polygon zkEVM

Technical Document

eSTARK: Extending the STARK Protocol with Arguments

v.1.0

February 10, 2023

Contents

1 Preliminaries

1.1 Notation

Whenever is possible, we denote quantities with upper-case latters and denote either polynomials' degree with lower-case letters.

We denote by $\mathbb{F}$ to a finite field of prime order p and $\mathbb{F}^*$ to its respective multiplicative group and define k to be the biggest non-negative integer such that $2^k \mid (p-1)$. We also write $\mathbb{K}$ to denote a finite field extension of $\mathbb{F}$, of size p^e, $e \geq 2$. Furthermore, we write $\mathbb{F}[X]$ (resp. $\mathbb{K}[X]$) for the ring of polynomials with coefficients over $\mathbb{F}$ (resp. $\mathbb{K}$) and write $\mathbb{F}_{<d}[X]$ (resp. $\mathbb{K}_{<d}[X]$) to denote the set of polynomials of degree lower than d.

Although all the protocols presented in this article work over any prime order, we fix our attention to fields that contain a multiplicative subgroup of size a large power of two. This restriction is crucial in order to achieve a practical protocol.

Given a cyclic subgroup S of $\mathbb{F}^*$, we denote by $L_i \in \mathbb{F}_{<|S|}[X]$ the i-th *Lagrange polynomial* for S. That is, L_i satisfies $L_i(g^i) = 1$ and $L_i(g^j) = 0$ for $j \neq i$, where g is used here to denote the generator of S. Moreover, we denote by $Z_S(X) = X^{|S|} - 1 \in \mathbb{F}[X]$ to the polynomial that vanishes only over S and call it the *vanishing polynomial* over S.

We denote by G to a cyclic subgroup of $\mathbb{F}^*$ with order n satisfying $n \mid 2^k$ and $1 < n < 2^k$, and let $g \in \mathbb{F}$ denote the generator of G. Similarly, we denote by H to a nontrivial coset of a cyclic subgroup of $\mathbb{F}^*$ with order m satisfying $m \mid 2^k$ and $n < m$.

Given a set of polynomials $p_1, p_2, \ldots, p_N \in \mathbb{K}[X]$ we denote by $\mathrm{MTR}(p_1, \ldots, p_N)$ to the Merkle root obtained after computing a Merkle tree [Mer88] whose leafs are the evaluations of $p_1, \ldots, p_N$ over the domain H. Additionally, given a set of elements $x_1, x_2, \ldots, x_N \in \mathbb{K}$, we also use $\mathrm{MTP}(x_1, \ldots, x_N)$ to denote the Merkle tree path (i.e., the Merkle proof) computed from the leaf containing these elements. If $X = \{x_1, \ldots, x_N\}$, then we use $\mathrm{MTP}(X)$ as a shorthand for $\mathrm{MTP}(x_1, \ldots, x_N)$.

Finally, in the description of the protocols, we use $\mathcal{P}$ to denote the prover entity and $\mathcal{V}$ to denote the verifier entity.

1.2 Interactive Oracle Proofs and STARKs

In this section we provide the definition of a polynomial IOP [BCS16] and the standard security notions associated with this model. Moreover, we introduce a popular polynomial IOP family of protocols known as STIK [BBHR19] and explain how a STIK can be compiled into a STARK.

Definition 1 (Polynomial IOP)**.** Given a function F, a (public-coin) *polynomial interactive oracle proof (IOP)* for F is an interactive protocol between two parties, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$, that comprises k rounds of interaction. $\mathcal{P}$ is given an input w and both $\mathcal{P}$ and $\mathcal{V}$ are given a common input x. At the start of the protocol, $\mathcal{P}$ provides to $\mathcal{V}$ a value y and claims to him the existence of a w satisfying $y = F(x, w)$.

In the i-th round, $\mathcal{V}$ sends a uniformly and independently random message α_i to $\mathcal{P}$; then $\mathcal{P}$ replies with a message of one of the two following forms: (1) a string m_i that $\mathcal{V}$ reads in full, or (2) a polynomial f_i that $\mathcal{V}$ can query (via random access) after the i-th round of interaction. At the end of the protocol, $\mathcal{V}$ outputs either accept or reject, indicating whether $\mathcal{V}$ accepts $\mathcal{P}$'s claim.

The security notions for IOPs are similar to the security notions of another preceding models (e.g., interactive proofs).

Definition 2 (Completeness and (Knowledge) Soundness)**.** We say that a polynomial IOP has *perfect completeness* and *soundness* error at most ε_s if the following two conditions hold.

- **Perfect Completeness**. For every x and every prover $\mathcal{P}$ sending a value y satisfying the polynomial IOP claim at the start of the protocol, it holds that:

$$\Pr\left[\mathcal{V}(x, \mathcal{P}(x, w)) = \mathsf{accept}\right] = 1,$$

  where $\mathcal{V}(x, \mathcal{P}(x, w))$ denotes the $\mathcal{V}$'s output after interacting with the prover on input x. The probability is taken over the internal randomness of $\mathcal{V}$.

- **Soundness**. For every x and every prover $\mathcal{P}^*$ sending a value y at the start of the protocol, if it holds that:

$$\Pr\left[\mathcal{V}(x, \mathcal{P}^*(x, w)) = \mathsf{accept}\right] \geq \varepsilon_s,$$

  then y satisfies the polynomial IOP claim.

If the next condition holds as well, we say that the polynomial IOP has *knowledge soundness* error at most ε_{ks}.

- **Knowledge Soundness**. There exists an algorithm $\mathcal{E}$, known as the *knowledge extractor*, that for every x and every prover $\mathcal{P}^*$ sending a value y at the start of the protocol, if it holds that:

$$\Pr\left[\mathcal{V}(x, \mathcal{P}^*(x, w)) = \mathsf{accept}\right] \geq \varepsilon_{ks},$$

  then $\mathcal{E}\left(x, \mathcal{P}^*(x, w)\right) = w$ and $y = F(x, w)$.

In words, soundness guarantees that a malicious prover cannot suceed with probability grater than ε_s on the "existence" claim of w; whereas knowledge soundness guarantees that a malicious prover cannot suceed with probability greater than ε_{ks} claiming "possession" of w satisfying $y = F(x, w)$. A polynomial IOP satisfying knowledge soundness is known as a *polynomial IOP of knowledge*.

Now we introduce the definition of a scalable and transparent IOP of knowledge (STIK) as per Definition 2 in [BBHR19].

Definition 3 (STIK)**.** A *scalable transparent IOP of knowledge (STIK)* is a polynomial IOP with knowledge soudnness ε_{ks} that satisfies the following properties:

- **Transparent.** In their design, the only cryptographic assumption is the existence of a family of collision resistant hash functions.

- **Doubly Scalable.** The verifier runs in time $\mathcal{O}(\log(n))$ and the prover runs in time $\mathcal{O}(n \log(n))$, where n informally denotes the size of the computation F.

When STIKs get instantiated for practical deployment, they result in protocols in which the prover is assumed to be computationally bounded. Protocols of such kind are known as *argument systems* (in contrast of proof systems), and consequently, the instatiation of a STIK results in a scalable transparent argument of knowledge (STARK).

Definition 4 (STARK)**.** A *scalable transparent argument of knowledge (STARK)* is a realization of a STIK by means of a family of collision resistant hash functions. More specifically, polynomials (in the form of oracles) sent from the prover to the verifier in the underlying polynomial IOP are substituted by Merkle trees; and whenever the verifier asks queries a polynomial f at v, the prover answers with $f(v)$ together with the Merkle path associated with it.

The Fiat-Shamir heuristic [FS87] can compile a STIK into a non-interactive argument of knowledge in the random oracle model, and therefore a realization of a non-interactive STIK is called a *non-interactive STARK*. In fact, non-interactive STARK are widely used in practise.

1.3 FRI

Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRI) [BCI+20] is a protocol for proving that a function $f\colon H \to \mathbb{F}$ is close to a polynomial of low degree d. Here, by low degree we mean that $d \ll |H|$. The FRI protocol consists in two phases. In the first phase, known as the *commit phase*, the prover commits to (via Merkle trees) a series of functions generated from f and from random elements $v_0, v_1, \dots$ from $\mathbb{K}$ provided by the verifier at each round. Then, in the second phase, known as the *query phase*, the prover provides a set of evaluations of the previously committed functions at a point randomly chosen by the verifier. A sufficiently large repetitions of the query phase guarantees the soundness of FRI. Following, we provide more details about how each phase works.

The Commit Phase

Let's denote by p_0 the function f of interest and assume for the simplciity of the exposition that the prover is honest (i.e., p_0 is a polynomial of low degree). In the commit phase, the polynomial p_0 is split into two other polynomials $g_{0,1}, g_{0,2}\colon H^2 \to \mathbb{K}$ of degree lower than $d/2$. These two polynomials satisfy the following relation with p_0:

$$p_0(X) = g_{0,1}(X^2) + X \cdot g_{0,2}(X^2). \tag{1}$$

Then, the verifier sends to the prover a uniformly sampled $v_0 \in \mathbb{K}$, and asks the prover to commit to the polynomial:

$$p_1(X) := g_{0,1}(X) + v_0 \cdot g_{0,2}(X).$$

Note that p_1 is a polynomial of degree less than $d/2$ and the commitment of p_1 is not over H but over $H^2 = \{x^2 \colon x \in H\}$, which is of size $|H|/2$.

The prover then continues by splitting p_1 into $g_{1,1}$ and $g_{1,2}$ of degree lower than $d/4$, then constructing p_2 with a uniformly sampled $v_1 \in \mathbb{K}$ sent by the verifier. Again, p_2 is of degree $d/2^2$ and committed over $H^{2^2} = \{x^2 \colon x \in H^2\}$, whose size is $|H|/2^2$. The whole derivation of p_{i+1} from p_i is often known as *split-and-fold* due to the prover splitting the initial polynomial into two and then folding it into one using a random value.

This process is repeated a total of $k = \log_2(d)$ times, point at which $\deg(p_k) = 0$ and the prover sends the constant p_k in plain to the verifier.

$$\begin{array}{lcr}
\mathcal{P}(p_0, d, H, \mathbb{F}, \mathbb{K}) & & \mathcal{V}(d, H, \mathbb{F}, \mathbb{K}) \\
\hline
 & \xrightarrow{\mathrm{MTR}(p_0)} & \\
 & \xleftarrow{v_0} & \\
 & \xrightarrow{\mathrm{MTR}(p_1)} & \\
 & \xleftarrow{v_1} & \\
 & \vdots & \\
 & \xleftarrow{v_{k-1}} & \\
 & \xrightarrow{p_k} &
\end{array}$$

The Query Phase

In the query phase, the verifier sends a uniformly sampled $r \in H$ to the prover and queries the evaluations $p_0(r)$, $p_0(-r)$ and $p_1(r^2)$. From $p_0(r)$ and $p_0(-r)$ the verifier computes $p_1(r^2)$ and checks that the computed value matches with the third value $p_1(r^2)$ provided by the prover. To obtain $p_1(r^2)$ from $p_0(r)$ and $p_0(-r)$, the verifier first solves the following system of linear equations for $g_{0,1}(r^2), g_{0,2}(r^2)$:

$$\begin{aligned} p_0(r) &= g_{0,1}(r^2) + r \cdot g_{0,2}(r^2), \\ p_0(-r) &= g_{0,1}(r^2) - r \cdot g_{0,2}(r^2), \end{aligned}$$

and then computes:

$$p_1(r^2) = g_{0,1}(r^2) + v_0 \cdot g_{0,2}(r^2).$$

The verifier continues by querying for $p_1(-r^2)$ and $p_2(r^4)$. From $p_1(r^2)$ and $p_1(-r^2)$ computes $p_2(r^4)$ as before and checks that the computed value is consistent with $p_2(r^4)$. Each step locally checks the consistency between each pair (p_i, p_{i+1}). The verifier continues in this way until it reaches the value of the constant p_k. The verifier checks that the value sent by the prover is indeed equal to the value that the verifier computed from the queries up until p_{k-1}. To fully ensure correctness, the prover must accompany the evaluations that he sends with a claim of their existence (via Merkle tree paths).

$$\begin{array}{lcr} \mathcal{P}(p_0, d, H, \mathbb{F}, \mathbb{K}) & & \mathcal{V}(d, H, \mathbb{F}, \mathbb{K}) \\ \hline & \vdots & \\ & \xrightarrow{\qquad\qquad p_k \qquad\qquad} & \\ & \xleftarrow{\qquad\qquad r \qquad\qquad} & \\ & \xrightarrow[\{\mathrm{MTP}(p_0(r)), \ldots, \mathrm{MTP}(p_{k-1}(r^{2^{k-1}}))\}]{\{p_0(r), \ldots, p_{k-1}(r^{2^{k-1}})\}} & \\ & \xrightarrow[\{\mathrm{MTP}(p_0(-r)), \ldots, \mathrm{MTP}(p_{k-1}(-r^{2^{k-1}}))\}]{\{p_0(-r), \ldots, p_{k-1}(-r^{2^{k-1}})\}} & \end{array}$$

Upon the completion of this process, the verifier has a first confirmation that the polynomials committed in the commit phase $p_0, p_1, \ldots, p_k$ are consistent each other.

Finally, and in order to achieve the required bounds for the soundness of the protocol, both the query phase and the batched consistency check is repeated multiple times. We give the specific soundness bound of a more generic version of FRI in Theorem 1. The full description of the transcript can be found in Figure 5a of Appendix A.1.

The Batched FRI Protocol

In this version of FRI, the prover wants to prove closeness to low degree polynomials of a set of functions f_0,f_1,…,$f_N : H \to \mathbb{F}$ at once. Obviously, we could run the FRI protocol for every function f_i in parallel, but there is a more efficient way proposed in Section 8.2 of [BCI+20]. In the batched FRI protocol, the prover instead applies the FRI protocol directly to a random linear combination of the function f_i. More specifically, assuming the prover has committed to functions $f_0, f_1, \ldots, f_N$ and the verifier has sent a uniformly sampled value $\varepsilon \in \mathbb{K}$, the prover computes the funtion:

$$f(X) := f_0(X) + \sum_{i=1}^{N} \varepsilon^i f_i(X), \tag{2}$$

and applies the FRI protocol to it.

Remark 1. In [BCI⁺20], they compute f as $f_0(X) + \sum_{i=1}^{N} \varepsilon_i \cdot f_i(X)$ instead, i.e., they use a random value $\varepsilon_i \in \mathbb{K}$ per function f_i instead of powers of a single one ε. Even if secure, the soundness bound of this alternative version is linearly increased by the number of functions N, so we might assume from now on that N is sublinear in $|\mathbb{K}|$ to ensure the security of protocols.

The Batched Consistency Check

As an extra check in the batched version, the verifier needs to ensure the correct relationship between functions $f_0, \dots, f_N$ and the first FRI polynomial $p_0 = f$. The verifier will use the evaluations of p_0 it received from the prover in each FRI query phase invocation. To allow for this check, the prover also sends the evaluations of functions $f_0, f_1, \dots, f_N$ at both r and $-r$ so that the verifier is able to check that:

$$p_0(r) = f_0(r) + \sum_{i=1}^{N} \varepsilon^i f_i(r),$$
$$p_0(-r) = f_0(-r) + \sum_{i=1}^{N} \varepsilon^i f_i(-r),$$

i.e., a local consistency check between $f_0, \dots, f_N$ and p_0. Clearly, the prover accompanies the new sent evaluations with their respective Merkle tree path. The full description of the transcript can be found in Figure 5b of Appendix A.1.

Similarly to the non-batched FRI protocol, both the query phase and the batched consistency check is repeated multiple times to ensure the protocol is sound. More precisely, the soudness error is as shown in the following theorem.

Theorem 1 (Batched FRI Soundness, [BCI⁺20], Theorems 7.2,8.3)**.** *Let $f_0, f_1, \dots, f_N \colon H \to \mathbb{K}$ be a batch of functions defined over H and let $m \geq 3$ be the Johnson bound. Suppose a batched FRI prover that interacts with a batched FRI verifier causes it to accept with probability greater than:*

$$\varepsilon_{\mathsf{FRI}} = \varepsilon_{\mathsf{C}} + (1-\theta)^s = N \cdot \frac{(m+\frac{1}{2})^7}{3\rho^{3/2}} \cdot \frac{|H|^2}{|\mathbb{K}|} + N \cdot \frac{(2m+1)\cdot(|H|+1)}{\sqrt{\rho}} \cdot \frac{\sum_{i=0}^{k-1} a_i}{|\mathbb{K}|} + (1-\theta)^s,$$

where $a_i = |H^{2^i}|/|H^{2^{i+1}}|$ is the ratio between consecutive prover messages in the commit phase, $\rho = |G|/|H|$ is the rate of the code, ε_{C} is the soundness error for the commit phase, $\theta = 1 - \sqrt{\rho}(1 + \frac{1}{2m})$ is the proximity parameter (equivalently, $\sqrt{\rho}(1+\frac{1}{2m})$ is the soundness error of one iteration of the query phase) and s is the number of queries.

Then, functions $f_0, f_1, \dots, f_N$ are θ-close to polynomials of degree lower than n.

Example 1. Say we have $p = 2^{64} - 2^{32} + 1$, $|\mathbb{K}| = p^3 \approx 2^{192}$, $|G| = 2^{23}$ and $|H| = 2^{24}$. If we set $m = 3$ and $N = 2^8$, then we have:

$$\varepsilon_{\mathsf{C}} < 2^8 \cdot \frac{2^{14}}{2 \cdot 2^{-3/2}} \cdot \frac{2^{48}}{2^{192}} + 2^8 \cdot \frac{2^{11} \cdot (2^{25})}{2^{-1/2}} \cdot \frac{2^{23}}{2^{192}} < 2^{-120},$$

Now, by obtaining $1 - \theta = 2^{-1/2} \cdot (1 + \frac{1}{6}) \approx 0.825$ and by setting $s = 400$ we have $(1-\theta)^s < 2^{-120}$, so the total FRI error is bounded by:

$$\varepsilon_{\mathsf{FRI}} = \varepsilon_{\mathsf{C}} + (1-\theta)^s < 2^{-120}.$$

In [BBHR18] it is shown that for the batched FRI protocol to achieve security parameter λ (i.e., $\varepsilon_{\mathsf{FRI}} \leq 2^{-\lambda}$), we need to use at least $s \geq \lambda / \log_2 \rho^{-1}$ many queries, and Theorem 1 shows that if $|\mathbb{K}| \gg |H|^2$ then $s \approx 2\lambda / \log_2 \rho^{-1}$.

