y}_K(\alpha, X) f_M(X, \beta).$$

Unfortunately, we have the same problem as before since the verifier will need to query f_M which requires $\mathcal{O}(|K|)$ field operations. To solve this, the

[4] the verifier will check $t(\beta_1) \stackrel{?}{=} \gamma$ at the end.

authors of Fractal [COS20, Claim 6.7] proposed a neat optimization. They noticed that if we consider a slightly different matrix $M^* \in \mathbb{F}^{H \times H}$ such that $M^*_{a,b} = M_{b,a} \cdot \mathbb{Y}_H(b,b)$, then $r_M(X,Y) \equiv M^*(X,Y)$. Therefore, the preprocessing step actually performs the arithmetization of § 4.2.2 over M^* in order to produce the polynomials $\mathsf{row}, \mathsf{col}, \mathsf{val}$. With this trick, the verifier only needs to check that $\gamma = f_{M^*}(\alpha, \beta)$, which can be done efficiently using the rational sumcheck.

The full protocol for a holographic "lincheck" is described in Figure 8.

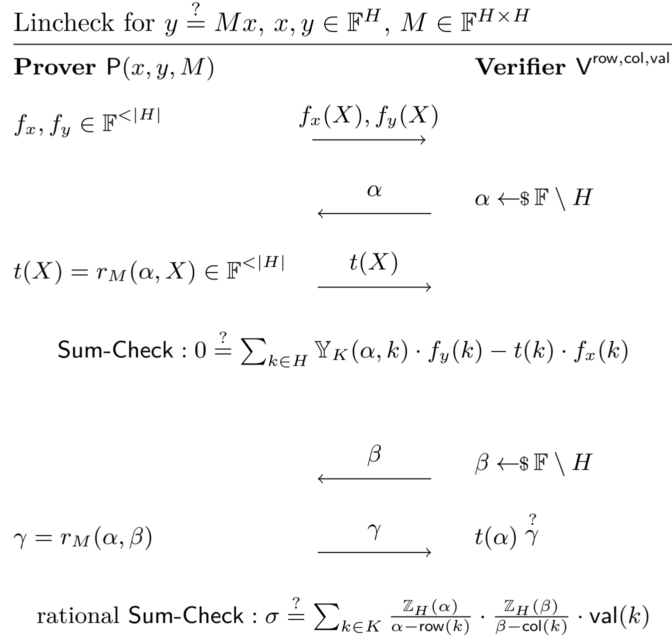


Figure 8: Holographic "lincheck"

4.2.5 Polynomial Commitments

The algebraic IOP model provides a useful layer of abstraction when designing protocols, since we do not yet need to worry about the actual representation of the polynomials sent by the prover.

In § 3.3.2, we focused on algebraic IOPs where the oracles are encoded as Reed-Solomon codes. Marlin (and many other recent proof systems) instead use *polynomial commitments* to define the actual IOP. To understand this framework, we need to look back at how an algebraic IOP is converted into an interactive protocol.

Whenever the prover sends a polynomial, it actually sends a commitment to it instead. The verifier can then query the polynomial by simply requesting the

evaluation from the prover. In order to trust that these evaluations were correct and that the polynomials were correctly described in terms of their expected degrees, the parties will need to run another protocol at the end of the algebraic IOP. At this stage, the prover provides proofs of consistency between the initial commitments and the answers it gave, and convinces the verifier the oracles were correctly encoded.

To make the comparison with the Reed-Solomon/FRI paradigm we have been using up to now, the Merkle tree roots of the RS codeword can be considered as the commitments to the polynomials. Then, the FRI protocol is run to make sure the polynomials are of correct degree and that the evaluations were correct.

The polynomial commitment scheme used in Marlin is based on the construction by Kate et al. [KZG10]. It requires a *bilinear map* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order q.

The group $\mathbb{G}_T$ is written using multiplicative notation, whereas $\mathbb{G}_1, \mathbb{G}_2$ use additive notation. For $\mathbb{G}_j = \mathbb{G}_1, \mathbb{G}_2$, the identity element of $\mathbb{G}_j$ is $[0]_j$ and it is generated by $[1]_j$. Multiplication of $[x]_j$ by $s \in \{0, 1, \dots q-1\}$ is defined using additive notation as:

$$s \cdot [x]_j = \underbrace{[x]_j + \cdots + [x]_j}_{s \text{ times}}, \text{ where } [x]_j = x \cdot [1]_j.$$

