RapidUp: Multi-Domain Permutation Protocol for Lookup Tables

Héctor Masip Ardevol[1*], Jordi Baylina Melé[2], Daniel Lubarov[3] and José L. Muñoz-Tapia[1]

[1] Universitat Politècnica de Catalunya, Spain, {hector.masip,jose.luis.munoz}@upc.edu
[2] Polygon Hermez, jordi@baylina.cat
[3] Mir Protocol, daniel@lubarov.com

Abstract. SNARKs for some standard cryptographic primitives tend to be plenty designed with SNARK-unfriendly operations such as XOR. Previous protocols such as [GW20] worked around this problem by the introduction of lookup arguments. However, these protocols were only appliable over the same circuit. RapidUp is a protocol that solves this limitation by unfolding the grand-product polynomial into two (equivalent) polynomials of the same size. Morevoer, a generalization of previous protocols is presented by the introduction of selectors.

Keywords: lookup · tables · permutation · SNARK · protocol

1 Introduction

At the hearth of the universal SNARK [BCTV13] $\mathcal{PlonK}$ [GWC19] there is a protocol proving that the vectors formed by the evaluations of two polynomials over an evaluation domain are a permutation of each other. This protocol is latter on generalized for multiple polynomials and used by the prover in the main $\mathcal{PlonK}$ protocol to prove that circuit gates are correctly connected.

This permutation check is accomplished by the construction of a "grand-product" polynomial. At a high level, the grand-product is defined, cumulatively, as the product of the quotient of the evaluations of the involved polynomials. Then, if it is verified that this polynomial correctly cycles back to 1 after its last evaluation, it is proven that the original evaluations are a permutation of each other.

The principal problem of this protocol is that it cannot be instantiated when the polynomials in question have to be evaluated over distinct (possibly related) domains. That is, the permutation check between polynomials f and g is only described in the case where the evaluations of both polynomials are over the same domain H.

In this article, we generalize this grand-product-based protocol in two different ways. First, our protocol generalizes to the aforementioned context. That is, it will work even in the scenario where the polynomials are evaluated over distinct domains of, possibly, different size. Second, our protocol can be applied even where the original evaluations are not a permutation of each other, but rather they contain this permutation. The latter becomes useful in the case where the original values are not permuted "in its whole".

Let's see an example. Consider sets $f = \{5, 4, 2, 5\}$ and $g = \{9, 7, 5, 8, 5, 1, 6, 9, 4\}$ representing the values obtained by evaluating two polynomials (also denoted f and g) over domains of size m and n, respectively. Even if not relevant at this point, these domains will consist on subgroups of the multiplicative group of some finite field.

*Corresponding author.

From a generic point of view, our protocol can be used to prove that a certain subset[1] of f, denoted as f', is a permutation of a subset of g, denoted as g'. Continuing with the previous example, we can say that the subset $f' = \{5, 4, 5\}$ of the multiset f, and the subset $g' = \{5, 5, 4\}$ of the multiset g, are related through the permutation $(1, 3, 2)$. More formally, a permutation between the subsets f' and g' can be defined as a relation in which, for each index i of one of the subsets, say f', there exists some index j in the other subset g', such that the corresponding elements f'_i, g'_j are the same. Notice that while the length of f and g might be different, the length of the subsets being checked is the same, i.e., $|f'| = |g'|$.

The main idea of our protocol to "split" the grand-product polynomial Z into two other polynomials Z_f, Z_g, so that the latter ones represent the two members of the quotient in Z. That is, if at some evaluation point the polynomial Z evaluates to A/B, then also in some (possibly distinct) evaluation point the polynomials Z_f, Z_g will evaluate to A and B, respectively. Following the structure of Z, these two polynomials will require to be equal in is last evaluation.

To be able to define such polynomials, we introduce two additional sets f^{sel} and g^{sel}, known as the *selectors*. These selectors will allow us to select the indexes of each multiset f and g that we are going to be including in the subsets f' and g' for the permutation check. The elements of these selectors are defined to be 1 if the index is selected to be included in the subset, and 0 otherwise. Following with the previous example, the selectors are $f^{\mathsf{sel}} = \{1, 1, 0, 1\}$ and $g^{\mathsf{sel}} = \{0, 0, 1, 0, 1, 0, 0, 0, 1\}$.

Now, we define two grand-product polynomials Z_f, Z_g that evaluate, respectively, to the following values when evaluated on their respective evaluation domain:

$$\begin{aligned} Z_f &\rightarrow \{1, (5+\gamma), (5+\gamma)(4+\gamma), (5+\gamma)(4+\gamma), (5+\gamma)(4+\gamma)(5+\gamma)\} \\ Z_g &\rightarrow \{1, 1, 1, (5+\gamma), (5+\gamma), (5+\gamma)(5+\gamma), (5+\gamma)(5+\gamma), \\ &\quad (5+\gamma)(5+\gamma), (5+\gamma)(5+\gamma), (5+\gamma)(5+\gamma)(4+\gamma)\} \end{aligned}$$

where γ is a random element from the underlying field.