FRI as a Polynomial Commitment Scheme

Although FRI has a different setting, it can be converted to a polynomial commitment scheme (for a definition, see [KZG10]) without much overhead. The scheme is based on the following claim: if $f \in \mathbb{F}[X]$ is a polynomial of degree lower than d, then $f(z)$ is the evaluation of f at the point z if and only if $f(X) - f(z) = (X - z) \cdot q(X)$, where $q \in \mathbb{F}[X]$ is some polynomial of degree lower than $d - 1$.

The technique, originally explained in [VP19], works as follows.

1. As with FRI, given a function $f \colon H \to \mathbb{F}$, the prover's commitment is defined as the evaluations of f over H.

2. The verifier uniformly samples a challenging point $z \in \mathbb{K} \backslash H$ at which he asks the prover to compute and send the evaluation of f.

3. The prover outputs $\widetilde{f(z)}$ along with a FRI proof π_{FRI} that the function:

$$q(X) := \frac{f(X) - \widetilde{f(z)}}{X - z},$$

is close to some polynomial of degree lower than $d - 1$.

4. If FRI passes, the verifier is convinced with high probability that the prover committed, in the first step, to a polynomial f of degree lower than d and that $\widetilde{f(z)} = f(z)$.

$$\begin{array}{lcr}
\mathcal{P}(f, d, H, \mathbb{F}, \mathbb{K}) & & \mathcal{V}(d, H, \mathbb{F}, \mathbb{K}) \\
\hline
 & \xrightarrow{\quad f \quad} & \\
 & \xleftarrow{\quad z \quad} & \\
 & \xrightarrow{\widetilde{f(z)}, \pi_{\mathrm{FRI}}} & \\
\end{array}$$

In [VP19], the authors prove that this scheme satisfies the standard notions of security related to polynomial commitment schemes: correctness, polynomial binding and evaluation binding.

1.4 Vanilla STARK

In this section, we review the STARK generation procedure from [Sta21] as applied to a particular statement.

Constraints and Trace

Let's say we want to generate a STARK for the following statement:

"I know some $a, b, c, d, e \in \mathbb{F}^n$ such that:

$$\begin{aligned} a_i b_i c_i &= a_i + b_i + c_i, \\ d_i^2 + 2a_{i+1} &= e_i, \\ &\text{for all } i \in [n].\text{''} \end{aligned} \tag{3}$$

Denote by $\mathrm{tr}_1, \mathrm{tr}_2, \mathrm{tr}_3, \mathrm{tr}_4, \mathrm{tr}_5 \in \mathbb{F}_{<n}[X]$ the polynomials that interpolate the values a_i,b_i,c_i,d_i,e_i over the domain G, respectively. That is, $\mathrm{tr}_1(g^i) = a_i$, $\mathrm{tr}_2(g^i) = b_i$, $\mathrm{tr}_3(g^i) = c_i$, $\mathrm{tr}_4(g^i) = d_i$, $\mathrm{tr}_5(g^i) = e_i$ for $i \in [n]$. From now on, we will refer to G as the *trace evaluation domain* and to $\mathrm{tr}_1, \mathrm{tr}_2, \mathrm{tr}_3, \mathrm{tr}_4, \mathrm{tr}_5$ as the *trace column polynomials*. Hence, the above constraint system (of size $2n$) can be "compressed" down into two polynomial constraints by means of the trace column polynomials. In particular, if for all $x \in G$ the following constraints are true:

$$\begin{aligned} \mathrm{tr}_1(x) \cdot \mathrm{tr}_2(x) \cdot \mathrm{tr}_3(x) &= \mathrm{tr}_1(x) + \mathrm{tr}_2(x) + \mathrm{tr}_3(x), \\ \mathrm{tr}_4(x)^2 + 2 \cdot \mathrm{tr}_1(gx) &= \mathrm{tr}_5(x), \end{aligned} \tag{4}$$

then the original constraint system (3) must hold. In general, we will be in the situation of generating a STARK for the knowledge of some polynomials $\mathrm{tr}_1, \dots, \mathrm{tr}_N : G \to \mathbb{F}$ that satisfy a system of polynomial constraints $\mathcal{C} = \{C_1, \dots, C_\ell\}$, where:

(a) $C_i \in \mathbb{F}[X_1, \dots, X_N, X'_1, \dots, X'_N]$ for all $i \in [\ell]$.

(b) For all $x \in G$ and all $i \in [\ell]$, we have:

$$C_i(\mathrm{tr}_1(x), \dots, \mathrm{tr}_N(x), \mathrm{tr}_1(gx), \dots, \mathrm{tr}_N(gx)) = 0, \tag{5}$$

when variables X_j are replaced by polynomials $tr_j(X)$ and variables X'_j are replaced by polynomials $tr_j(gX)$ in each C_i.

From Polynomial Constraints to Rational Functions

Given a constraint C_i, a rational function is associated to each one of them:

$$q_i(X) := \frac{C_i(\mathrm{tr}_1(X), \dots, \mathrm{tr}_N(X), \mathrm{tr}_1(gX), \dots, \mathrm{tr}_N(gX))}{Z_G(X)}, \tag{6}$$

where, recalling that $\deg(\mathrm{tr}_i) \leq n - 1$, each q_i is a polynomial of degree at most $\deg(C_i) \cdot (n-1) - n$ if and only if the polynomial Z_G divides C_i (i.e., C_i is satisfed over G).

Following with our particular example, we have:

$$\begin{aligned} q_1(X) &:= \frac{\mathrm{tr}_1(X) \cdot \mathrm{tr}_2(X) \cdot \mathrm{tr}_3(X) - \mathrm{tr}_1(X) - \mathrm{tr}_2(X) - \mathrm{tr}_3(X)}{Z_G(X)}, \\ q_2(X) &:= \frac{\mathrm{tr}_4(X)^2 + 2 \cdot \mathrm{tr}_1(gX) - \mathrm{tr}_5(X)}{Z_G(X)}, \end{aligned}$$

where $\deg(C_1) = 3$, $\deg(C_2) = 2$ and, $q_1(X)$, $q_2(X)$ are of degree at most $2n - 3$ and $n - 2$, respectively. In fact, the constraints in expression (4) get satisfied if and only if $\deg(q_1) \leq 2n - 3$ and $\deg(q_2) \leq n - 2$.

The Quotient Polynomial

In the next step, polynomials q_i are combined into a single polynomial Q known as the *quotient polynomial*. In the STARK proposed in [Sta21], to generate Q, the degree of each q_i is adjusted to a sufficiently large power of two. More precisely, we define $D_i := \deg(C_i)(n-1) - |G|$ and call D to the first power of two for which $D > D_i$ for all $i \in [\ell]$. Then, we compute the adjusted version of the rational functions:

$$\hat{q}_i(X) := (\mathfrak{a}_i X^{D - D_i - 1} + \mathfrak{b}_i) \cdot q_i(X) \tag{7}$$

where $\mathfrak{a}_i, \mathfrak{b}_i \in \mathbb{K}$ for all $i \in [\ell]$.

In our example, we have $D_1 = 2n-3, D_2 = n-2$, and therefore we take $D = 2n$ (recall n is a power of 2) and then:

$$\begin{aligned}\hat{q}_1(X) :=& \quad (\mathfrak{a}_1 X^2 + \mathfrak{b}_1) \cdot q_1(X),\\ \hat{q}_2(X) :=& \quad (\mathfrak{a}_2 X^{n+1} + \mathfrak{b}_2) \cdot q_2(X),\end{aligned}$$

hence, $\deg(q_1) \leq 2n-3$ and $\deg(q_2) \leq n-2$ if and only if $\deg(\hat{q}_1), \deg(\hat{q}_2) < 2n$.

Given polynomials $\hat{q}_i$, we can now compute the quotient polynomial as follows:

$$Q(X) := \sum_{i=1}^{\ell} \hat{q}_i(X) = \sum_{i=1}^{\ell} (\mathfrak{a}_i X^{D-D_i-1} \cdot \mathfrak{b}_i) \frac{C_i(\mathrm{tr}_1(X), \dots, \mathrm{tr}_N(X), \mathrm{tr}_1(gX), \dots, \mathrm{tr}_N(gX))}{Z_G(X)} \tag{8}$$

that satisfies $\deg(Q) < D$.

Then, Q is split into $S := D/n$ polynomials $Q_1, \dots, Q_S \in \mathbb{K}[X]$ of degree lower than n satisfying:

$$Q(X) = \sum_{i=1}^{S} X^{i-1} Q_i(X^S) \tag{9}$$

Notice that the polynomials Q_i are bounded by the same degree as the trace column polynomials, so we refer to them as the *trace quotient polynomials*.

Continuing with our example, the quotient polynomial is $Q(X) := \hat{q}_1(X) + \hat{q}_2(X)$ satisfying $\deg(Q) < 2n$. In this case, we represent the quotient polynomial $Q(X)$ as two polynomials $Q_1, Q_2 \in \mathbb{F}_{<n}[X]$ such that $Q(X) = Q_1(X^2) + X \cdot Q_2(X^2)$.

Trace Low Degree Extension

Since the quotient polynomial Q is defined through rational functions q_i that we are going to evaluate, in order for Cstr. (6) to be well-defined, we need to ensure that the denominators of these rational functions are never zero. Therefore, from now on, the polynomials will not be evaluated over the trace evaluation domain, but rather over a larger and disjoint domain, which we refer to as the *evaluation domain*.

More specifically, we introduce the evaluation domain to be:

1. Larger than G, so that the trace column polynomial have enough redundancy to ensure soundness of the FRI protocol.

2. Disjoint of G, so that Cstr. (6) is well-defined.

In order to achieve the previous two requirements, we will choose the evaluation domain to be the coset H, where remember that $|H| = 2^k \cdot n$, with $k \geq 1$. We will refer to 2^k as the *blowup factor*. Therefore, we need to evaluate all the trace column polynomials over the evaluation domain. We refer to the resulting set of polynomial evaluations as the *trace Low Degree Extension (LDE)*.

The trace LDE is computed in two steps:

1. We calculate the interpolation polynomial on the trace evaluation domain of each trace column polynomial using the Inverse Fast Fourier Transform (IFFT).

2. We evaluate the polynomials that result from the previous step on the evaluation domain using the Fast Fourier Transform (FFT).

Trace Consistency Check

At this point, the verifier has everything he needs to perform a local consistency check between the trace column polynomials and the trace quotient polynomials, refered to as the *trace consistency check*. Hence, after the prover commits to the trace quotient polynomials, the verifier uniformly samples a random z and request the prover to send the necessary polynomial evaluations on either z or gz.

More specifically, for a given $z \in \mathbb{K}\backslash(G \cup \bar{H})$ (here, $\bar{H} = \{x \in \mathbb{K} \mid x^S \in H\}$) uniformly sampled by a verifier, the prover sends back trace column polynomials evaluations $\mathrm{tr}_i(z), \mathrm{tr}_j(gz)$ and trace quotient polynomials evaluations $Q_k(z^S)$ for $i, j \in N$ and $k \in [S]$. Notice that the necessary evaluations of the trace column polynomials strictly depend on the particular polynomial expressions in (5). Denote by $\mathsf{Evals}(z)$ the set of polynomial evaluations over z and by $\mathsf{Evals}(gz)$ the set of polynomial evaluations over gz. Naturally, there could exist evaluations of a single polynomial in both sets.

With these evaluations, the verifier is able to check that:

$$\sum_{i=1}^{S} z^{i-1} Q_i(z^S) = \sum_{i=1}^{\ell} (\mathfrak{a}_i X^{D-D_i-1} \cdot \mathfrak{b}_i) \frac{C_i(\mathrm{tr}_1(z), \ldots, \mathrm{tr}_N(z), \mathrm{tr}_1(gz), \ldots, \mathrm{tr}_N(gz))}{Z_G(z)}. \quad (10)$$

In our particular example, the prover sends back $\mathrm{tr}_1(z)$,$\mathrm{tr}_1(gz)$,$\mathrm{tr}_2(z)$,$\mathrm{tr}_3(z)$,$\mathrm{tr}_4(z)$,$\mathrm{tr}_5(z)$ (so that $\mathsf{Evals}(z) = \{\mathrm{tr}_1(z), \mathrm{tr}_2(z), \mathrm{tr}_3(z), \mathrm{tr}_4(z), \mathrm{tr}_5(z)\}$ and $\mathsf{Evals}(gz) = \{\mathrm{tr}_1(gz)\}$) and $Q_1(z^2), Q_2(z^2)$, and the verifier checks that:

$$\begin{aligned} Q_1(z^2) + z \cdot Q_2(z^2) = \quad & (\mathfrak{a}_1 z^2 + \mathfrak{b}_1) \cdot \frac{\mathrm{tr}_1(z) \cdot \mathrm{tr}_2(z) \cdot \mathrm{tr}_3(z) - \mathrm{tr}_1(z) - \mathrm{tr}_2(z) - \mathrm{tr}_3(z)}{Z_G(z)} + \\ & + (\mathfrak{a}_2 z^{n+1} + \mathfrak{b}_2) \cdot \frac{\mathrm{tr}_4(z)^2 + 2 \cdot \mathrm{tr}_1(gz) - \mathrm{tr}_5(z)}{Z_G(z)}. \end{aligned} \quad (11)$$

The problem is that the verifier cannot be sure that the received values are actually evaluations of previously committed polynomials. In fact, it is easy to obtain values from $\mathbb{K}$ that satisfy (10) but are not from the expected polynomials. Therefore, after the prover sends the evaluations to the verifier, they engage in the part of protocol to ensure that the values are the actually evaluations of the corresponding polynomials.

The FRI Polynomial

In order to ensure the validity of the values sent from the prover to the verifier, the prover proceeds to create another set of constraints, then translate them to a problem of low degree testing, and finally, combines them through the use of random field elements. To this end, the prover computes the F polynomial:

$$\begin{aligned} \mathsf{F}(X) := \quad & \sum_{i \in I_1} \varepsilon_i^{(1)} \cdot \frac{\mathrm{tr}_i(X) - \mathrm{tr}_i(z)}{X - z} + \sum_{i \in I_2} \varepsilon_i^{(2)} \cdot \frac{\mathrm{tr}_i(gX) - \mathrm{tr}_i(gz)}{X - gz} \\ & + \sum_{i=1}^{S} \varepsilon_i^{(3)} \cdot \frac{Q_i(X) - Q_i(z^S)}{X - z^S}, \end{aligned} \quad (12)$$

where $I_1 = \{i \in [N] \colon \mathrm{tr}_i(z) \in \mathsf{Evals}(z)\}$, $I_2 = \{i \in [N] \colon \mathrm{tr}_i(gz) \in \mathsf{Evals}(gz)\}$ and $\varepsilon_i^{(1)}, \varepsilon_j^{(2)}, \varepsilon_k^{(3)} \in \mathbb{K}$ for all $i \in I_1, j \in I_2, k \in [S]$.

What we obtain is that, the F polynomial is of degree lower than $n-1$ if and only if: (1) both the trace columns polynomials tr_i and the trace quotient polynomials Q_i are of degree lower than n, and (2) all the values sent by the prover in the previous step are actually evaluations of the corresponding polynomials.

Finishing with our example, the prover computes the F polynomial as:

$$\begin{aligned}\mathsf{F}(X) := \quad & \varepsilon_1^{(1)} \frac{\mathrm{tr}_1(X) - \mathrm{tr}_1(z)}{X - z} + \varepsilon_2^{(1)} \cdot \frac{\mathrm{tr}_2(X) - \mathrm{tr}_2(z)}{X - z} + \varepsilon_3^{(1)} \cdot \frac{\mathrm{tr}_3(X) - \mathrm{tr}_3(z)}{X - z} \\ & + \varepsilon_4^{(1)} \cdot \frac{\mathrm{tr}_4(X) - \mathrm{tr}_4(z)}{X - z} + \varepsilon_5^{(1)} \cdot \frac{\mathrm{tr}_5(X) - \mathrm{tr}_5(z)}{X - z} + \varepsilon_1^{(2)} \cdot \frac{\mathrm{tr}_1(X) - \mathrm{tr}_1(gz)}{X - gz} \\ & + \varepsilon_1^{(3)} \cdot \frac{Q_1(X) - Q_1(z^2)}{X - z^2} + \varepsilon_2^{(3)} \cdot \frac{Q_2(X) - Q_2(z^2)}{X - z^2}.\end{aligned}$$

Proof Verification

In order to verify the STARK, the verifier performs the following checks:

(a) **Trace Consistency.** Checks that the trace quotient polynomials $Q_1, \dots, Q_S$ are consistent with the trace column polynomials $\mathrm{tr}_1, \dots, \mathrm{tr}_N$ by means of the evaluations of these polynomials at either z, gz or z^S. I.e., the verifier checks that Eq. (10) is satisfied.

(b) **Batched FRI Verification.** It runs the batched FRI verification procedure on the polynomial F.

If either (a) or (b) fail at any point, the verifier aborts and rejects. Otherwise, the verifier accepts.

The full description of the transcript can be found in one-shot in Appendix 1.4, and we include the FRI message exchange in Figure 6.

Soundness

We finally recall the soundness error of the vanilla STARK protocol as in Theorem 4 of [Sta21]. Recall that $\rho = |G|/|H|$ is the rate of the code.

Theorem 2 (Soundness)**.** *Fix integer $m \geq 3$. Suppose a prover that interacts with a verifier in the vanilla STARK protocol causes it to accept with probability two times greater than:*

$$\varepsilon_{\mathsf{STARK}} := \ell \left(\frac{1}{|\mathbb{K}|} + \frac{(D + |G| + S) \cdot \ell}{|\mathbb{K}| - S|H| - |G|} \right) + \varepsilon_{\mathsf{FRI}}, \tag{13}$$

then Eq. (5) is satisfied (i.e., the original statement is true), where $\ell = m/\rho$ and $\varepsilon_{\mathsf{FRI}}$ is as defined in Theorem 1.