The pairing e is non-degenerate, which means that $e([1]_1, [1]_2) \neq [1]_T$. Moreover, for $s, x, y, z \in \{0, 1, \dots q-1\}$, e satisfies:

$$\begin{aligned} e([x]_1 + [y]_1, [z]_2) &= e([x]_1, [z]_2) \cdot e([y]_1, [z]_2) \\ e([x]_1, [y]_2 + [y]_2) &= e([x]_1, [y]_2) \cdot e([x]_1, [z]_2) \\ e(s \cdot [x]_1, [y]_2) &= e([x]_1, s \cdot [y]_2) = e([x]_1, [y]_2)^s. \end{aligned}$$

We define the (simplified) polynomial commitment scheme by the following algorithms:

$\mathsf{pk} \leftarrow \mathsf{Setup}(D)$ Output a secret $x \in \{0, \dots, q-1\}$, and public parameters $\mathsf{pk} = \big([1]_1, [x]_1, \dots, [x^D]_1, [1]_2, [x]_2\big)$.
For $f(X) = \sum_{i=0}^{D} a_i X^i$, we define $[f(x)]_j := \sum_{i=0}^{D} a_i [x^i]_j$ for $j = 1, 2$.

$c \leftarrow \mathsf{Commit}(\mathsf{pk}, f(X))$ Commit to $f(X) \in \mathbb{F}^{\leq D}[X]$ as $c = [f(x)]_1 \in \mathbb{G}_1$.

$\pi \leftarrow \mathsf{Open}(\mathsf{pk}, f(X), z, y)$ Create a proof of consistency between a commitment c, the polynomial f, and an evaluation $y = f(z)$ for $y, z \in \mathbb{F}$. First, compute the witness polynomial $w(X) = \frac{f(X)-y}{X-z}$ and set $\pi = [w(x)]_1$.

$\{0, 1\} \in \mathsf{Check}(\mathsf{pk}, f(X), z, y, \pi, c)$ Check the proof of opening π by verifying

$$e(c - [y]_1, [1]_2) \stackrel{?}{=} e(\pi, [x]_2 - [z]_2).$$

Output 1 if equality holds, and 0 otherwise.

We can check that the last equation holds, by observing that

$$\begin{aligned}
e(\pi, [x]_2 - [z]_2) &= e\left([w(x)]_1, [x-z]_2\right) \\
&= e\left(\left[\frac{f(x)-y}{x-z}\right]_1, [x-z]_2\right) \\
&= e\left((x-z)^{-1} \cdot [f(x)-y]_1, [x-z]_2\right) \\
&= e\left([f(x)-y]_1, (x-z)^{-1} \cdot [x-z]_2\right) \\
&= e\left(c - [y]_1, [1]_2\right).
\end{aligned}$$

Compared with the Reed-Solomon construction, the description does not mention any "low-degree test", but only a test of correct evaluation. Fortunately, all the low-degree checks the verifier had to do were for polynomials h of the form

$$h(X) = \frac{f(X) - g(X)}{\mathbb{Z}_H(X)}.$$

With a polynomial commitment scheme, the verifier instead performs a polynomial identity test by testing $f(r) \stackrel{?}{=} h(r)\mathbb{Z}_H(r) + g(r)$ for some $r \leftarrow\$ \mathbb{F} \setminus H$.

The main advantage of using this type of polynomial commitments, is that the proofs are much smaller. Indeed, after applying the non-interactive transformation, the authentication paths for the Merkle tree are replaced by a single group element. On the other hand, elliptic curve operations can be a lot slower than invoking a hash function. Proper evaluation is thus required to accurately compare both of these constructions.

4.2.6 Trusted Setup

Unfortunately, the security of the polynomial commitment scheme depends on the secret x being unknown to all participants. Since it is hard to trust a single party to safely destroy x, an MPC ceremony can be performed in which many participants can contribute randomness. It can be done so that as long as a single party has acted honestly, then nobody could have influenced the outcome.

Other proof systems like [Gro16] depend on a *circuit-specific* setup, rendering it impractical for general purpose proofs. In contrast, IOPs based on FRI do not rely on this kind of setup and are sometimes referred to as *transparent*.

4.3 Fractal

Fractal [COS20] builds upon both Aurora and Marlin, by using the low-degree testing framework from Aurora, and the preprocessing techniques in Marlin. Therefore it achieves both sub-linear verification *and* plausible post-quantum security without any trusted setup.

The algebraic IOP step is essentially the same as the one described in § 4.2, since we already incorporated the optimization proposed by Fractal. We focus on the recursive proof composition property instead.

4.3.1 Recursion

Suppose we are interested in the result of composing a function $F : \mathbb{F}^n \to \mathbb{F}^n$ T times. That is, we start with $z_0 \in \mathbb{F}^n$, and then iteratively compute $z_i = F(z_{i-1}) \in \mathbb{F}^n, i \geq 1$ We then want to prove that a given z_T is the result of having correctly computed $F^T(z_0)$. Naïvely generating a proof for this type of statement would be undesirable, since the circuit for F^T would be T times that of F.

Instead, we apply the idea of *incrementally verifiable computation* [Val08] where a proof π_i is generated at every iteration of F, simultaneously proving that $z_i = F(z_i)$ for some z_{i-1}, and that there exists a valid proof π_{i-1} attesting that $z_{i-1} = F^{i-1}(z_0)$. We can therefore consider that π_T proves that $z_T = F^T(z_0)$, without needing access to any previous intermediary results or proofs.

More concretely, for each iteration i, we define an NP instance $x_i = (F, i, z_i)$, and the corresponding witness $w_i = (z_{i-1}, \pi_{i-1})$. The predicate function $\mathsf{R}(x_i, w_i)$ returns true when $z_i = F(z_{i-1})$ and $\mathsf{acc} \leftarrow \mathsf{V}(F, x_{i-1}, \pi_{i-1})$, where $x_{i-1} = (F, i-1, z_{i-1})$. The prover then creates the proof π which proves that $\mathsf{R}(x_i, w_i) = \mathsf{true}$.

We note that it is essential that the proofs π_i are *SNARK*s. Indeed, if it is not succinct and $|\pi_i|$ was linear in the size of statement (x_i, w_i) being proved, then its size grow exponentially. Since π is an argument of knowledge, we can trust that each proof was generated by a computationally bounded party who actually knew the previous value and proof. By induction, it guarantees that the full chain is correct.

If we take a deeper look at the last step, we notice that the circuit which computes R needs to evaluate the function F provided as input. Therefore, R will need to be designed so that it is capable of simulating arbitrary circuits. Theoretically, this type of simulation would only incur a constant multiplicative increase in run-time over running the original circuit. In practice though, this would be highly inefficient.

To solve this, we consider a slightly different construction which exploits the preprocessing power of the SNARK. Rather than providing the function F to R at each iteration, we can instead directly embeds F in order to check that $z_i = F(z_{i-1})$. The next part of the circuit for R must then run the SNARK verifier to check the proof $\mathsf{V}(\mathsf{vk}, (\mathsf{vk}, z_{i-1}), \pi_{i-1})$ and check that it accepts. Here, vk is the verification key resulting from preprocessing R.

More concretely, it would be setup as follows:

preprocessing Construct a circuit for the relation $\mathsf{R}(x_i, w_i)$ where $x_i = (\mathsf{vk}, z_i)$ and $w_i = (\pi_{i-1}, z_{i-1})$. In particular, R checks that $z_i = F(z_{i-1})$ and $\mathsf{acc} = \mathsf{V}(\mathsf{vk}, (\mathsf{vk}, z_{i-1}), \pi_{i-1})$. Perform the preprocessing on R to obtain the verification/proving keys pk, vk.

proving Run the prover P for the circuit R and return $\pi_i \leftarrow \mathsf{P}(\mathsf{pk}, x_i, w_i)$.

verification Run the verifier V for the circuit R and check $\mathsf{acc} = \mathsf{V}(\mathsf{vk}, x_i; \pi_i)$. The verifier is convinced that the prover knew a valid pair (z_{i-1}, π_{i-1}).

The big obstacle we need to overcome with this construction lies in the creation of the circuit R which has to include a description of the circuit for V. While a full description would be too long, we mention one interesting aspect that must be taken into account: the hash function function used to instantiate the random oracle. In a non-recursive SNARK, we would usually chose a hash function such as SHA-2 or BLAKE2 since these result in better performance. Unfortunately, representing these in a circuit is highly inefficient since they rely mostly on binary arithmetic and we would therefore have to consider each bit as one field element. Instead, using an algebraic hash function allows us to work with field elements directly, and allow us to work with Merkle trees more efficiently.

4.4 Evaluation

By choosing to study Aurora, Marlin, and Fractal, we are able to accurately compare the impact of both preprocessing and the use of different polynomial commitment schemes. Our baseline for this will be Fractal. Comparing it to Aurora should show the benefits of preprocessing since they are both based on algebraic IOPs using the Reed-Solomon encoding and FRI. Since the algebraic IOP in Fractal and Marlin is essentially the same, their differences lie mainly in the polynomial commitments.