We give some intuitions for the computation of this error bound. The first term in Eq. (13) corresponds to the probability of sets $\{\mathfrak{a}_1, \mathfrak{b}_1, \dots, \mathfrak{a}_T, \mathfrak{b}_T\}$ being "good" under a dishonest prover. This means that if all $\mathfrak{a}_i, \mathfrak{b}_i$ are uniformly and independently randomly sampled, then the probability the quotient polynomial Q is, in fact, a polynomial is at most $\ell/|\mathbb{K}|$. The second term corresponds to the probability of sampling a $z \in \mathbb{K} \backslash (G \cup \bar{H})$ for which the FRI protocol passes with probability greater than $\varepsilon_{\mathsf{FRI}}$.

1.5 Arguments

In this section, we introduce the "arguments" that we will use to extend the vanilla STARK. Here, by argument, we mean a relation between polynomials that cannot be directly expressed through an identity. We often refer to these arguments as *non-identity constraint.* The three arguments we will introduce are *multiset equality*, *connection* and *lookup.*

The protocols that instantiate these arguments are all based on the same idea of the computation of a grand-product polynomial over the two (or more) vectors involved in the argument. Specifically, a polynomial is cumulatively computed as the quotient of a function of the first vector and a function of the second vector. Then, a set of identities is proposed as an ensurance for a verifier of the protocol that not only the grand-product was correctly computed by a prover, but also that the specific intention of the protocol is satisfied. In order to ensure the soundness of the protocols, random values uniformly sampled by the verifier are used in such computation.

An alternative of the previous idea is based on the addition of logarithmic derivatives (see e.g. [Hab22a]) instead, but they come with tradeoffs that make them unwanted for the STARK context.

Recall that $G = \langle g \rangle$ is a cyclic subgroup of $\mathbb{F}^*$ of order n.

Multiset Equality

Given two vectors $f = (f_1, \dots, f_n)$ and $t = (t_1, \dots, t_n)$ in $\mathbb{F}^n$, a multiset equality argument, denoted $f \doteq t$, is used for checking that f and t are a permutation of each other. The protocol that instantiates the mulsiet equality arguments works by computing the following grand-product polynomial $Z \in \mathbb{K}_{<n}[X]$:

$$Z(g^i) = \begin{cases} 1, & \text{if } \ i = 1 \\ \displaystyle\prod_{j=1}^{i-1} \frac{(f_j + \gamma)}{(t_j + \gamma)}, & \text{if } \ i = 2, \dots, n \end{cases}$$

where $\gamma \in \mathbb{K}$ is the value sent from the verifier.

The definition of the previous polynomial is based on the following lemma.

Lemma 1 (Soundness of Multiset Equality)**.** *Fix two vectors $f = (f_1, \dots, f_n)$ and $t = (t_1, \dots, t_n)$ in $\mathbb{F}^n$. If the following holds with probability larger than $n/|\mathbb{K}|$ over a random $\gamma \in \mathbb{K}$:*

$$\prod_{i=1}^{n}(f_i + \gamma) = \prod_{i=1}^{n}(t_i + \gamma),$$

then $f \doteq t$.

Proof. Assume that $f \not\doteq t$. Then, there must be some $i^* \in [n]$ such that $t_{i^*} \neq f_i$ for all $i \in [n]$. Define degree n polynomials $F(X) := \prod_{i=1}^{n}(f_i + X)$ and $T(X) := \prod_{i=1}^{n}(t_i + X)$. By the assumption we have that $F \neq T$ and therefore by the Schwartz-Zippel lemma $F(\gamma) \neq T(\gamma)$ except with probability $n/|\mathbb{K}|$. ■

As a consequence of Lemma 1, the identities that must be checked by the verifier for $x \in G$ are the following:

$$L_1(x) \cdot (Z(x) - 1) = 0, \tag{14}$$

$$Z(x \cdot g) \cdot (t(x) + \gamma) = Z(x) \cdot (f(x) + \gamma), \tag{15}$$

where $f, t \in \mathbb{F}_{<n}[X]$ are the polynomials resulting from the interpolation of $\{f_i\}_{i \in [n]}$ and $\{t_i\}_{i \in [n]}$ over G,respectively.

Connection

The protocol for a connection argument and the definitions and results we provide next are adapted from [GWC19].

Given some vectors $f_1, \dots, f_k \in \mathbb{F}^n$ and a partition $\mathcal{T} = \{T_1, \dots, T_k\}$ of the set $[k] \times [n]$, a connection argument, denoted $(f_1, \dots, f_k) \propto \{T_1, \dots, T_k\}$, is used to check that the partition $\mathcal{T}$ divides the field elements $\{f_{i,j}\}_{i \in [k], j \in [n]}$ into sets with the same value. More specifically, we have $f_{i_1,j_1} = f_{i_2,j_2}$ if and only if $(i_1, j_1), (i_2, j_2)$ belong to the same block of $\mathcal{T}$.

In order to express the partition $\mathcal{T}$ within a grand-product polynomial, we define a permutation $\sigma\colon [kn] \to [kn]$ as follows: σ is such that for each block T_i of $\mathcal{T}$, $\sigma(\mathcal{T})$ contains a cycle going over all elements of T_i. Then, the protocol that instantiates the connection arguments works by computing the following grand-product polynomial $Z \in \mathbb{K}_{<n}[X]$:

$$Z(g^i) = \begin{cases} 1, & \text{if } \ i = 1 \\ \displaystyle\prod_{\ell=1}^{k}\prod_{j=1}^{i-1} \frac{(f_{\ell,j} + \gamma \cdot (\ell-1)\cdot n + j) + \delta)}{(f_{\ell,j} + \gamma \cdot \sigma((\ell-1)\cdot n + j) + \delta)}, & \text{if } \ i = 2, \ldots, n \end{cases}$$

where $\gamma, \delta \in \mathbb{K}$ are the values sent from the verifier.

The definition of the previous polynomial is based on the following lemma, a proof of which can be found[1] in Claim A.1. of [GWC19] and is similar to the one of Lemma 1.

Lemma 2 (Soundness of Connection)**.** *Fix $f_1, \ldots, f_k \in \mathbb{F}^n$ and a partition $\mathcal{T} = \{T_1, \ldots, T_k\}$ of $[k] \times [n]$. If the following holds with probability larger than $kn/|\mathbb{K}|$ over randoms $\gamma, \delta \in \mathbb{K}$:*

$$\prod_{\ell=1}^{k}\prod_{j=1}^{n}(f_{\ell,j} + \gamma \cdot ((\ell-1)\cdot n + j) + \delta) = \prod_{\ell=1}^{k}\prod_{j=1}^{n}(f_{\ell,j} + \gamma \cdot \sigma((\ell-1)\cdot n + j) + \delta),$$

then, $(f_1, \ldots, f_k) \propto \{T_1, \ldots, T_k\}$.

As a consequence of Lemma 2, the identities that must be checked by the verifier for $x \in G$ are the following:

$$\begin{gathered} L_1(x)\cdot(Z(x)-1) = 0, \\ Z(x\cdot g) = Z(x)\cdot\frac{(f_1(x) + \gamma\cdot S_{\mathsf{ID}_1}(x) + \delta)}{(f_1(x) + \gamma\cdot S_{\sigma_1}(x) + \delta)}\cdot\cdots\cdot\frac{(f_k(x) + \gamma\cdot S_{\mathsf{ID}_k}(x) + \delta)}{(f_k(x) + \gamma\cdot S_{\sigma_k}(x) + \delta)}, \end{gathered} \tag{16}$$

where $S_{\mathsf{ID}_\ell}(g^j) = (\ell-1)\cdot n+j$ is the polynomial mapping G-elements to indexes in $[kn]$ and $S_{\sigma_\ell}(g^j) = \sigma((\ell-1)\cdot n+j)$ is the polynomial defined by σ. Since the permutation σ perfectly relates with the partition $\mathcal{T}$ it refers to, from now on we denote a connection argument between polynomials $f_1, \ldots, f_k \in \mathbb{F}[X]$ and a partition $\mathcal{T}$ as $(f_1, \ldots, f_k) \propto (S_{\sigma_1}, \ldots, S_{\sigma_k})$. As we will see in later sections, this overloading notation will become very natural.

For more details see [GWC19].

Lookup

The protocol for a lookup argument and the definitions and results we provide next are adapted from [GW20], with the "alternating method" provided in [PFM+22].

Given two vectors $f = (f_1, \ldots, f_n)$ and $t = (t_1, \ldots, t_n)$ in $\mathbb{F}^n$, a lookup argument, denoted $f \in t$, is used for checking that the set A formed with the values $\{f_i\}_{i\in[n]}$ is contained in the set B formed with the values $\{t\}_{i\in[n]}$. Notice that $|A|, |B| \leq n$.

In the protocol, the prover has to construct an auxiliary vector $s = (s_1, \ldots, s_{2n})$ containing every element of f and t where the order of appearance is the same as in t. The main idea behind the protocol is that if $f \in t$, then f contributes to s with repeated elements. To check this fact, a vector Δs is defined as follows:

$$\Delta s = (s_1 + \gamma\, s_2, s_2 + \gamma\, s_3, \cdots, s_{2n} + \gamma\, s_1).$$

Then, the protocol essentially checks that Δs is consistent with the elements of f, t and s. To do so, the vector s is split in two vectors $h_1, h_2 \in \mathbb{F}^n$. In the protocol described

[1] The claim in [GWC19] is for a slightly more general protocol.

in [GW20], h_1 and h_2 contain the lower and upper halves of s, while in our protocol in [PFM+22], we use h_1 to store elements with odd indexes and h_2 for even indexes, that is:

$$h_1 = (s_1, s_3, s_5, ..., s_{2n-1}) \quad \text{and} \quad h_2 = (s_2, s_4, s_6, ..., s_{2n}). \tag{17}$$

With this setting in mind, the grand-product polynomial is defined as:

$$Z(g^i) = \begin{cases} 1, & \text{if } i = 1 \\ (1+\gamma)^{i-1} \prod_{j=1}^{i-1} \frac{(\delta + f_j)(\delta(1+\gamma) + t_j + \gamma t_{j+1})}{(\delta(1+\gamma) + s_{2j-1} + \gamma s_{2j})(\delta(1+\gamma) + s_{2j} + \gamma s_{2j+1})}, & \text{if } i = 2, \dots, n \end{cases}$$

where $\gamma, \delta \in \mathbb{K}$ are the values sent from the verifier.

The definition of the previous polynomial is based on the following lemma, which is a slight modification of Claim 3.1. of [GW20].

Lemma 3 (Soundness of Lookup)**.** *Fix three vectors $f = (f_1, \dots, f_n), t = (t_1, \dots, t_n)$ and $s = (s_1, \dots, s_{2n})$ with elements in $\mathbb{F}$. If the following holds with probability larger than $(4n-2)/|\mathbb{K}|$ over randoms $\gamma, \delta \in \mathbb{K}$:*

$$(1+\gamma)^n \prod_{i=1}^{n} (\delta + f_i) \prod_{i=1}^{n-1} (\delta(1+\gamma) + t_i + \gamma t_{i+1}) = \prod_{i=1}^{2n-1} (\delta(1+\gamma) + s_i + \gamma s_{i+1}),$$

then $f \in t$ and s is the sorted by t concatenation of f and t.

As a consequence of Lemma 3, the identities that must be checked by the verifier for $x \in G$ are the following:

$$\begin{gathered} L_1(x)\,(Z(x) - 1) = 0, \\ Z(x \cdot g) = Z(x) \frac{(1+\gamma)(\delta + f(x))(\delta(1+\gamma) + t(x) + \gamma t(gx))}{(\delta(1+\gamma) + h_1(x) + \gamma h_2(x))(\delta(1+\gamma) + h_2(x) + \gamma h_1(x \cdot g))}. \end{gathered} \tag{18}$$

where $f, t \in \mathbb{F}_{<n}[X]$ are the polynomials resulting from the interpolation of $\{f_i\}_{i \in [n]}$ and $\{t_i\}_{i \in [n]}$ over G,respectively; and $h_1, h_2 \in \mathbb{F}_{<n}[X]$ are the polynomials resulting from the interpolation of the values defined in Eq. (17) over G.

For more details see [GW20] and [PFM+22].

2 Our Techniques

In this section we explain the main differences between the techniques used during the rounds of the vanilla STARK (Section 1.4) and our techniques.

2.1 Commiting to Multiple Polynomial at Once

For this section, explicitly set $H = \{h_1, h_2, h_3, \dots, h_m\}$. Also, denote by $f_0, f_1, \dots, f_N \in \mathbb{F}_{<n}[X]$ the set of polynomials that we want to construct the Merkle Tree on. That is, the input to the Merkle Tree constructor is the set of evaluations:

$$\boxed{\begin{array}{ccccc} f_0(h_1), & f_1(h_1), & f_2(h_1), & \dots, & f_N(h_1) \\ f_0(h_2), & f_1(h_2), & f_2(h_2), & \dots, & f_N(h_2) \\ \vdots & \vdots & \vdots & \dots & \vdots \\ f_0(h_m), & f_1(h_m), & f_2(h_m), & \dots, & f_N(h_m) \end{array}}$$

A leaf element of the Merkle Tree will consist on the evaluations of all the polynomials at a single point. This will be convinient for a latter step of the STARK generation. For instance, in the batched FRI protocol we group evaluations of all the original polynomials at a common point for succinctly answering a batched consistency check. Specifically, assuming that $f_0, f_1, \dots, f_N$ are the original polynomials, the leaf elements of the Merkle Tree, indexed by the corresponding value, will consist on:

$$\boxed{\begin{array}{ccc} \text{leaf } h_1 & \Longrightarrow & \mathcal{H}(f_0(h_1), f_1(h_1), f_2(h_1), \dots, f_N(h_1)) \\ \text{leaf } h_2 & \Longrightarrow & \mathcal{H}(f_0(h_2), f_1(h_2), f_2(h_2), \dots, f_N(h_2)) \\ \vdots & & \vdots \\ \text{leaf } h_m & \Longrightarrow & \mathcal{H}(f_0(h_m), f_1(h_m), f_2(h_m), \dots, f_N(h_m)) \end{array}}$$

where $\mathcal{H}$ is any collision resistant hash function. Notice how if h_i is the requested point of check by the verifier, then the prover can prove the consistency of all the evaluations $f_0(h_i), \dots, f_N(h_i)$ with the corresponding Merkle root in a single Merkle path.

A proof that the resulting non-interactive protocol is knowledge sound after applying the Fiat-Shamir using this strategy can be found, for example, in Theorem 4 of [AFK21].

To be more specific, the hash function H is set to be the Poseidon [GKR+21] hash function. Poseidon was chosen because it was created to minimize prover and verifier complexities when zero-knowledge proofs are generated and validated. Notably, the best hashing performance is obtained when the state size is limited to 12 field elements, 4 of which are occupied by the capacity of the hash function. This implies that, in order to get the best Poseidon performance, we have to restrict the input size to be of 8 field elements.

Leaf hashes are performed "linearly". By linearly we mean that, if the input to the hash function is $t_1(sh^i)$, $t_2(sh^i)$, $\dots$, $t_N(sh^i)$, then we process it as follows:

1. The input is split in chunks of 8 elements, filling with repetitions of the 0 element if N is not a multiple of 8.

2. The first chunk is hashed using Poseidon with capacity $(0, 0, 0, 0)$.

3. The following chunk is hashed using Poseidon with the capacity being the output of the previous hash.

4. Go to Step 3 until there are no more chunks.

An example of this process can be seen in the Figure 1.

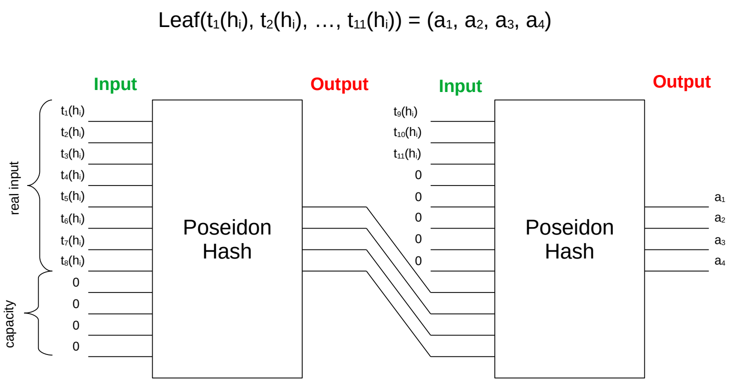

Figure 1: Leaf hash computation on input $(t_1(sh^i), \dots, t_{11}(sh^i))$ in a linear manner.

Once all the hashed leafs are obtained, one starts to construct the Merkle tree by continually hashing two child nodes using Poseidon with capacity $(0, 0, 0, 0)$ and defining the parent node as the output. This that this is well defined because Poseidon's output consist on 4 field elements, while its input size consists on 8. See Figure 2 for an example with 4 leafs.

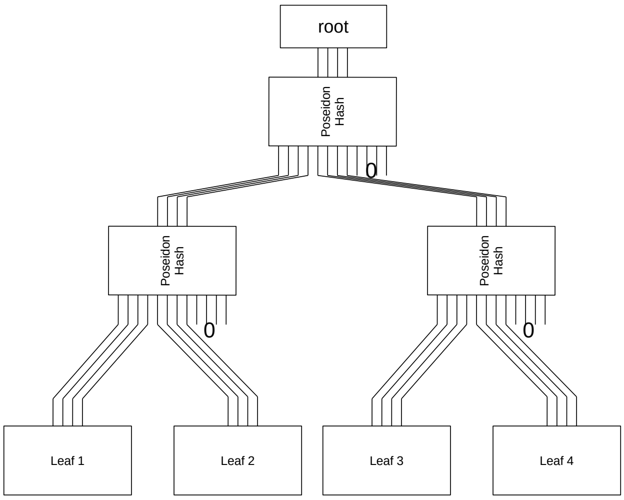

Figure 2: Merkle's tree hash computation assuming 4 leafs.

In fact, this procedure has been extended in order to being able to compute hashes faster using several GPUs. First of all, we are splitting all our polynomials in 4 chunks of size:

$$\texttt{batchSize} = \left\lfloor \max\left(8, \frac{N+3}{4}\right) \right\rfloor.$$

Of course, it may be possible that not all of the chunks have exactly this amount of elements. In this case, we prioritize to fill the first 3 ones up to this size, letting the last one become smaller, but not so smaller. The idea is that, with the formula above, the chunk size is increased by 1 once N increase by 4 (when $N > 32$, which is the first time when all chunks has exactly `batchSize` number of elements). Hence, for N big enough, the last chunk never will become smaller than $\texttt{batchSize} - 3$, becoming a almost uniform distribution of the polynomials among the 4 chunks. Table 3 shows several examples on how this chunk division sizes looks like.

After the polynomial splitting in 4 chunks (say T_1, T_2, T_3 and T_4), we perform the previously defined linear hash of all of them in a parallel way, ending up with a set of a maximum amount of 16 field elements corresponding to the 4 outputs of 4 field elements each. To finish this updated version of the linear hash, we perform the linear hash of this 16 elements as done previously, which outputs a final total amount of 4 field elements. More precisely, if

$$LH(T_i) = (H_{i,1}, H_{i,2}, H_{i,3}, H_{i,4}) \quad i \in \{1, 2, 3, 4\},$$

then the final output will be

$$LH(H_{1,1}, H_{1,2}, H_{1,3}, H_{1,4}, H_{2,1}, H_{2,2}, H_{2,3}, H_{2,4}, H_{3,1}, H_{3,2}, H_{3,3}, H_{3,4}, H_{4,1}, H_{4,2}, H_{4,3}, H_{4,4}),$$

where LH denotes the single-GPU version of the linear hash.

2.2 Transcript Generation and Computing Verifier Challenges

We will describe our protocol version as a non-interactive protocol using the Fiat-Shamir heuristic [FS87]. Therefore, we need to specify how we are generating the random challenges

N	`batchSize`	**Chunk 1**	**Chunk 2**	**Chunk 3**	**Chunk 4**
1	8	1	0	0	0
...	...	...	...	...	...
8	8	8	0	0	0
9	8	8	1	0	0
10	8	8	2	0	0
...	...	...	...	...	...
17	8	8	8	1	0
...	...	...	...	...	...
25	8	8	8	8	1
...	...	...	...	...	...
32	8	8	8	8	8
33	9	9	9	9	6
34	9	9	9	9	7
35	9	9	9	9	8
36	9	9	9	9	9
37	10	10	10	10	7
38	10	10	10	10	8
...	...	...	...	...	...
51	13	13	13	13	12
...	...	...	...	...	...

Figure 3: Chunk's size distribution for several values of N.

from $\mathbb{K}$ (or equivalently, 3 elements of $\mathbb{F}$). Through all this section we will use an instance of a Poseidon hash function having state size of 12 field elements (8 for inputs and 4 for capacity) and output size of 12 field elements.

The strategy for generating the transcript is similar to the linear hash strategy described before. Suppose we want to add $c_1, \dots, c_r$ elements to the transcript. We proceed as follows:

1. The input is split in chinks of 8 field elements, filling with repetitions of the 0 element if r is not a multiple of 8.
2. The first chunk is hashed using Poseidon with capacity $(0, 0, 0, 0)$.
3. The following chunk is hashed using Poseidon with the capacity being the 4 last elements of the output of the previous hash.
4. Go to Step 3 until there are no more chunks.

Observe that the 8 remaining elements of each hash output are not being used until we stop the loop between the steps 3 and 4. We depict the previous strategy in Figure 4.

When we stop adding elements to the transcript, we end up with an output consisting in 8 field elements, say $(t_1, \dots, t_8)$. For a given transcript state we can extract at most 3 challenges of $\mathbb{K}$. The first two elements are trivially obtained

$$t_1 + t_2\varphi + t_3\varphi^2, \quad t_4 + t_5\varphi + t_6\varphi^2$$

for φ being a root of the irreducible polynomial used to construct $\mathbb{K}$ from $\mathbb{F}$. Since we do not have enough elements to construct a third extension field element, we proceed as follows. We construct an field element t_9 hashing 8 zeros with the capacity being the last 4

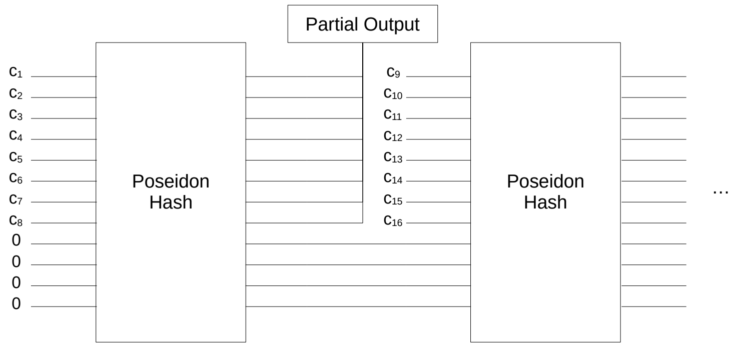

Figure 4: First two steps of the transcript generation.

output elements of the last hash performed at the time of generating the transcript (that is, the elements that will become the capacity of the next hash when adding a new element into the transcript). Hence, we get a third element in $\mathbb{K}$

$$t_7 + t_8\varphi + t_9\varphi^2.$$

However, observe that we can not extract more than that.

We will denote by $\mathsf{transcript}$ the transcript instance and we will define for it the following operations:

- **Add:** Having elements $c_1, \dots, c_r \in \mathbb{F}$, we denote by

$$\mathsf{add}_{\mathsf{transcript}}(c_1, \dots, c_r)$$

  the operation of adding $c_1, \dots, c_r$ to the transcript using the previous procedure.

- **Extract:** Having a transcript state T, we denote by

$$\mathsf{extract}_i(\mathsf{transcript}) \in \mathbb{K} \quad i \in \{1, 2, 3\}$$

  the result of extracting a single extension field $\mathbb{K}$ element from it using the previously described procedure. Using the notation above:

$$\begin{aligned}
\mathsf{extract}_1(\mathsf{transcript}) &= t_1 + t_2\varphi + t_3\varphi^2, \\
\mathsf{extract}_2(\mathsf{transcript}) &= t_4 + t_5\varphi + t_6\varphi^2, \\
\mathsf{extract}_3(\mathsf{transcript}) &= t_7 + t_8\varphi + t_9\varphi^2.
\end{aligned}$$

A proof that the resulting non-interactive protocol is knowledge sound after applying the Fiat-Shamir using this strategy can be found, for example, in Theorem 4 of [AFK21].

2.3 Preprocessed Polynomials and Public Values

Among the set of polynomials that are part of the polynomial constraint system representing the problem's statement, we differentiate between two types: *committed polynomials* and *preprocessed polynomials*.

Committed polynomals are those polynomials for which the verifier only has oracle access and are therefore committed (via Merkle trees) by the prover before the verifier starts querying them. In other words, these polynomals can only be known, in principle, in its entire form by the prover of the protocol. Constrastly, the verifier is limited to know a "small fraction" of these polynomials' coefficients. In practise, this fraction is randomly chosen by the verifier and is proportional to the number of oracle queries that the verifier makes to the particular polynoial. For the shake of the protocols to be scalable, the number of queries made to committed polynomials should be at most logarithmic in their degree. An example of committed polynomials are trace columns polynomials tr_i.

On the other hand, preprocessed polynomials are totally known by the verifier even before the execution of the corresponding protocol. More precisely, once a polynomial constraint system $\mathcal{C}$ is fixed, the verifier has complete access (either in coefficient form or in evaluation form) to the set of preprocessed polynomials. As with committed polynomials, the verifier ends up needing only a small subset of evaluations of such polynomials. An example of preprocessed polynomials are Lagrange polynomials L_i.

Example 2. As an example, the polynomial constraint such that for all $x \in G$ satisfies:

$$L_1(x)(\mathrm{tr}_1(x) - 7) = 0, \tag{19}$$

is composed of the committed polynomial tr_1 and the preprocessed (Lagrange) polynomial L_1, and is satisfied if and only if $\mathrm{tr}_1(g) = 7$.

Finally, *public values* are defined as the set of committed polynomials evaluations that are attested by some constraint. Clearly, public values are known to both the prover and the verifier and a particular polynomial can have many public values associated with it. In Example 2, the value $\mathrm{tr}_1(g)$ is a public value since Eq. (19) constraints it to be equal to 7.

2.4 Adding Selected Vector Arguments

In this section, we describe how to augment the type of available constraints with the arguments presented in Section 1.5. Recall that we will add three new types of arguments:

- **Lookup ($\in$).** The set constructed from the evaluations of a polynomial f over a multiplicative subgroup G is contained in an equally defined set of another polynomial t. We denote it as $f \in t$.
- **Multiset Equality ($\doteq$).** The vector constructed from the evaluations of a polynomial f over a multiplicative subgroup G is a permutation of an equally defined vector of another polynomial t. We denote it as $f \doteq t$.
- **Connection ($\propto$).** The vector constructed from the evaluations of a set of polynomials $f_1, \dots, f_\ell$ over a multiplicative subgroup G does not vary after applying a known permutation σ to them. We denote it as $(f^{(1)}, \dots, f^{(\ell)} \propto (S^{(\sigma_1)}, \dots, S^{(\sigma_\ell)})$.

In order to include non-identity constraints to the protocol, we will represent them through their succinct set of identity constraints. We denote by $M^{\in}$ the number of lookup instantiations, $M^{\doteq}$ the number of multiset equality instantiations and $M^{\propto}$ the number of connection instantiations.

As detailed in Section 1.5, for the lookup argument, we need to compute and commit the associated polynomials $h_{1,j}$ and $h_{2,j}$ before being able to compute the corresponding grand-product polynomial for each lookup constraint $j \in [M]$. This sums up to $2M$ polynomials. After this, for each non-identity constraint, we compute the associate grand-product polynomial Z and commit to it. This definition of this polynomial is different depending on which argument we are executing as shown in Section 1.5. This sums up to M lookup polynomials, M' multiset equality polynomials and M'' connection polynomials. Overall, adding non-identity constraints adds up to $3M + M' + M''$ polynomials (that will need to be committed) and $2(M + M' + M'')$ polynomial constraints to the STARK.

Following, we explain how we generalize both lookups and multiset equalities to not only involving multiple polynomials, but also to a subset of the resulting vector. Therefore, somewhat artificially, enlarge the expressiveness of our arguments and let us handle more generic non-identity constraints. Let's explain first how we reduce vector lookups or multiset equalities to simple (i.e., one polynomial) lookups or multiset equalities.

Definition 5 (Vector Arguments)**.** Given polynomials $f_i, t_i \in \mathbb{K}_{<n}[X]$ for $i \in [N]$, a *vector lookup* is the argument in which for all $x \in G$ there exists some $y \in G$ such that:

$$(f_1(x), \ldots, f_N(x)) = (t_1(y), \ldots, t_N(y)). \tag{20}$$

A *vector mutliset equality* is defined analogously, but multiplicies of elements should be the same. That is, if for instance there exists $x_1, x_2 \in G$ such that $(f_1(x_1), \ldots, f_N(x_1)) = (f_1(x_2), \ldots, f_N(x_2))$, then there should exist $y_1, y_2 \in G$ such that $(t_1(y_1), \ldots, t_N(y_1)) = (t_1(y_2), \ldots, t_N(y_2))$ and for which Eq. (20) holds.

To reduce the previous vector argments to simple ones, we make use of a uniformly sampled element $\alpha \in \mathbb{K}$. Namely, instead of trying to generate a grand-product polynomial for Eq. (20), we define the following polynomials:

$$F'(X) := \sum_{i=1}^{N} \alpha^{i-1} f_i(X), \quad T'(X) := \sum_{i=1}^{N} \alpha^{i-1} t_i(X), \tag{21}$$

and compute the grand-product polynomial for the relation $F' \in T'$ or $F' \doteq T'$. Notice that both F' and T' are in general polynomials with coefficients over the field extension $\mathbb{K}$ even if every coefficient of f_i, t_i are precisely over the base field $\mathbb{F}$. This reduction leads to the following result, whose proof is a direct consequence of the Schwartz–Zippel lemma.

Lemma 4. *Given polynomials $f_i, t_i \in \mathbb{K}_{<n}[X]$ for $i \in [N]$ and $F', T' \in \mathbb{K}_{<n}[X]$ as defined by Eq. (21), if $F' \in T'$ (resp. $F' \doteq T'$), then Eq. (20) (resp. the equation for vector multiset equality) holds excepts with probability $n \cdot (N-1)/|\mathbb{K}|$ over the random election of α.*

Now, let's go one step further by the introduction of *selectors*. Informally speaking, a selected lookup (mutliset equality) is a lookup (multiset equality) not between the specified two polynomials f, t, but between the polynomials generated by the multiplication of f and t with (generally speaking) independently generated selectors. We generalize to the vector setting.

Definition 6 (Selected Vector Arguments)**.** We are given polynomials $f_i, t_i \in \mathbb{K}[X]$ for $i \in [N]$. Furthermore, we are also given two polynomials $f^{\mathrm{sel}}, t^{\mathrm{sel}} \in \mathbb{F}[X]$ whose range over the domain G is $\{0, 1\}$. That is, f^{sel} and t^{sel} are *selectors*. A *selected vector lookup* is the argument in which for all $x \in G$ there exists some $y \in G$ such that:

$$f^{\mathrm{sel}}(x) \cdot (f_1(x), \ldots, f_N(x)) = t^{\mathrm{sel}}(y) \cdot (t_1(y), \ldots, t_N(y)). \tag{22}$$

A *selected vector multiset equality* is defined analogously, but multiplicies of elements should be the same as explained in Def. 5.

Remark 1. Note that if $f^{\mathrm{sel}} = t^{\mathrm{sel}} = 1$, then Eq. (22) is reduced to (20); if $f^{\mathrm{sel}} = t^{\mathrm{sel}} = 0$ then the argument is trivial; and if either f^{sel} or t^{sel} are equal to the constant 1, then we remove the need for f^{sel} or t^{sel}, respectively.

To reduce the previous selected vector lookup to simple ones, we proceed in two steps. First, we use the reduction in Eq. (21) to reduce the inner vector argument to a simple one. This process outputs polynomials $F', T' \in \mathbb{K}[X]$. Second, we make use of another uniformly sampled $\beta \in \mathbb{K}$ as follows. Namely, instead of trying to generate a grand-product polynomial for Eq. (22), we define the following polynomials:

$$\begin{aligned} T(X) &:= t^{\text{sel}}(X)[T'(X) - \beta] + \beta, \\ F(X) &:= f^{\text{sel}}(X)[F'(X) - T(X)] + T(X), \end{aligned} \tag{23}$$

and compute the grand-product polynomial for the relation $F \in T$. Importantly, the presentation "re-ordering" in Eq. (23) is relevant: if β had been introduced in the definition of F instead, then there would be situations in which we would end up having β as a lookup value and therefore the lookup argument not being satisfied even if the selectors are correct.

Example 3. Choose $N = 1$, $n = 2^3$. We compute the following values:

x	$f_1(x)$	$F'(x)$	$f^{\text{sel}}(x)$	$F(x)$	$t_1(x)$	$T'(x)$	$t^{\text{sel}}(x)$	$T(x)$
g	3	3	0	1	1	1	1	1
g^2	7	7	1	7	1	1	0	β
g^3	4	4	0	7	7	7	1	7
g^4	1	1	1	1	6	6	0	β
g^5	5	5	1	5	5	5	1	5
g^6	1	1	0	5	5	5	1	5
g^7	2	2	1	2	5	5	0	β
g^8	5	5	1	5	7	2	1	2

Notice how $F \in T$. However, if we would have instead defined F, T as $F(X) = f^{\text{sel}}(X)[F'(X) - \beta] + \beta$ and $T(X) = t^{\text{sel}}(X)[T'(X) - F(X)] + F(X)$ then we would end up having β as a lookup value, which implies that $F \notin T$ even tho f_1, t_1 and $f^{\text{sel}}, t^{\text{sel}}$ are correct.

To reduce selected vector mutliset equalities to simple ones, we follow a similar process than with selected vector lookups. We also first use the reduction in Eq. (21) to reduce the inner vector argument to simple one, but then we define:

$$\begin{aligned} F(X) &:= f^{\text{sel}}(X)[F'(X) - \beta] + \beta, \\ T(X) &:= t^{\text{sel}}(X)[T'(X) - \beta] + \beta, \end{aligned} \tag{24}$$

Here, we have been able to firstly define F since we are dealing with mutliset equalities instead of inclusions.

Simlarly to the vector-to-simple reduction, we obtain the following result by observing that polynomials F, T (either from Eq. (23) or Eq. (24)) are of total degree $N - 1$ over variables α, β.

Lemma 5. *Given polynomials $f_i, t_i \in \mathbb{K}_{<n}[X]$ for $i \in [N]$ and $F, T \in \mathbb{K}_{<n}[X]$ as defined by Eq. (23) (resp. Eq. (24)), if $F \in T$ (resp. $F \doteq T$), then Eq. (22) (resp. the equation for selected vector multiset equalities) holds excepts with probability $n \cdot (N-1)/|\mathbb{K}|$ over the random and independent election of α and β.*

Lemmas 4 and 5 imply the following bounds.

Theorem 3 (Soundness Bounds)**.** *Given polynomials $f_i, t_j \in \mathbb{K}_{<n}[X]$ for $i \in [N]$, we obtain:*

1. ***Plookup**. Let $F, T \in \mathbb{K}_{<n}[X]$ as defined by Eq. (23). If a prover that interacts with a verifier causes it to accept with probability greater than:*

$$\varepsilon_{\mathsf{Plo}} := n\frac{N-1}{|\mathbb{K}|} + \frac{4n-2}{|\mathbb{K}|},$$

then Eq. (22) holds.

2. ***Multiset Equality***. *Let $F, T \in \mathbb{K}_{<n}[X]$ as defined by Eq. (24). If a prover that interacts with a verifier causes it to accept with probability greater than:*

$$\varepsilon_{\mathsf{MulEq}} := n\frac{N-1}{|\mathbb{K}|} + \frac{n}{|\mathbb{K}|},$$

then Eq. (22) holds.

3. ***Connection***. *Let $F, T \in \mathbb{K}_{<n}[X]$ as defined by Eq. (24). If a prover that interacts with a verifier causes it to accept with probability greater than:*

$$\varepsilon_{\mathsf{MulEq}} := \ell\frac{n}{|\mathbb{K}|},$$

then Eq. (22) holds.