Asymptotic Performance In the following table, we describe the asymptotic efficiency of the zkSNARKs produced by the three proof systems. We define n as the number of inputs for the circuit, and m as the number of non-zero entries in the matrices A, B, C. Since these are usually sparse in practice, m will be similar to the number of rows. For more detailed figures, we refer the reader to the original papers.

	rounds	proof length	queries	t_{P}	t_{V}
Aurora	$\mathcal{O}(\log n)$	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(m + n \log n)$	$\mathcal{O}(m)$
Fractal	$\mathcal{O}(\log m)$	$\mathcal{O}(\log^2 m)$	$\mathcal{O}(\log m)$	$\mathcal{O}(m \log m)$	$\mathcal{O}(\log^2 m)$
Marlin	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(m \log m)$	$\mathcal{O}(\log m)$

The first three columns are tightly linked, since we considered provers who send a constant number of polynomials per round, to which the verifier makes a constant number of queries. Recalling the BCS transformation from § 3.3.3, the proof must include commitments to all the prover's polynomials as well as all the answers to the verifier's queries accompanied by opening proofs. The run-times for the prover in Fractal and Marlin is the same as the time required to pre-process the R1CS matrices, and so we omit the latter. Moreover, we note that the resulting super-linear run-time of the prover is mostly due to the Fast Fourier transform that is required to obtain low-degree extensions of vectors.

Looking at the first two rows, we see that by leveraging preprocessing, Fractal achieves logarithmic verification time instead of linear time for Aurora. This

enables the more efficient "lincheck" where the verifier is no longer required to iterate over all entries of the matrices A, B, C. However, the efficiency of Fractal depends on m rather than n, which means it may actually perform worse for circuits where the number of constraints is a lot greater than the number of variables.

Finally, we can observe in the two last rows that using a pairing based polynomial commitment should result in much smaller proofs since opening proofs are single group elements, rather that $\log |L|$-sized authentication paths. However, Marlin and Fractal have very similar run-times, though it is hard to compare concretely since operations over elliptic curves will generally be a lot slower than simple field operations and hashes.

Evaluation While asymptotic figures are helpful, we also need to look at concrete performance in order to get an accurate comparison. To do so, we use the open-source libiop [SCI] and arkworks [CHM+] libraries provided by the authors of the papers, written in C++ and Rust respectively.

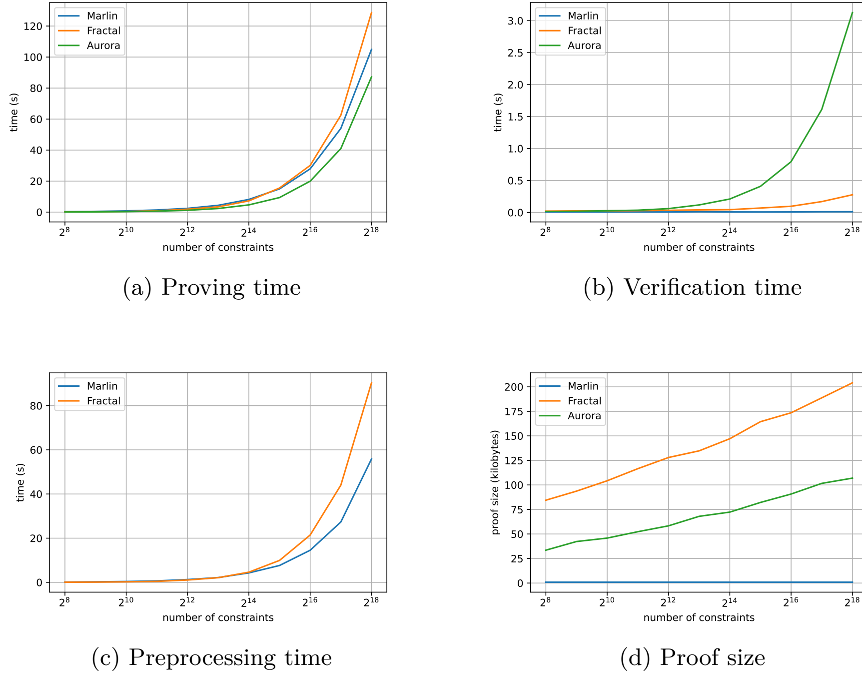
For the Marlin benchmark, we use the base field of the BLS12-381 elliptic curves, and ignore the cost of running the trusted setup. For Aurora and Fractal, the default field size is 181 bits, and we set it to use the conjectured FRI soundness upper-bound (this result in fewer necessary repetitions). We considered randomly generated circuits in n variables and constraints, such that $m \sim n$, and n ranges from 2^8 to 2^{18}. We present our results in Figure 9 which include the timings for proving, verification, and preprocessing statements as well as resulting proof sizes for the 3 different zkSNARKs.