Example 4. Say that for all $x \in G$ the prover wants to prove that he knows some polynomials $\mathrm{tr}_1, \mathrm{tr}_2, \mathrm{tr}_3, \mathrm{tr}_4, \mathrm{tr}_5 \in \mathbb{F}_{<n}[X]$ such that:

$$\begin{aligned} \mathrm{tr}_1 &\in \mathrm{tr}_3, \\ \mathrm{tr}_3 &\doteq \mathrm{tr}_4, \\ (\mathrm{tr}_2, \mathrm{tr}_1, \mathrm{tr}_5) &\propto (S_{\sigma_1}, S_{\sigma_2}, S_{\sigma_3}), \end{aligned} \tag{25}$$

where we have used the notation $\doteq$ to denote that c and d are a permutation of each other, without specifying a particular permutation.

Following the previous section and Section 2.6, the polynomial constraint system (25) gets transformed to the following one, so that for all $x \in G$:

$$\begin{aligned} L_1(x)\,(Z_1(x) - 1) &= 0, \\ Z_1(gx) = Z_1(x)&\frac{(1+\beta)(\gamma + \mathrm{tr}_1(x))(\gamma(1+\beta) + \mathrm{tr}_3(x) + \beta \mathrm{tr}_3(gx))}{(\gamma(1+\beta) + h_{1,1}(x) + \beta h_{1,2}(x))(\gamma(1+\beta) + h_{1,2}(x) + \beta h_{1,1}(gx))}, \\ L_1(x)\,(Z_2(x) - 1) &= 0, \\ Z_2(gx) &= Z_2(x)\frac{(\gamma + \mathrm{tr}_3(x))}{(\gamma + \mathrm{tr}_4(x))}, \\ L_1(x)\,(Z_3(x) - 1) &= 0, \\ \mathrm{im}_1(x) &= (\mathrm{tr}_1(x) + \beta k_1 x + \gamma)(\mathrm{tr}_5(x) + \beta k_2 x + \gamma), \\ \mathrm{im}_2(x) &= (\mathrm{tr}_1(x) + S_{\sigma_2}(x) + \gamma)(\mathrm{tr}_5(x) + S_{\sigma_3}(x) + \gamma), \\ Z_3(gx) &= Z_3(x)\frac{(\mathrm{tr}_2(x) + \beta x + \gamma)\mathrm{im}_1(x)}{(\mathrm{tr}_2(x) + S_{\sigma_1}(x) + \gamma)\mathrm{im}_2(x)}, \end{aligned}$$

where we notice that the only type of argument that sometimes need to be adjusted are the connection arguments.

We end this section by explaining the protocol corresponding to a multiple execution of the previously protocols combined all together. Denote by M to the number of lookups, by M' the number of multiset equalities and by M'' the number of connections.

Protocol 1. The protocol starts with a set of polynomials $f_{i,j}, t_{i,j} \in \mathbb{F}_{<n}[X]$ for $i \in [N]$ and $j \in [M + M' + M'']$ known to the prover. Here, for each $j \in [M]$, $\{f_{i,j}, t_{i,j}\}_i$ correspond to the polynomials of each M plookup invocations; for each $j \in [M+1, M+M']$, $\{f_{i,j}, t_{i,j}\}_i$ correspond to the polynomials of each M' multiset equality invocations and for each $j \in [M + M' + 1, M + M' + M'']$, $\{f_{i,j}\}_i$ correspond to the polynomials of each M'' connection invocations and $\{t_{i,j}\}_i$ correspond to the polynomials $\{S_{i,\sigma_j}\}_i$ derived from each permutation σ_j. For each $j \in [M + M']$, the prover possibly also knows selectors $f_j^{\mathrm{sel}}, t_j^{\mathrm{sel}}$.

1. **Execution Trace Oracles:** The prover sends oracle functions $[f_{i,j}], [t_{i,j}], [f_j^{\text{sel}}], [t_j^{\text{sel}}]$ to the verifier, who responds with uniformly sampled values $\alpha, \beta \in \mathbb{K}$.

2. **Plookup Oracles:** The prover computes the Plookup polynomials $h_{1,j}, h_{2,j}$ for each plookup invocation $j \in [M]$. Then, he sends oracle functions of them to the verifier, who answers with uniformly sampled values $\gamma, \delta \in \mathbb{K}$.

3. **Grand-Product Oracles:** The prover computes the grand-product polynomials Z_j for each argument $j \in [M + M' + M'']$ and sends oracle functions of them to the verifier.

4. **Verification:** For each $j \in [M]$ and all $x \in G$, the verifier checks that constraints in Eq. (18) hold; for each $j \in [M+1, M+M']$, constraints in Eq. (14) hold; and for each $j \in [M+M'+1, M+M'+M'']$, constraints in Eq. (16) hold.

$\mathcal{P}$		$\mathcal{V}$
	$\xrightarrow{\{[f_{i,j}], [t_{i,j}], [f_j^{\text{sel}}], [t_j^{\text{sel}}]\}_{i,j}}$	
	$\xleftarrow{\{\alpha, \beta\}}$	
	$\xrightarrow{\{[h_{1,1}], [h_{2,1}], \ldots, [h_{1,M}], [h_{2,M}]\}}$	
	$\xleftarrow{\{\gamma, \delta\}}$	
	$\xrightarrow{\{[Z_1], \ldots, [Z_{M+M'+M''}]\}}$	

Using Theorem 3 and the Parallel Repetition Theorem for polynomial IOPs [BCS16], [Gol98] we obtain the following result. Use M_1, M_2, M_3 to refer to the number of simple, vector and selected vector lookups. We have $M = M_1 + M_2 + M_3$. For the multiset equality scenario, analogously define M_1', M_2', M_3'.

Corollary 1 (Soundness of Protocol 1)**.** *Let $\varepsilon_{\mathsf{Plo}}, \varepsilon_{\mathsf{MulEq}}, \varepsilon_{\mathsf{Con}}$ be the soundness for a single invocation of the protocols asserting the lookup, multiset equality and connection arguments, respectively. Then if the prover interacts with the verifier in Protocol 1 and causes it accept with probability greater than:*

$$\varepsilon_{\mathsf{Args}} := \left(n\frac{1}{|\mathbb{K}|}\right)^{M_1} \left(n\frac{N}{|\mathbb{K}|}\right)^{M_2} \left(n\frac{N}{|\mathbb{K}|}\right)^{M_3} \cdot (\varepsilon_{\mathsf{MulEq}})^{M'} \cdot (\varepsilon_{\mathsf{Con}})^{M''},$$

then each of the M lookup, M' multiset equality and M'' connection arguments get satisifed.

2.5 On the Quotient Polynomial

In the vanilla STARK protocol, the quotient polynomial Q (Eq. 8) is computed by adjusting the degree of the rational functions:

$$q_i(X) := \frac{C_i(\text{tr}_1(X), \ldots, \text{tr}_N(X), \text{tr}_1(gX), \ldots, \text{tr}_N(gX))}{Z_G(X)},$$

to a sufficiently large power of two D with the help of two random values $\mathfrak{a}_i, \mathfrak{b}_i$. The sum of the resulting polynomials $\hat{q}_i := (\mathfrak{a}_i X^{D-\deg(q_i)-1} + \mathfrak{b}_i) \cdot q_i(X)$ is precisely Q.

There are two major issues with the previous definition of the quotient polynomial: (1) it leads to an amount of uniformly sampled values $\mathfrak{a}_i, \mathfrak{b}_i$ proportional to the number of constraints; and (2) [Sta21] (or any other source, as far as we know) does not provide a proof of why the degree adjustment is necessary at all. On the other side, problematic (1) becomes a real problem when the proof size should be as small as possible and therefore this made us explore sound alternatives.

Therefore, we obtain a single random value $\mathfrak{a} \in \mathbb{K}$ and define the quotient polynomial as a random linear combination of the rational functions q_i as follows:

$$Q(X) := \sum_{i=1}^{\ell} \mathfrak{a}^{i-1} q_i(X).$$

Note that we not only we remove the degree adjustment of the q_i's, but also use powers of a uniformly sampled value $\mathfrak{a}$ instead of sampling one value per constraint. A proof that this alternative way of computing the quotient polynomial is sound was carefully analized in Theorem 7 of [Hab22b] (and based on Theorem 7.2 of [BCI+20]). Importantly, the soundness bound of this alternative version is linearly increased by the number of constraints ℓ, so we might assume from now on that ℓ is sublinear in $|\mathbb{K}|$ to ensure the security of protocols.

2.6 Controlling the Constraint Degree wih Intermediate Polynomials

In the vanilla STARK protocol, the initial set of constraints that one attest to compute the proof over is of unbounded degree. However, when one arrives at the point after computing the quotient polynomial Q, it should be split into polynomials of degree lower than n to ensure the same redundancy is added as with the trace column polynomials tr_i for a sound application of the FRI protocol. In this section we explain an alternative for this process and propose the split to happen "at the beginning" and not "at the end" of the proof computation.

Therefore, we will proceed with this approach assuming that the arguments in Section 1.5 are included among the initial set of constraints. The constraints imposed by the grand-products polynomials Z_i of multiset equalities and lookups are of known degree: degree 2 for the former and degree 3 for the latter. Based on this information, we will propose a splitting procedure that allows for polynomial constraints up to degree 3, but will split any exceeding it.

Say the initial set of polynomial constraints $\mathcal{C} = \{C_1, \dots, C_\ell\}$ contain a constraint of total degree greater or equal than 4. For instance, say that we have $\mathcal{C} = \{C_1, C_2\}$ with:

$$\begin{aligned} C_1(X_1, X_2, X_3, X'_1, X'_2, X'_3) &= X_1 \cdot X_2 \cdot X'_2 \cdot X'_3 - X_3^3, \\ C_2(X_1, X_2, X_3, X'_1, X'_2, X'_3) &= X_2 - 7 \cdot X'_1 + X'_3. \end{aligned} \tag{26}$$

Now, instead of directly computing the (unbounded) quotient polynomial Q and then doing the split, we will follow the following process:

1. Split the constraints of degree $t \geq 4$ into $\lceil t/3 \rceil$ constraints of degree lower or equal than 3 through the introduction of one formal variable and one constraint per split.

2. Compute the rational functions q_i. Notice the previous step restricts the degree of the q_i's to be lower than $2n$.

3. Compute the quotient polynomial $Q \in \mathbb{F}_{<2n}[X]$ and then split it into (at most) two polynomials Q_1 and Q_2 of degree lower than n as follows:

$$Q(X) = Q_1(X) + X^n \cdot Q_2(X), \tag{27}$$

   where Q_1 is obtained by taking the first n coefficients of Q and Q_2 is obtained by taking the last n coefficients (filling with zeros if necessary).

Remark 2. Here, we might have that Q_2 is identically equal to 0. This is in contrast with the technique used for the split in Eq. (9), where the quotient polynomial Q is distributed uniformly across each of the trace quotient polynomials Q_i.

This process will "control" the degree of Q so that it will be always of degree lower than $2n$.

Following with the example in Eq. (26), we rename C_2 to C_3 and introduce the formal variable Y_1 and the constraint:

$$C_2(X_1, X_2, X_3, X_1', X_2', X_3', Y_1) = X_1 \cdot X_2 - Y_1, \tag{28}$$

Now, in order to compute the rational functions q_i, we have to compose C_2 not only with the trace column polynomials tr_i but also with additional polynomials corresponding with the introduced variables Y_i. We will denote these polynomials as im_i and refer to them as *intermediate polynomials*.

Hence, the set of constraints in (26) gets augmented to the following set:

$$C_1(X_1, X_2, X_3, X_1', X_2', X_3', Y_1) = Y_1 \cdot X_2' \cdot X_3' - X_3^3,$$

$$C_2(X_1, X_2, X_3, X_1', X_2', X_3', Y_1) = X_1 \cdot X_2 - Y_1,$$

$$C_3(X_1, X_2, X_3, X_1', X_2', X_3', Y_1) = X_2 - 7 \cdot X_1' + X_3',$$

where we include the variable Y_1 in C_3 for notation simplicity. Note that now what we have is two constraints of degree lower than 3, but we have added one extra variable and constraint to take into account.

Discussing more in depth the tradeoff generated between the two approaches, we have for one side that $\deg(Q) = \max_i\{\deg(q_i)\} = \max_i\{\deg(C_i)(n-1) - |G|\}$. Denote by $i_{\max}$ the index of the q_i where the maximum is attained. Then, the number of polynomials S in the split of Q is equal to:

$$\left\lceil \frac{\deg(Q)}{n} \right\rceil = \left\lceil \frac{\deg(C_{i_{\max}})(n-1) - |G|}{n} \right\rceil = \deg(C_{i_{\max}}) + \left\lceil -\frac{|G|}{n} \right\rceil,$$

which is equal to either $\deg(C_{i_{\max}}) - 1$ or $\deg(C_{i_{\max}})$.

We must compare this number with the number of additional constraints (or polynomials) added in our proposal. So, on the other side we have that the overall number of constraints $\tilde{\ell}$ is:

$$\sum_{i=1}^{\ell} \left\lceil \frac{\deg(C_i)}{3} \right\rceil,$$

with $\tilde{\ell} \geq \ell$.

We conclude that the appropriate approach should be chosen based on the minimum value between $\tilde{\ell} - \ell$ and S. Specifically, if the goal is to minimize the number of polynomials in the proof generation, then the vanilla STARK approach should be taken if $\min\left\{\tilde{\ell} - \ell, S\right\} = S$, and our approach should be taken if $\min\left\{\tilde{\ell} - \ell, S\right\} = \tilde{\ell} - \ell$.

Example 5. To give some concrete numbers, let us compare both approaches using the following set of constraints:

$$C_1(X_1, X_2, X_3, X_4, X_1') = X_1 \cdot X_2^2 \cdot X_3^4 \cdot X_4 - X_1',$$

$$C_2(X_1, X_2, X_3) = X_1 \cdot X_2^3 + X_3^3,$$

$$C_3(X_2, X_3, X_4, X_2') = X_2^3 \cdot X_3 \cdot X_4 + X_2',$$

In the vanilla STARK approach, we obtain $S = 8$. On the other side, using the early splitting technique explained before, by substituting $X_1 \cdot X_2^2$ by Y_1 and $X_2 \cdot X_3 \cdot X_4$ by Y_2 we transform the previous set of constraints into an equivalent one having all constraints of degree less or equal than 3. This reduction only introduces 2 additional constraints:

$$C_1(X_1', Y_1, Y_2) = Y_1^2 \cdot Y_2 - X_1',$$

$$C_2(X_2, X_3, Y_1) = Y_1 \cdot X_2 + X_3^3,$$

$$C_3(X_2, X_2', Y_2) = Y_2 \cdot X_2^2 + X_2',$$

$$C_4(X_1, X_2, Y_1) = Y_1 - X_1 \cdot X_2^2$$

$$C_5(X_2, X_3, X_4, Y_2) = Y_2 - X_2 \cdot X_3 \cdot X_4$$

Henceforth, the early splitting technique is convenient in this case, introducing 3 new polynomials instead of the 7 that proposes the vanilla STARK approach.

However, early splittings are not unique. That is, we can reduce the degree of the constraints differently, giving producing more polynomials and worsening our previous splitting in terms of numbers of polynomials. For example, the following set of constraints (achieved by substituting $X_1 \cdot X_2^2$ by Y_1, X_3^3 by Y_2, $X_3 \cdot X_4$ by Y_3 and X_2^3 by Y_4) is equivalent to the former ones, but in this case we added 4 extra polynomial constraints:

$$C_1(X_1', Y_1, Y_2, Y_3) = Y_1 \cdot Y_2 \cdot Y_3 - X_1',$$

$$C_2(X_2, Y_1, Y_2) = Y_1 \cdot X_2 + Y_2,$$

$$C_3(X_2', Y_3, Y_4) = Y_3 \cdot Y_4 + X_2',$$

$$C_4(X_1, X_2, Y_1) = Y_1 - X_1 \cdot X_2^2,$$

$$C_5(X_3, Y_2) = Y_2 - X_3^3,$$

$$C_6(X_3, X_4, Y_3) = Y_3 - X_3 \cdot X_4,$$

$$C_7(X_2, Y_4) = Y_4 - X_2^3$$

2.7 FRI Polynomial Computation

Recall from Section 1.4 the F polynomial was computed as follows:

$$\begin{aligned}\mathsf{F}(X) := \quad & \sum_{i \in I_1} \varepsilon_i^{(1)} \cdot \frac{\mathrm{tr}_i(X) - \mathrm{tr}_i(z)}{X - z} + \sum_{i \in I_2} \varepsilon_i^{(2)} \cdot \frac{\mathrm{tr}_i(gX) - \mathrm{tr}_i(gz)}{X - gz} \\ & + \sum_{i=1}^{S} \varepsilon_i^{(3)} \cdot \frac{Q_i(X) - Q_i(z^S)}{X - z^S},\end{aligned}$$

where $I_1 = \{i \in [N] \colon \mathrm{tr}_i(z) \in \mathsf{Evals}(z)\}$, $I_2 = \{i \in [N] \colon \mathrm{tr}_i(gz) \in \mathsf{Evals}(gz)\}$ and $\varepsilon_i^{(1)}, \varepsilon_j^{(2)}, \varepsilon_k^{(3)} \in \mathbb{K}$ for all $i \in I_1, j \in I_2, k \in [S]$. This way of computing the F polynomial has again (see Section 2.5) the issue that the number of random values sent from the verifier is proportional to the number of polynomials involved in the previous sum.

We will therefore compute the F polynomial by requesting two random values $\varepsilon_1, \varepsilon_2 \in \mathbb{K}$ instead, using ε_1 to compute the part regarding evaluations at z and gz separately, and finally mixing it all together with ε_1.

Following with the previous example, we define polynomials $\mathsf{F}_1, \mathsf{F}_2 \in \mathbb{K}_{<n}[X]$:

$$\mathsf{F}_1(X) := \sum_{i \in I_1} \varepsilon_2^{i-1} \cdot \frac{\mathrm{tr}_i(X) - \mathrm{tr}_i(z)}{X - z} + \sum_{i=1}^{S} \varepsilon_2^{|I_1|+i-1} \cdot \frac{Q_i(X) - Q_i(z)}{X - z}$$

$$\mathsf{F}_2(X) := \sum_{i \in I_2} \varepsilon_2^{i-1} \cdot \frac{\mathrm{tr}_i(gX) - \mathrm{tr}_i(gz)}{X - gz},$$

and then we set $\mathsf{F}(X) := \mathsf{F}_1(X) + \varepsilon_1 \cdot \mathsf{F}_2(X)$. Note that since $\varepsilon_1, \varepsilon_2$ are uniformly sampled elements, then so is $\varepsilon_1 \cdot \varepsilon_2^i$ for all $i \geq 0$.