While the verification time for Fractal is indeed a lot faster than with Aurora, Fractal actually performs worse in terms of all other metrics. Surprisingly though, the proving time of Aurora turned out the be the fastest. Looking at proof sizes, the constant size (880 bytes) proofs of Marlin are impressive, but those of Aurora and Fractal are still very reasonable at only a few hundred kilobytes.

5 Conclusion

Only a few years ago, proof systems like those we presented in this report would have seemed inimaginable. Interestingly though, most of the concepts they are based on have been known for many decades, and come from many different areas of research, including complexity theory, information theory, and cryptography. A good background in these fields is a necessary prerequisite to start learning about recent zkSNARK constructions, making them harder to approach.

With this report, the goal was to explain zkSNARKs "from the ground up" and explain all the steps and concepts involved along the way. Moreover, we wanted to motivate these with examples and intuition as a way of "filling in the gaps" one might encounter when reading more recent publication. Indeed, many important aspects of proof systems lie in tiny details, such as the use of FFTs, or the use oracles in protocols, or even the some elementary facts about polynomials. These details can be brought to light when they are implemented in software,



(a) Proving time

(b) Verification time

(c) Preprocessing time

(d) Proof size

Figure 9: Evaluation of Aurora, Marlin, and Fractal using libiop and arkworks. Run on Intel 2.6 GHz CPU, 16GB RAM, multi-threading disabled.

which provides another way of developing intuition about the system as a whole. In fact, this link between theory and implementation is also understanding the systems as a whole, since asymptotic performance may be misleading at times.

Thankfully, the amount of research surrounding zkSNARKs does not seem to be slowing down, and we look forward to studying new improvements and ideas that make zkSNARKs more efficient. One particular interesting research direction being explored is finding linear-time provers, which requires generalizing algebraic IOPs to *tensor* IOPs. Circuits themselves are also being thoroughly studied, and are now being designed in similar ways to modern CPUs. Moreover, concepts like incrementally verifiable computation (or even more generally *proof carrying data*) open the door for many new interesting paradigms where SNARKs could be used.

References

[ALM$^+$92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *33rd FOCS*, pages 14–23. IEEE Computer Society Press, October 1992.

[AS92] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *33rd FOCS*, pages 2–13. IEEE Computer Society Press, October 1992.

[BBHR17] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. *Electron. Colloquium Comput. Complex.*, 24:134, 2017.

[BCG$^+$13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg, August 2013.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[BCG$^+$19] Eli Ben-Sasson, Alessandro Chiesa, Lior Goldberg, Tom Gur, Michael Riabzev, and Nicholas Spooner. Linear-size constant-query IOPs for delegating computation. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 494–521. Springer, Heidelberg, December 2019.

[BCI$^+$13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

[BCR$^+$19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Heidelberg, October / November 2016.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014.

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 390–420. Springer, Heidelberg, August 1993.

[BGKS20] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: sampling outside the box improves soundness. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 5:1–5:32. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[BSKR10] Eli Ben-Sasson, Swastik Kopparty, and Jaikumar Radhakrishnan. Subspace polynomials and limits to list decoding of reed–solomon codes. *IEEE Transactions on Information Theory*, 56(1):113–120, 2010.

[CHM+] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. arkworks-marlin.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Heidelberg, May 2020.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short pcps. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*, pages 278–291, 2007.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

[KR08] Yael Tauman Kalai and Ran Raz. Interactive PCP. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 536–547. Springer, Heidelberg, July 2008.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[Lev73] L. A. Levin. Universal problems of full search. *Probl. Peredachi Inf.*, 9(3):115–116, 1973.

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st FOCS*, pages 2–10. IEEE Computer Society Press, October 1990.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, August 1988.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.

[SCI] SCIPR Lab. libiop: a c++ library for iop-based zksnarks.

[Tha17] Justin Thaler. The sum-check protocol. `http://people.cs.georgetown.edu/jthaler/sumcheck.pdf`, September 2017. [Online; accessed 10-July-2021].

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008.