A commonly used alternative version of the F polynomial computation in practice involves requesting a single random value $\varepsilon \in \mathbb{K}$ and directly computing

$$\widetilde{\mathsf{F}}(X) := \sum_{i \in I_1} \varepsilon^{i-1} \cdot \frac{\mathrm{tr}_i(X) - \mathrm{tr}_i(z)}{X - z} + \sum_{i \in I_2} \varepsilon^{|I_1|+i-1} \cdot \frac{\mathrm{tr}_i(gX) - \mathrm{tr}_i(gz)}{X - gz} + \sum_{i=1}^{S} \varepsilon^{|I_1|+|I_2|+i-1} \cdot \frac{Q_i(X) - Q_i(z^S)}{X - z^S}.$$

This version has the disadvantage of not being computable in parallel like the previous version, so we prefer the first option (even if it means increasing the proof size by one field element). Specifically, when the powers of ε_2 are being computed, it is possible to compute the polynomials F_1 and F_2 both sequentially and in parallel, while $\widetilde{\mathsf{F}}$ can only be computed sequentially after the computation of the powers of ε.

3 Our eSTARK Protocol

3.1 Extended Algebraic Intermediate Representation (eAIR)

In this section we introduce the notion of eAIRs and eAIR satisfiability as a natural extension of the well-studied AIRs [BBHR19]. Informally speaking, an eAIR is an AIR whose expressiveness is extended with more type of allowed constraints. In the following recall that $G = \langle g \rangle$ is a multiplicative subgroup of $\mathbb{F}$ of order n.

Definition 7 (AIR and AIR Satisfiability)**.** Given polynomials $p_1, \dots, p_M \in \mathbb{F}[X]$, an *algebraic intermediate representation (AIR)* A is a set of algebraic constraints $\{C_1, \dots, C_K\}$ such that each C_i is a polynomial over $\mathbb{F}[X_1, \dots, X_M, X'_1, \dots, X'_M]$. For each C_i, the first half variables $X_1, \dots, X_M$ will be replaced by polynomials $p_1(X), \dots, p_M(X)$, whereas the second half variables $X'_1, \dots, X'_M$ will be replaced by polynomials $p_1(gX), \dots, p_M(gX)$.

Moreover, we say that polynomials $p_1, \dots, p_M$ *satisfy a given AIR* $\mathsf{A} = \{C_1, \dots, C_K\}$ if and only if for each $i \in [K]$ we have that:

$$C_i(p_1(x), \dots, p_M(x), p_1(gx), \dots, p_M(gx)) = 0, \forall x \in G.$$

Remark 2. There are two main simplifications between our definition for AIR and the definition for AIR in [Sta21]: (1) we define constraints only over the "non-shifted" and "shifted-by-one" version of the corresponding polynomials, i.e., $p_i(X)$ and $p_i(gX)$, respectively; and (2) we enforce constraints to vanish over the whole G and not over a subset of it. The following definitions can, however, support a more generic version.

Now, we extend the definition of an AIR by allowing the arguments defined in Section 2.4 as new types of available constraints.

Definition 8 (Extended AIR)**.** Given polynomials $p_1, \dots, p_M \in \mathbb{F}[X]$, an *extended algebraic intermediate representation (eAIR)* eA is a set of constraints $\mathsf{eA} = \{C_1, \dots, C_K\}$ such that each C_i can be one of the following form:

(a) A polynomial over $\mathbb{F}[X_1, \dots, X_M, X'_1, \dots, X'_M]$ as in Def. 7.

(b) A positive integer R_i, a set of $2R_i$ polynomials $C_{i,j}$ over $\mathbb{F}[X_1, \dots, X_M]$ and two selectors $f_i^{\mathrm{sel}}, t_i^{\mathrm{sel}}$ over $\mathbb{F}[X]$ (recall that $f_i^{\mathrm{sel}}(x), t_i^{\mathrm{sel}}(x) \in \{0,1\}$ for all $x \in G$).

(c) An integer $S_i \in [M]$, a subset of S_i polynomials $p_{(1)}, \dots, p_{(S_i)} \in \mathbb{F}[X]$ from the set $\{p_1, \dots, p_M\}$ and S_i more polynomials $S_{\sigma_1}, \dots, S_{\sigma_{S_i}}$.

Finally, we refer to the set constraints of the form described in (a) as the set of *identity constraints* of eA and to the set of constraints of the form described by eiher (b) or (c) as the set of *non-identity constraints* of eA.

Definition 8 aims to capture the arguments in Section 2.4 in a slightly more generic way. Here, the polynomials subject to these arguments are not exactly static polynomials $p_1, \dots, p_M$, but they are generated as a polynomial combination between $p_1, \dots, p_M$.

In what follows, we use $\overline{\mathrm{P}}$ as a shorthand for $(p_1, \dots, p_M)$ and denote by $C \circ \overline{\mathrm{P}}$ to the univariate polynomial over $\mathbb{F}[X]$ resulting from the substitution of each of the variables X_i, X'_i of the constraint $C \in \mathbb{F}[X_1, \dots, X_M, X'_1, \dots, X'_M]$ by $p_i(X), p_i(gX)$, respectively. That is $C \circ \overline{\mathrm{P}}$ is the polynomal $C(p_1(X), \dots, p_M(X), p_1(gX), \dots, p_M(gX))$.

Definition 9 (Extended AIR Satisfiability)**.** We say that polynomials $p_1, \dots, p_M \in \mathbb{F}[X]$ *satisfy a given eAIR* $\mathsf{eA} = \{C_1, \dots, C_K\}$ if and only if for each $i \in [K]$ one and only one of the following is true for all $x \in G$:

$$(C_i \circ \overline{\mathrm{P}})(x) = 0,$$

$$f_i^{\mathrm{sel}}(x) \cdot ((C_{i,1} \circ \overline{\mathrm{P}})(x), \dots, (C_{i,R_i} \circ \overline{\mathrm{P}})(x)) \in t_i^{\mathrm{sel}}(x) \cdot ((C_{i,R_i+1} \circ \overline{\mathrm{P}})(x), \dots, (C_{i,2R_i} \circ \overline{\mathrm{P}})(x)),$$

$$f_i^{\mathrm{sel}}(x) \cdot ((C_{i,1} \circ \overline{\mathrm{P}})(x), \dots, (C_{i,R_i} \circ \overline{\mathrm{P}})(x)) \doteq t_i^{\mathrm{sel}}(x) \cdot ((C_{i,R_i+1} \circ \overline{\mathrm{P}})(x), \dots, (C_{i,2R_i} \circ \overline{\mathrm{P}})(x)),$$

$$(p_{(1)}(x), \dots, p_{(S_i)}(x)) \propto (S_{\sigma_1}(x), \dots, S_{\sigma_{S_i}}(x)).$$

3.2 The Setup Phase

During the protocol of Section 3.3, both the prover and the verifier will need to have access to a set of preprocessed polynomials $\mathrm{pre}_i \in \mathbb{F}[X]$. In particular, the prover will need to have full access to them, either in coefficient or in evaluation form, in order to be able to correctly generate the proof. On the other hand, the verifier will only need to have acces to a subset the evaluations of these polynomial over the domain H.

To this end, in our protocol we assume the existence of a phase, known as the *setup phase*, that is prior to the protocol message exchange but after the particular statement to be proven (or equivalently, the set of constraints that describe the statement) is fixed. In the setup phase, the preprocessed polynomials are computed and the prover and the verifier receive different information regearding them. Particularly, the setup phase, with input a set of polynomial constraints, consists on the following steps:

1. The trace LDE of each preprocessed polynomial is computed.

2. The Merkle tree of the set of preprocessed polynomials is computed.

3. Finally, the complete tree is sent to the prover and its corresponding root is sent to the verifier. This way, when the verifier needs to compute the evaluation of any preprocessed polynomial over $h \in H$, he can request it from the prover, who will

respond with the evaluation along with its corresponding Merkle tree path. The verifier then verifies the accuracy of the information received by using the root of the tree.

Remark 3. Since the computational effort of the setup phase is greater than $\mathcal{O}(\log(n))$, we cannot include this phase as part of the verifier description if we want our protocol to satisfy verifier scalability.

Note that the setup phase does not include a measure for the verifier to be sure that the computation of Merkle tree of preprocessed polynomials is correct. However, as the setup phase input is basically the set of polynomial constraints representing the problem's statement (something that the verifier also knows), the verifier can run at any time the setup phase to check the validity of the computations.

Moreover, both the prover and the verifier will need to have access to the evaluations over H of the vanishing polynomial $Z_G(X) := X^n - 1$ and the first Lagrange polynomial $L_1(X) := \frac{g(X^n-1)}{n(X-g)}$. However, Z_G and L_1 will appear later on in the protocol, and although in principle we do not consider them preprocessed polynomials, they are publicly known and therefore included in the Merkle tree computation of the setup phase.

3.3 Our IOP for eAIR

Before the start of the protocol, we assume the prover and verifier have fixed a specific eAIR instance $\mathsf{eA} = \left\{\widetilde{C}_1, \ldots, \widetilde{C}_{T'}\right\}$ and that the constraints of eA are ordered as follows: first, the identity constraints, then the lookup arguments, followed by the permutation arguments, and finally, the connection arguments. Additionally, we assume that the setup phase has been successfully executed.

Throughout the description of the protocol, we use the following useful notation:

- Let N, R be two non-negative integers. We set N to be the number of trace column polynomials and R to be the number of preprocessed polynomals.
- Among the set of polynomial constraints eA, we denote by $M, M', M'' \in \mathbb{Z}^*$ the number of lookup arguments, permutation arguments and connection arguments, respectively.
- The prover parameters pp are composed by the finite field $\mathbb{F}$, the domains G and H, the field extension $\mathbb{K}$, the eAIR instance eA, all the public values, the set of committed polynomials and the Merkle tree of preprocessed polynomals.
- The verifier parameters vp are composed by the finite field $\mathbb{F}$, the domains G and H, the field extension $\mathbb{K}$, the eAIR instance eA, all the public values and the Merkle tree's root of preprocessed polynomials.

Our IOP for eAIR, which can be seen as an extension of the DEEP-ALI protocol [BGKS19], is a follows.

Protocol 2 (IOP for eAIR)**.** The protocol starts with a set of trace column polynomials $\mathrm{tr}_1, \ldots, \mathrm{tr}_N \in \mathbb{F}_{<n}[X]$ and preprocessed polynomials $\mathrm{pre}_1, \ldots, \mathrm{pre}_R \in \mathbb{F}_{<n}[X]$. The following protocol is used by a prover to prove to a verifier that polynomials $\mathrm{tr}_1, \ldots, \mathrm{tr}_N, \mathrm{pre}_1, \ldots, \mathrm{pre}_R$ satisfy eA:

1. **Trace Column Oracles:** The prover sets oracle functions $[\mathrm{tr}_1], \ldots, [\mathrm{tr}_N]$ to $\mathrm{tr}_1, \ldots, \mathrm{tr}_N \in \mathbb{F}_{<n}[X]$ for the verifier, who responds with uniformly sampled values $\alpha, \beta \in \mathbb{K}$. During this step, the prover also computes the intermediate polynomials resulting from the subset of identity constraints. Let $K \in \mathbb{Z}^*$ be the number of these polynomials, denoted as $\mathrm{im}_i \in \mathbb{F}_{<n}[X]$, where $i \in [K]$. It is important to recall that new identity

constraints must also be considered when introducing intermediate polynomials, as demonstrated in Eq. (28). As additional intermediate polynomials may be introduced in Round 3, the prover will set oracles for $\mathrm{im}_1, \dots, \mathrm{im}_K$ in that round.

2. **Plookup Oracles:** As explained by Section 2.4, the prover, if needed:

   - Uses α to reduce both vector lookups and vector permutations into simple (possibly selected) lookups and permutations.
   - Uses β to reduce both selected lookups and selected permutations into simple (non-selected) lookups and permutations.

   After the previous two reductions, the prover computes the Plookup polynomials $h_{i,1}, h_{i,2} \in \mathbb{K}_{<n}[X]$ for each lookup argument, with $i \in [M]$. Then, he sets oracle functions $[h_{1,1}]$,$[h_{1,2}]$,...,$[h_{M,1}]$,$[h_{M,2}]$ for the verifier, who answers with uniformly sampled values $\gamma, \delta \in \mathbb{K}$.

3. **Grand-Product and Intermediate Oracles:** The prover uses γ, δ to compute the grand-product polynomials $Z_i \in \mathbb{K}_{<n}[X]$ for each argument, with $i \in [M + M' + M'']$.

   Importantly, some identity constraints induced by the constraints asserting the validity of the connection argument's grand-product polynomials might be of degree greater or equal than 4. Therefore, following Section 2.6, the prover split these constraints into multiple constraints of degree at most 3 by the introduction of intermediate polynomials. Let $K' \in \mathbb{Z}^*$ be the number of introduced intermediate polynomials, denoted as $\mathrm{im}_{K'+i} \in \mathbb{K}[X]$, where $i \in [K']$.

   The prover sets oracle functions $[Z_1]$,...,$[Z_{M+M'+M''}]$ and $[\mathrm{im}_1]$,...,$[\mathrm{im}_{K+K'}]$ for the verifier. The verifier answers with a uniformly sampled value $\mathfrak{a} \in \mathbb{K}$.

   At this point, the original eAIR $\mathsf{eA} = \{\widetilde{C}_1, \dots, \widetilde{C}_{T'}\}$ has been reduced to an AIR $\mathsf{A} = \{C_1, \dots, C_T\}$, with $T \geq T'$, so we continue by executing the DEEP-ALI protocol over A with the modifications mentioned in Sections 2.5 and 2.6.

   Remark 3. Rounds 2 and 3 are skipped by both the prover and the verifier if the eAIR instance eA is in fact an AIR. In such case, Round 4 follows from Round 1.

4. **Trace Quotient Oracles:** The prover computes the polynomial $Q(X) \in \mathbb{K}[X]$:

$$Q(X) := \sum_{i=1}^{T} \mathfrak{a}^{i-1} \frac{(C_i \circ \overline{\mathrm{P}})(X)}{Z_G(X)},$$

   where we have now used $\overline{\mathrm{P}}$ to denote the sequence of polynomials containing $\mathrm{tr}_1, \dots, \mathrm{tr}_N$, $\mathrm{pre}_1, \dots, \mathrm{pre}_R$, $h_{1,1}, h_{1,2}, \dots, h_{M,1}, h_{M,2}$, $Z_1, \dots, Z_{M+M'+M''}$ and finally $\mathrm{im}_1, \dots, \mathrm{im}_{K+K'}$. Therefore, $(C_i \circ \overline{\mathrm{P}})(X)$ is the (univariate) polynomal resulting from the composition of the (multivariate) polynomial C_i and the non-shifted and shifted version of the (univariate) polynomials in $\overline{\mathrm{P}}$. Then, the prover splits Q into two trace quotient polynomials Q_1 and Q_2 of degree lower than n, and sets their oracles $[Q_1]$ and $[Q_2]$ for the verifier. Polynomials Q_1 and Q_2 satisfy the following:

$$Q_1(X) + X^n \cdot Q_2(X) = \sum_{i=1}^{T} \mathfrak{a}^{i-1} \frac{(C_i \circ \overline{\mathrm{P}})(X)}{Z_G(X)}. \tag{29}$$

5. **DEEP Query Answers:** The verifier samples a uniformly sampled DEEP query $z \in \mathbb{K} \backslash (G \cup H)$. Note that the verifier prohibits $z \in G$ to enable evaluation of the right-hand side of Eq. (29) and prohibits $z \in H$ to enable evaluation of the

polynomial F, defined in Round 6, during the FRI protocol. Then, the verifier queries either the oracles set by the prover in previous rounds or the oracles to the preprocessed polynomials to obtain the evaluation sets $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$. Two observations should be made: first, it is possible for a polynomial to have evaluations in both $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$; and second, evaluations of preprocessed polynomials are also included within $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$. The verifier then sends to the prover uniformly sampled values $\varepsilon_1, \varepsilon_2 \in \mathbb{K}$.

6. **FRI Protocol:** Among the set of polynomals in $\overline{\mathrm{P}}$, respectively denote by f_i and h_i those whose evaluation respectively belong to $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$. The prover computes the polynomials $\mathsf{F}_1, \mathsf{F}_2 \in \mathbb{K}[X]$:

$$\mathsf{F}_1(X) := \sum_{i=1}^{|\mathsf{Evals}(z)|} \varepsilon_2^{i-1} \frac{f_i(X) - f_i(z)}{X - z}$$

$$\mathsf{F}_2(X) := \sum_{i=1}^{|\mathsf{Evals}(gz)|} \varepsilon_2^{i-1} \frac{h_i(X) - h_i(gz)}{X - gz},$$

after which he computes the polynomial $\mathsf{F}(X) := \mathsf{F}_1(X) + \varepsilon_1 \cdot \mathsf{F}_2(X)$. Finally, the prover and the verifier run the FRI protocol to prove the low degreeness of F, which starts by setting oracle acces to F for the verifier.

7. **Verification:** Similar to the vanilla STARK verifier, the verifier proceeds as follows:

   (a) **ALI Consistency.** Checks that the trace quotient polynomials Q_1 and Q_2 are consistent with the trace column polynomials $\mathrm{tr}_1, \dots, tr_N$, the preprocessed polynomials $\mathrm{pre}_1, \dots, \mathrm{pre}_R$, the plookup-related polynomials $h_{1,1}$,$h_{1,2}$, $\dots$, $h_{M,1}$,$h_{M,2}$, the grand-product polynomials $Z_1, \dots, Z_{M+M'+M''}$ and the intermediate polynomials $\mathrm{im}_1, \dots, \mathrm{im}_{K+K'}$. The verifier achieves so by means of Eq. (29) and the evaluations in $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$.

   (b) **Batched FRI Verification.** It runs the batched FRI verification procedure on the polynomial F.

   If either (a) or (b) fail at any point, the verifier aborts and rejects. Otherwise, the verifier accepts.

3.4 From a STIK to a Non-Interactive STARK

As described in Definition 4, transforming the STIK of Section 3.3 to a STARK is very straightforward. First, oracles sent from the prover to the verifier in each round are substituted by a single Merkle tree root as explained in Section 2.1. So, for instance, oracle access to $[\mathrm{tr}_1], \dots, [\mathrm{tr}_N]$ set by the prover in the first round is substituted by the Merkle tree root of the Merkle tree containing the evaluations of $\mathrm{tr}_1, \dots, \mathrm{tr}_N$ over the domain H. Second, instead of letting the verifier asking a query to a set of polynomial oracles $[f_1], \dots, [f_N]$ at the same point v, the verifier asks for these evaluations to the prover and the prover answers with $f_1(v), \dots, f_N(v)$ together with the Merkle path associated with them. We do not specify the specific hash function used in the computation of each Merkle tree, but we only state that this hash function does not have any relation to the hash function used to render the protocol non-interactive, as long as its output space is in the appropriate field.

We denote by seed to the concatenation of the initial eAIR instance $\mathsf{eA} = \left\{\widetilde{C}_1, \dots, \widetilde{C}_{T'}\right\}$, all the public values and the Merkle tree's root of preprocessed polynomials. This value will act as the seed to the hash function $\mathcal{H}$ in order to simulate the first verifier's message.

3.4.1 Full Protocol Description

We split the protocol's description between the prover algorithm and verifier algorithm and compose each round in the prover algorithm of the computation the verifier's challenges (via Fiat-Shamir) and the actual messages computed by the prover. We reuse the notation and assumptions from Section 3.3.

PROVER ALGORITHM

Round 1: Trace Column Polynomials

Given the trace column polynomials $\mathrm{tr}_1, \mathrm{tr}_2, \ldots, \mathrm{tr}_N \in \mathbb{F}_{<n}[X]$, the prover commits to them, as explained in Section 2.1, computing their associated Merkle root:

$$\begin{array}{|lcr|} \mathcal{P}(\mathrm{pp}) & & FS \\ \hline & \xrightarrow{\mathrm{MTR}(\mathrm{tr}_1, \ldots, \mathrm{tr}_N)} & \end{array}$$

During this step, the prover also computes the intermediate polynomials $\mathrm{im}_1, \ldots, \mathrm{im}_K$ resulting from the subset of identity constraints.

Round 2: Plookup Polynomials

First of all we initialize a transcript instance $\mathsf{transcript}$. Now, the prover adds the seed and the Merkle root for the commitment of the trace polynomials $\{\mathrm{tr}_i\}_i$ into the transcript

$$\mathsf{add}_{\mathsf{transcript}}(\mathsf{seed}, \mathrm{MTR}(\{\mathrm{tr}_i\}_i))$$

and extracts the corresponding challenges $\alpha, \beta \in \mathbb{K}$ to be sent to the verifier:

$$\alpha = \mathsf{extract}_1(\mathsf{transcript}), \quad \beta = \mathsf{extract}_2(\mathsf{transcript}).$$

Using α and β, the prover computes the Plookup polynomials $h_{i,1}, h_{i,2} \in \mathbb{K}_{<n}[X]$ for each lookup argument, with $i \in [M]$, and commits to them:

$$\begin{array}{|lcr|} \mathcal{P}(\mathrm{pp}) & & FS \\ \hline & \xrightarrow{\mathrm{MTR}(\mathrm{tr}_1, \ldots, \mathrm{tr}_N)} & \\ & \xleftarrow{\{\alpha, \beta\}} & \\ & \xrightarrow{\mathrm{MTR}(h_{1,1}, h_{1,2}, \ldots, h_{M,1}, h_{M,2})} & \end{array}$$

Round 3: Grand-Product and Intermediate Polynomials

The prover adds the Merkle root of the tree for the commitment of the set $\{h_{i,j}\}_{i,j}$ of polynomials to the transcript

$$\mathsf{add}_{\mathsf{transcript}}(\mathrm{MTR}(\{h_{i,j}\}_{i,j})),$$

and extracts the corresponding challenges $\gamma, \delta \in \mathbb{K}$ to be sent to the verifier

$$\gamma = \mathsf{extract}_1(\mathsf{transcript}), \quad \delta = \mathsf{extract}_2(\mathsf{transcript}).$$

The prover uses γ, δ to compute the grand-product polynomials $Z_i \in \mathbb{K}_{<n}[X]$ for each argument, with $i \in [M + M' + M'']$. The prover also computes the remaining intermediate polynomials $\mathrm{im}_{K+1}, \ldots, \mathrm{im}_{K+K'}$.

In this round, the prover commits to both the grand-product polynomials and all the intermediate polynomials. To save one element in the proof, the prover uses the same Merkle tree for both the grand-product and the intermediate polynomials:

$$
\begin{array}{lcr}
\mathcal{P}(\mathrm{pp}) & & FS \\
\hline
& \xrightarrow{\mathrm{MTR}(\mathrm{tr}_1, \ldots, \mathrm{tr}_N)} & \\
& \xleftarrow{\{\alpha, \beta\}} & \\
& \xrightarrow{\mathrm{MTR}(h_{1,1}, h_{1,2}, \ldots, h_{M,1}, h_{M,2})} & \\
& \xleftarrow{\{\gamma, \delta\}} & \\
& \xrightarrow{\mathrm{MTR}(Z_1, \ldots, Z_{M+M'+M''}, \mathrm{im}_1, \ldots, \mathrm{im}_{K+K'})} &
\end{array}
$$

Round 4: Trace Quotient Polynomials

The prover adds the Merkle roots of the trees for the commitments by the sets $\{Z_i\}_i$ and $\{\mathrm{im}_i\}_i$ of polynomials to the transcript

$$\mathsf{add}_{\mathsf{transcript}}(\mathrm{MTR}(\{Z_i\}_i), \mathrm{MTR}(\{\mathrm{im}_i\}_i)),$$

and extracts the corresponding challenge $\mathfrak{a} \in \mathbb{K}$ to be sent to the verifier

$$\mathfrak{a} = \mathsf{extract}_1(\mathsf{transcript}).$$

The prover uses $\mathfrak{a}$ to compute the quotient polynomial $Q \in \mathbb{K}_{<2n}[X]$:

$$Q(X) := \sum_{i=1}^{T} \mathfrak{a}^{i-1} \frac{(C_i \circ \overline{\mathrm{P}})(X)}{Z_G(X)},$$

and splits it into two polynomials Q_1 and Q_2 of degree lower than n as in Eq. (27). Then, the prover commits to these two polynomials:

$$
\begin{array}{lcr}
\mathcal{P}(\mathrm{pp}) & & FS \\
\hline
& \xrightarrow{\mathrm{MTR}(\mathrm{tr}_1, \ldots, \mathrm{tr}_N)} & \\
& \xleftarrow{\{\alpha, \beta\}} & \\
& \xrightarrow{\mathrm{MTR}(h_{1,1}, h_{1,2}, \ldots, h_{M,1}, h_{M,2})} & \\
& \xleftarrow{\{\gamma, \delta\}} & \\
& \xrightarrow{\mathrm{MTR}(Z_1, \ldots, Z_{M+M'+M''}, \mathrm{im}_1, \ldots, \mathrm{im}_{K+K'})} & \\
& \xleftarrow{\mathfrak{a}} & \\
& \xrightarrow{\mathrm{MTR}(Q_1, Q_2)} &
\end{array}
$$

Recall that Q_1, Q_2 satisfy the following relation with Q:

$$Q_1(X) + X^n \cdot Q_2(X) = \sum_{i=1}^{T} \mathfrak{a}^{i-1} \frac{(C_i \circ \overline{\mathrm{P}})(X)}{Z_G(X)}. \tag{30}$$

Round 5: DEEP Query Answers

The prover adds the Merkle root of the tree for the commitments of the Q_1 and Q_2 polynomials to the transcript

$$\mathsf{add}_{\mathsf{transcript}}(\mathrm{MTR}(Q_1, Q_2)),$$

and extracts the corresponding challenge $z \in \mathbb{K}$ to be sent to the verifier

$$z = \mathsf{extract}_1(\mathsf{transcript}).$$

If z falls either in G or H, then we introduce a counter as an extra input to the hash function and keep incrementing it until $z \in \mathbb{K} \backslash (G \cup H)$. The probability that z is not of the expected form is proportional to $(|G| + |H|)/|\mathbb{K}|$, which is sufficiently small for all practical purposes.

The prover computes the evaluation sets $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$:

$$\begin{array}{lcr}
\mathcal{P}(\mathrm{pp}) & & FS \\
\hline
& \xrightarrow{\mathrm{MTR}(\mathrm{tr}_1, \ldots, \mathrm{tr}_N)} & \\
& \xleftarrow{\{\alpha, \beta\}} & \\
& \xrightarrow{\mathrm{MTR}(h_{1,1}, h_{1,2}, \ldots, h_{M,1}, h_{M,2})} & \\
& \xleftarrow{\{\gamma, \delta\}} & \\
& \xrightarrow{\mathrm{MTR}(Z_1, \ldots, Z_{M+M'+M''}, \mathrm{im}_1, \ldots, \mathrm{im}_{K+K'})} & \\
& \xleftarrow{\mathfrak{a}} & \\
& \xrightarrow{\mathrm{MTR}(Q_1, Q_2)} & \\
& \xleftarrow{z} & \\
& \xrightarrow{\{\mathsf{Evals}(z), \mathsf{Evals}(gz)\}} &
\end{array}$$

Round 6: FRI Protocol

The prover adds the the sets $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$ into the transcript

$$\mathsf{add}_{\mathsf{transcript}}(\mathsf{Evals}(g), \mathsf{Evals}(gz)),$$

and extracts the corresponding challenges $\varepsilon_1, \varepsilon_2 \in \mathbb{K}$ to be sent to the verifier

$$\varepsilon_1 = \mathsf{extract}_1(\mathsf{transcript}), \quad \varepsilon_2 = \mathsf{extract}_2(\mathsf{transcript}).$$

Among the set of polynomals in $\overline{\mathrm{P}}$, respectively denote by f_i and h_i those whose evaluation respectively belong to $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$. The prover computes the polynomials $\mathsf{F}_1, \mathsf{F}_2 \in \mathbb{K}[X]$:

$$\mathsf{F}_1(X) := \sum_{i=1}^{|\mathsf{Evals}(z)|} \varepsilon_2^{i-1} \frac{f_i(X) - f_i(z)}{X - z}$$

$$\mathsf{F}_2(X) := \sum_{i=1}^{|\mathsf{Evals}(gz)|} \varepsilon_2^{i-1} \frac{h_i(X) - h_i(gz)}{X - gz},$$

after which he computes the polynomial $\mathsf{F}(X) := \mathsf{F}_1(X) + \varepsilon_1 \cdot \mathsf{F}_2(X)$. Finally, the prover executes the (non-interactive version of the) FRI protocol to prove the low degreeness of the F polynomials, after which he obtains a FRI proof π_{FRI}.

$$
\begin{array}{lcr}
\mathcal{P}(\mathrm{pp}) & & FS \\
\hline
& \xrightarrow{\mathrm{MTR}(\mathrm{tr}_1,\dots,\mathrm{tr}_N)} & \\
& \xleftarrow{\{\alpha,\beta\}} & \\
& \xrightarrow{\mathrm{MTR}(h_{1,1},h_{1,2},\dots,h_{M,1},h_{M,2})} & \\
& \xleftarrow{\{\gamma,\delta\}} & \\
& \xrightarrow{\mathrm{MTR}(Z_1,\dots,Z_{M+M'+M''},\mathrm{im}_1,\dots,\mathrm{im}_{K+K'})} & \\
& \xleftarrow{\mathfrak{a}} & \\
& \xrightarrow{\mathrm{MTR}(Q_1,Q_2)} & \\
& \xleftarrow{z} & \\
& \xrightarrow{\{\mathsf{Evals}(z),\mathsf{Evals}(gz)\}} & \\
& \xleftarrow{\varepsilon_1,\varepsilon_2} & \\
& \xrightarrow{\pi_{\mathrm{FRI}}} &
\end{array}
$$

The full description of the transcript can be found in Figure 7 of Appendix A
The prover returns:

$$
\pi_{\mathrm{eSTARK}} = \begin{pmatrix} \mathrm{MTR}(\mathrm{tr}_1,\dots,\mathrm{tr}_N), \mathrm{MTR}(h_{1,1},h_{1,2},\dots,h_{M,1},h_{M,2}), \\ \mathrm{MTR}(Z_1,\dots,Z_{M+M'+M''},\mathrm{im}_1,\dots,\mathrm{im}_{K+K'}), \\ \mathrm{MTR}(Q_1,Q_2), \mathsf{Evals}(z), \mathsf{Evals}(gz), \pi_{\mathrm{FRI}} \end{pmatrix}
$$

VERIFIER ALGORITHM

In order to verify the STARK, the verifier performs the following steps:

1. Checks that $\mathrm{MTR}(\mathrm{tr}_1,\dots,\mathrm{tr}_N)$, $\mathrm{MTR}(h_{1,1},h_{1,2},\dots,h_{M,1},h_{M,2})$, $\mathrm{MTR}(Z_1,\dots,Z_{M+M'+M''}, \mathrm{im}_1,\dots,\mathrm{im}_{K+K'})$, $\mathrm{MTR}(Q_1,Q_2)$ are all in $\mathbb{K}$.

2. Checks that every element in both $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$ is from $\mathbb{K}$.

3. Computes the challenges $\alpha,\beta,\gamma,\delta,\mathfrak{a},z,\varepsilon_1,\varepsilon_2 \in \mathbb{K}$ from the elements in seed and π_{eSTARK} (except for π_{FRI}, which is used to compute the challenges within FRI).

4. **ALI Consistency.** Checks that the trace quotient polynomials Q_1 and Q_2 are consistent with the trace column polynomials $\mathrm{tr}_1,\dots,tr_N$, the preprocessed polynomials $\mathrm{pre}_1,\dots,\mathrm{pre}_R$, the plookup-related polynomials $h_{1,1}$,$h_{1,2}$, $\dots$, $h_{M,1}$,$h_{M,2}$, the grand-product polynomials $Z_1,\dots,Z_{M+M'+M''}$ and the intermediate polynomials $\mathrm{im}_1,\dots,\mathrm{im}_{K+K'}$. The verifier achieves so by means of Eq. (30) and the evaluations contained in $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$.

5. **Batched FRI Verification.** Using π_{FRI}, it runs the batched FRI verification procedure on the polynomial F.

If either any of the previous steps fail at any point, the verifier aborts and rejects. Otherwise, the verifier accepts.

References

[AFK21] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. Cryptology ePrint Archive, Report 2021/1377, 2021. https://eprint.iacr.org/2021/1377.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, Heidelberg, August 2019.

[BCI+20] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed-solomon codes. Cryptology ePrint Archive, Report 2020/654, 2020. https://eprint.iacr.org/2020/654.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. Cryptology ePrint Archive, Report 2016/116, 2016. https://eprint.iacr.org/2016/116.

[BGKS19] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. Cryptology ePrint Archive, Report 2019/336, 2019. https://eprint.iacr.org/2019/336.

[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.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021.

[Gol98] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer, 1998.

[GW20] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020. https://eprint.iacr.org/2020/315.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[Hab22a] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Report 2022/1530, 2022. https://eprint.iacr.org/2022/1530.

[Hab22b] Ulrich Haböck. A summary on the FRI low degree test. Cryptology ePrint Archive, Report 2022/1216, 2022. https://eprint.iacr.org/2022/1216.

[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.

[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.

[PFM+22] Luke Pearson, Joshua Fitzgerald, Héctor Masip, Marta Bellés-Muñoz, and Jose Luis Muñoz-Tapia. PlonKup: Reconciling PlonK with plookup. Cryptology ePrint Archive, Report 2022/086, 2022. https://eprint.iacr.org/2022/086.

[Sta21] StarkWare. ethSTARK documentation. Cryptology ePrint Archive, Report 2021/582, 2021. https://eprint.iacr.org/2021/582.

[VP19] Alexander Vlasov and Konstantin Panarin. Transparent polynomial commitment scheme with polylogarithmic communication complexity. Cryptology ePrint Archive, Report 2019/1020, 2019. https://eprint.iacr.org/2019/1020.

A Full Protocol Descriptions

In this section we provide "skeleton" descriptions of the STARKs constructed in this article. By skeleton, we mean that we only give those descriptions in terms of the communication complexity between both the prover and the verifier, avoiding details such as the computational work carried on by any of the two parties in each step.

A.1 FRI

Define for this section the public parameters pp as $(d, H, \mathbb{F}, \mathbb{K})$

$\mathcal{P}(p_0, \mathrm{pp})$ $\qquad\qquad\qquad\qquad$ $\mathcal{V}(\mathrm{pp})$

$$\xrightarrow{\mathrm{MTR}(p_0)}$$
$$\xleftarrow{v_0}$$
$$\xrightarrow{\mathrm{MTR}(p_1)}$$
$$\xleftarrow{v_1}$$
$$\vdots$$
$$\xleftarrow{v_{k-1}}$$
$$\xrightarrow{p_k}$$
$$\xleftarrow{h_1}$$
$$\xrightarrow[\{\mathrm{MTP}(p_0(h_1)),\ldots,\mathrm{MTP}(p_{k-1}(h_1^{2^{k-1}}))\}]{\{p_0(h_1),\ldots,p_{k-1}(h_1^{2^{k-1}})\}}$$
$$\xrightarrow[\{\mathrm{MTP}(p_0(-h_1)),\ldots,\mathrm{MTP}(p_{k-1}(-h_1^{2^{k-1}}))\}]{\{p_0(-h_1),\ldots,p_{k-1}(-h_1^{2^{k-1}})\}}$$
$$\vdots$$
$$\xleftarrow{h_s}$$
$$\xrightarrow[\{\mathrm{MTP}(p_0(h_s)),\ldots,\mathrm{MTP}(p_{k-1}(h_s^{2^{k-1}}))\}]{\{p_0(h_s),\ldots,p_{k-1}(h_s^{2^{k-1}})\}}$$
$$\xrightarrow[\{\mathrm{MTP}(p_0(-h_s)),\ldots,\mathrm{MTP}(p_{k-1}(-h_s^{2^{k-1}}))\}]{\{p_0(-h_s),\ldots,p_{k-1}(-h_s^{2^{k-1}})\}}$$

(a) FRI transcript.

$\mathcal{P}(\{f_i\}_i, \mathrm{pp})$ $\qquad\qquad\qquad\qquad$ $\mathcal{V}(\mathrm{pp})$

$$\xrightarrow{\mathrm{MTR}(p_0)}$$
$$\xleftarrow{v_0}$$
$$\xrightarrow{\mathrm{MTR}(p_1)}$$
$$\xleftarrow{v_1}$$
$$\vdots$$
$$\xleftarrow{v_{k-1}}$$
$$\xrightarrow{p_k}$$
$$\xleftarrow{h_1}$$
$$\xrightarrow[\{\mathrm{MTP}(p_0(h_1)),\ldots,\mathrm{MTP}(p_{k-1}(h_1^{2^{k-1}}))\}]{\{p_0(h_1),\ldots,p_{k-1}(h_1^{2^{k-1}})\}}$$
$$\xrightarrow[\{\mathrm{MTP}(p_0(-h_1)),\ldots,\mathrm{MTP}(p_{k-1}(-h_1^{2^{k-1}}))\}]{\{p_0(-h_1),\ldots,p_{k-1}(-h_1^{2^{k-1}})\}}$$
$$\xrightarrow[\{\mathrm{MTP}(f_0(h_1)),\ldots,\mathrm{MTP}(f_N(h_1))\}]{\{f_0(h_1),f_1(h_1),\ldots,f_N(h_1)\}}$$
$$\xrightarrow[\{\mathrm{MTP}(f_0(-h_1)),\ldots,\mathrm{MTP}(f_N(-h_1))\}]{\{f_0(-h_1),f_1(-h_1),\ldots,f_N(-h_1)\}}$$
$$\vdots$$
$$\xleftarrow{h_s}$$
$$\xrightarrow[\{\mathrm{MTP}(p_0(h_s)),\ldots,\mathrm{MTP}(p_{k-1}(h_s^{2^{k-1}}))\}]{\{p_0(h_s),\ldots,p_{k-1}(h_s^{2^{k-1}})\}}$$
$$\xrightarrow[\{\mathrm{MTP}(p_0(-h_s)),\ldots,\mathrm{MTP}(p_{k-1}(-h_s^{2^{k-1}}))\}]{\{p_0(-h_s),\ldots,p_{k-1}(-h_s^{2^{k-1}})\}}$$
$$\xrightarrow[\{\mathrm{MTP}(f_0(h_s)),\ldots,\mathrm{MTP}(f_N(h_s))\}]{\{f_0(h_s),f_1(h_s),\ldots,f_N(h_s)\}}$$
$$\xrightarrow[\{\mathrm{MTP}(f_0(-h_s)),\ldots,\mathrm{MTP}(f_N(-h))\}]{\{f_0(-h_s),f_1(-h_s),\ldots,f_N(-h_s)\}}$$

(b) Batched FRI transcript.

Figure 5: FRI transcripts of Section 1.3.

A.2 Vanilla STARK

In this section, we will outline the STARK protocol step by step. The following definitions will be employed in the protocol:

- We call *prover parameters*, and denote them by pp, to the finite field $\mathbb{F}$, the domains

G and H, the field extension $\mathbb{K}$, the set of constraints, and the set of polynomial evaluations that the prover has access to.

- We call *verifier parameters*, and denote them by vp, to the finite field $\mathbb{F}$, the domains G and H, the field extension $\mathbb{K}$, the set of constraints, and the set of polynomial evaluations that the verifier has access to.

Protocol 3. The protocol starts with a set of constraints $\mathcal{C} = \{C_1, \ldots, C_T\}$ and a set of polynomials $\mathrm{tr}_1, \ldots, \mathrm{tr}_N \in \mathbb{F}[X]$ (presumably) satisfying them. The prover and verifier proceed as follows:

1. The prover sends oracle functions $[\mathrm{tr}_1], \ldots, [\mathrm{tr}_N]$ for $\mathrm{tr}_1, \ldots, \mathrm{tr}_N$ to the verifier:

$$\begin{array}{lcr} \mathcal{P}(\mathrm{pp}) & & \mathcal{V}(\mathrm{vp}) \\ \hline & \xrightarrow{\quad [\mathrm{tr}_1], \ldots, [\mathrm{tr}_N] \quad} & \end{array}$$

2. The verifier samples uniformly random values $\mathfrak{a}_1, \mathfrak{b}_1, \ldots, \mathfrak{a}_T, \mathfrak{b}_T \in \mathbb{K}$ and sends them to the prover:

$$\begin{array}{lcr} \mathcal{P}(\mathrm{pp}) & & \mathcal{V}(\mathrm{vp}) \\ \hline & \xrightarrow{\quad [\mathrm{tr}_1], \ldots, [\mathrm{tr}_N] \quad} & \\ & \xleftarrow{\quad \{\mathfrak{a}_1, \mathfrak{b}_1, \ldots, \mathfrak{a}_T, \mathfrak{b}_T\} \quad} & \end{array}$$

3. The prover computes the quotient polynomial Q, after which he computes the trace quotient polynomials $Q_1, \ldots, Q_S$ and sends their oracle representatives $[Q_1], \ldots, [Q_S]$ to the verifier.

$$\begin{array}{lcr} \mathcal{P}(\mathrm{pp}) & & \mathcal{V}(\mathrm{vp}) \\ \hline & \xrightarrow{\quad [\mathrm{tr}_1], \ldots, [\mathrm{tr}_N] \quad} & \\ & \xleftarrow{\quad \{\mathfrak{a}_1, \mathfrak{b}_1, \ldots, \mathfrak{a}_T, \mathfrak{b}_T\} \quad} & \\ & \xrightarrow{\quad [Q_1], \ldots, [Q_S] \quad} & \end{array}$$

4. The verifier samples a uniformly random value $z \in \mathbb{K} \backslash (G \cup \bar{H})$ and sends it to the prover:

$$\begin{array}{lcr} \mathcal{P}(\mathrm{pp}) & & \mathcal{V}(\mathrm{vp}) \\ \hline & \xrightarrow{\quad [\mathrm{tr}_1], \ldots, [\mathrm{tr}_N] \quad} & \\ & \xleftarrow{\quad \{\mathfrak{a}_1, \mathfrak{b}_1, \ldots, \mathfrak{a}_T, \mathfrak{b}_T\} \quad} & \\ & \xrightarrow{\quad [Q_1], \ldots, [Q_S] \quad} & \\ & \xleftarrow{\quad z \quad} & \end{array}$$

5. The prover computes the sets $\mathsf{Evals}(z)$ and $\mathsf{Evals}(gz)$ of trace column polynomials evaluations. Then, he sends $\mathsf{Evals}(z), \mathsf{Evals}(gz)$ and $Q_i(z^S)$ for all $i \in [S]$ to the verifier:

$$
\begin{array}{lcr}
\mathcal{P}(\mathrm{pp}) & & \mathcal{V}(\mathrm{vp}) \\
\hline
& \xrightarrow{\quad [\mathrm{tr}_1], \dots, [\mathrm{tr}_N] \quad} & \\
& \xleftarrow{\quad \{\mathfrak{a}_1, \mathfrak{b}_1, \dots, \mathfrak{a}_T, \mathfrak{b}_T\} \quad} & \\
& \xrightarrow{\quad [Q_1], \dots, [Q_S] \quad} & \\
& \xleftarrow{\quad z \quad} & \\
& \xrightarrow{\quad \{\mathsf{Evals}(z), \mathsf{Evals}(gz), Q_1(z^S), \dots, Q_S(z^S)\} \quad} &
\end{array}
$$

6. Recalling that $I_1 = \{i \in [N] \colon \mathrm{tr}_i(z) \in \mathsf{Evals}(z)\}$ and $I_2 = \{i \in [N] \colon \mathrm{tr}_i(gz) \in \mathsf{Evals}(gz)\}$, the verifier sends uniformly random values $\varepsilon_i^{(1)}, \varepsilon_j^{(2)}, \varepsilon_k^{(3)} \in \mathbb{K}$, where $i \in I_1, j \in I_2, k \in [S]$, to the prover:

$$
\begin{array}{lcr}
\mathcal{P}(\mathrm{pp}) & & \mathcal{V}(\mathrm{vp}) \\
\hline
& \xrightarrow{\quad [\mathrm{tr}_1], \dots, [\mathrm{tr}_N] \quad} & \\
& \xleftarrow{\quad \{\mathfrak{a}_1, \mathfrak{b}_1, \dots, \mathfrak{a}_T, \mathfrak{b}_T\} \quad} & \\
& \xrightarrow{\quad [Q_1], \dots, [Q_S] \quad} & \\
& \xleftarrow{\quad z \quad} & \\
& \xrightarrow{\quad \{\mathsf{Evals}(z), \mathsf{Evals}(gz), Q_1(z^S), \dots, Q_S(z^S)\} \quad} & \\
& \xleftarrow{\quad \{\{\varepsilon_i^{(1)}\}_{i \in I_1}, \{\varepsilon_j^{(2)}\}_{j \in I_2}, \{\varepsilon_k^{(3)}\}_{k \in [S]}\} \quad} &
\end{array}
$$

7. The prover computes the F polynomial. Then, the prover initiates the FRI message exchange along with the verifier to prove the low degreeness of the F polynomial to the verifier. The prover sends the resulting FRI proof π_{FRI} to the verifier:

$$
\begin{array}{lcr}
\mathcal{P}(\mathrm{pp}) & & \mathcal{V}(\mathrm{vp}) \\
\hline
& \xrightarrow{\quad [\mathrm{tr}_1], \dots, [\mathrm{tr}_N] \quad} & \\
& \xleftarrow{\quad \{\mathfrak{a}_1, \mathfrak{b}_1, \dots, \mathfrak{a}_T, \mathfrak{b}_T\} \quad} & \\
& \xrightarrow{\quad [Q_1], \dots, [Q_S] \quad} & \\
& \xleftarrow{\quad z \quad} & \\
& \xrightarrow{\quad \{\mathsf{Evals}(z), \mathsf{Evals}(gz), Q_1(z^S), \dots, Q_S(z^S)\} \quad} & \\
& \xleftarrow{\quad \{\{\varepsilon_i^{(1)}\}_{i \in I_1}, \{\varepsilon_j^{(2)}\}_{j \in I_2}, \{\varepsilon_k^{(3)}\}_{k \in [S]}\} \quad} & \\
& \xrightarrow{\quad \pi_{\mathrm{FRI}} \quad} &
\end{array}
$$

8. The verifier runs the three checks commented in Section 1.4: trace consistency, FRI consistency and batched consistency. If any of the checks fail, the verifier rejects. Otherwise, the verifier accepts.

$\mathcal{P}(\mathrm{pp})$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathcal{V}(\mathrm{vp})$

$$\xrightarrow{[\mathrm{tr}_1], \dots, [\mathrm{tr}_N]}$$

$$\xleftarrow{\{\mathfrak{a}_1, \mathfrak{b}_1, \dots, \mathfrak{a}_T, \mathfrak{b}_T\}}$$

$$\xrightarrow{[Q_1], \dots, [Q_S]}$$

$$\xleftarrow{z}$$

$$\xrightarrow{\{\mathsf{Evals}(z), \mathsf{Evals}(gz), Q_1(z^S), \dots, Q_S(z^S)\}}$$

$$\xleftarrow{\{\{\varepsilon_i^{(1)}\}_{i \in I_1}, \{\varepsilon_j^{(2)}\}_{j \in I_2}, \{\varepsilon_k^{(3)}\}_{k \in [S]}\}}$$

.... FRI: Commit Phase ....

$$\xrightarrow{\mathrm{MTR}(\mathsf{F} = p_0)}$$

$$\xleftarrow{v_0}$$

$$\xrightarrow{\mathrm{MTR}(p_1)}$$

$$\xleftarrow{v_1}$$

$$\vdots$$

$$\xleftarrow{v_{k-1}}$$

$$\xrightarrow{p_k}$$

.... FRI: Query Phase ....

$$\xleftarrow{h_1}$$

$$\xrightarrow[\{\mathrm{MTP}(p_0(h_1)), \dots, \mathrm{MTP}(p_{k-1}(h_1^{2^{k-1}}))\}]{\{p_0(h_1), \dots, p_{k-1}(h_1^{2^{k-1}})\}}$$

$$\xrightarrow[\{\mathrm{MTP}(p_0(-h_1)), \dots, \mathrm{MTP}(p_{k-1}(-h_1^{2^{k-1}}))\}]{\{p_0(-h_1), \dots, p_{k-1}(-h_1^{2^{k-1}})\}}$$

$$\xrightarrow[\{\mathrm{MTP}(\mathsf{Evals}(h_1)) \dots, \mathrm{MTP}(Q_S(h_1^S))\}]{\{\mathsf{Evals}(h_1), Q_1(h_1^S), \dots, Q_S(h_1^S)\}}$$

$$\xrightarrow[\{\mathrm{MTP}(\mathsf{Evals}(-h_1)) \dots, \mathrm{MTP}(Q_S((-h_1)^S))\}]{\{\mathsf{Evals}(-h_1), Q_1((-h_1)^S), \dots, Q_S((-h_1)^S)\}}$$

$$\xleftarrow{h_s}$$

$$\xrightarrow[\{\mathrm{MTP}(p_0(h_s)), \dots, \mathrm{MTP}(p_{k-1}(h_s^{2^{k-1}}))\}]{\{p_0(h_s), \dots, p_{k-1}(h_s^{2^{k-1}})\}}$$

$$\xrightarrow[\{\mathrm{MTP}(p_0(-h_s)), \dots, \mathrm{MTP}(p_{k-1}(-h_s^{2^{k-1}}))\}]{\{p_0(-h_s), \dots, p_{k-1}(-h_s^{2^{k-1}})\}}$$

$$\xrightarrow[\{\mathrm{MTP}(\mathsf{Evals}(h_s)) \dots, \mathrm{MTP}(Q_S(h_s^S))\}]{\{\mathsf{Evals}(h_s), Q_1(h_s^S), \dots, Q_S(h_s^S)\}}$$

$$\xrightarrow[\{\mathrm{MTP}(\mathsf{Evals}(-h_s)) \dots, \mathrm{MTP}(Q_S((-h_s)^S))\}]{\{\mathsf{Evals}(-h_s), Q_1((-h_s)^S), \dots, Q_S((-h_s)^S)\}}$$

Figure 6: Full STARK transcript of Section 1.4

A.3 eSTARK

In this section, we provide the full eSTARK transcript. We reuse definitions from Section 3.3.

$\mathcal{P}(\mathrm{pp})$ $\qquad\qquad\qquad\qquad$ $\mathcal{V}(\mathrm{vp})$

$$\xrightarrow{\mathrm{MTR}(\mathrm{tr}_1, \ldots, \mathrm{tr}_N)}$$

$$\xleftarrow{\{\alpha, \beta\}}$$

$$\xrightarrow{\mathrm{MTR}(h_{1,1}, h_{1,2}, \ldots, h_{M,1}, h_{M,2})}$$

$$\xleftarrow{\{\gamma, \delta\}}$$

$$\xrightarrow{\mathrm{MTR}(Z_1, \ldots, Z_{M+M'+M''}, \mathrm{im}_1, \ldots, \mathrm{im}_{K+K'})}$$

$$\xleftarrow{\mathfrak{a}}$$

$$\xrightarrow{\mathrm{MTR}(Q_1, Q_2)}$$

$$\xleftarrow{z}$$

$$\xrightarrow{\{\mathsf{Evals}(z), \mathsf{Evals}(gz)\}}$$

$$\xleftarrow{\varepsilon_1, \varepsilon_2}$$

....................FRI: Commit Phase....................

$$\xrightarrow{\mathrm{MTR}(\mathsf{F} = p_0)}$$

$$\xleftarrow{v_0}$$

$$\xrightarrow{\mathrm{MTR}(p_1)}$$

$$\xleftarrow{v_1}$$

$$\vdots$$

$$\xleftarrow{v_{k-1}}$$

$$\xrightarrow{p_k}$$

....................FRI: Query Phase....................

$$\xleftarrow{h_1}$$

$$\xrightarrow[\{\mathrm{MTP}(p_0(h_1)), \ldots, \mathrm{MTP}(p_{k-1}(h_1^{2^{k-1}}))\}]{\{p_0(h_1), \ldots, p_{k-1}(h_1^{2^{k-1}})\}}$$

$$\xrightarrow[\{\mathrm{MTP}(p_0(-h_1)), \ldots, \mathrm{MTP}(p_{k-1}(-h_1^{2^{k-1}}))\}]{\{p_0(-h_1), \ldots, p_{k-1}(-h_1^{2^{k-1}})\}}$$

$$\xrightarrow{\{\mathsf{Evals}(h_1), \mathrm{MTP}(\mathsf{Evals}(h_1))\}}$$

$$\xrightarrow{\{\mathsf{Evals}(-h_1), \mathrm{MTP}(\mathsf{Evals}(-h_1))\}}$$

$$\vdots$$

$$\xleftarrow{h_s}$$

$$\xrightarrow[\{\mathrm{MTP}(p_0(h_s)), \ldots, \mathrm{MTP}(p_{k-1}(h_s^{2^{k-1}}))\}]{\{p_0(h_s), \ldots, p_{k-1}(h_s^{2^{k-1}})\}}$$

$$\xrightarrow[\{\mathrm{MTP}(p_0(-h_s)), \ldots, \mathrm{MTP}(p_{k-1}(-h_s^{2^{k-1}}))\}]{\{p_0(-h_s), \ldots, p_{k-1}(-h_s^{2^{k-1}})\}}$$

$$\xrightarrow{\{\mathsf{Evals}(h_s), \mathrm{MTP}(\mathsf{Evals}(h_s))\}}$$

$$\xrightarrow{\{\mathsf{Evals}(-h_s), \mathrm{MTP}(\mathsf{Evals}(-h_s))\}}$$

Figure 7: Full STARK transcript of Section 3.4.

B Code Naming Equivalences

Aiming for clear equations and explanations, we prefer not to use the same names appearing in code. Henceforth, in this section, we show several equivalence tables in order to make the reader easier to overcome the problem of reading different names between the code and the technical documentation.

Technical Documentation	**Code**
α	`u`
β	`defVal`
γ	`gamma`
δ	`betta`
$\mathfrak{a}$	`vc`
z	`xi`
ε_1	`v1`
ε_2	`v2`

Figure 8: Challenges Naming Equivalences.

Technical Documentation	**Code**
$Q(X) := Q_1(X) + X^n \cdot Q_2(X)$	`q`
$Q_1(X)$	`qq1`
$Q_2(X)$	`qq2`
$\mathsf{F}(X)$	`friPol`

Figure 9: Polynomials Naming Equivalences.