Observe that these polynomials end up in the same product value. Hence, the idea is that the subset $\{5, 4, 5\} \subset \{f_i\}$ is also a subset in $\{g_i\}$ if and only if Z_f and Z_g evaluate to the same element in their respective last element.

1.1 Use Case: Lookup Tables

A scenario in which this protocol can be applied is the case of lookup tables. This lookup technique reduces the native SNARK representation that some standard primitives, such as Keccak-256 [BDPA13], are made of. At a high level, this technique works by precomputing a lookup table of all the possible combinations of inputs/outputs for some particular "SNARK-unfriendly" operation[2]. Then, instead of representing the outsourced operation in some SNARK language, the prover claims that the corresponding witnesses exist in the table.

Plookup [GW20] provided a simple and efficient protocol of this idea. The main problem of their approach is that the involved vectors in lookup, the witness-related f and the table-related t, have to be of close length. In particular, their protocol can be applied to vectors f, t such that $|t| = |f| + 1$. This restriction was accommodated latter on by the alternating method described in [PFM+22]. Here, they could design a similar protocol for vectors f, t of the same length; which lead to an even simpler lookup protocol with less constraints.

However, there are some situations in which vectors f, t are not of close length. In a traditional scenario, one can find $|f|$ to be a small power of two and $|t|$ a big power

[1] It could be the case where $f' = f$ or $g' = g$ or both.

[2] One should typically think about bitwise operations, such as XOR.

of two. This is the bottleneck of the previous protocols, since one who wishes to use them and faces the aforementioned problem must accommodate the length of $|f|$ by, if appropriate, padding to f repetitions of its last element until its size fits with the protocol. This accommodation leads to efficiency loses in both the prover and the verifier part, but mainly in the prover side.

In this article we present a solution to the length accommodation issue. The protocol presented in Section 4 follows a similar nature than the protocols in [GW20] or even [BCG+18], in the context of efficient SNARK arithmetization of common operations.

1.2 Organization of the Article

This article is organized as follows. In Section 2 we provide some background and preliminaries. In Section 3 we define and discuss a more general version of a polynomial protocol, where we cover the need for polynomial protocols defined over distinct domains. Finally, in Section 4 we present the design and the security proof of our protocol.

2 Preliminaries

2.1 Terminology and Conventions

Given $n \in \mathbb{N}$, we will use $[n]$ as a shorthand for the set $\{1, 2, \ldots, n\}$. Given appropriate $n, m \in \mathbb{Z}_0^+$, we will use $[n, m]$ to denote the set $\{n, n+1, n+2, \ldots, m\}$.

We assume our field $\mathbb{F}$ is of prime order and denote by $\mathbb{F}^*$ its respective multiplicative group. We denote by $\mathbb{F}_{<n}[X]$ the set of univariate polynomials over $\mathbb{F}$ of degree smaller than n. When there is no room for confusion (e.g., when expressions only involve univariate polynomials), we will simply write f instead of $f(X)$.

For a polynomial $f \in \mathbb{F}_{<n}[X]$ and for $i \in [n]$ we denote $f_i := f(\omega^i)$. For a vector $f \in \mathbb{F}^n$, we also denote by f the polynomial in $\mathbb{F}_{<n}[X]$ with $f(\omega^i) = f_i$. We indistinguishably use f_i to denote the i-th element of vector f or the evaluation of polynomial f on input ω^i, as it is equivalent in the context of this article.

In our protocols, we typically denote by H a multiplicative subgroup of $\mathbb{F}$ of order N with generator ω. When we write H^2 we refer to the set that originates from taking the square to all the elements in H. Generally, we write H^{2^h} when we take the 2^h-th power. For a subset $H \subset \mathbb{F}^*$, we call *vanishing polynomial* to the polynomial defined as $Z_H(X) := \prod_{x \in H}(X - x)$.

Finally, we assume all algorithms described in this article receive as an implicit parameter the security parameter λ.

2.2 Lagrange Polynomials

For $i \in [n]$, we denote by $L_i \in \mathbb{F}_{<n}[X]$ the i-th *Lagrange polynomial* for H. That is, L_i satisfies $L_i(\omega^i) = 1$ and $L_i(\omega^j) = 0$ for $j \neq i$. It can be checked that the i-th Lagrange polynomial has the form:

$$L_i(X) = \frac{\omega^i \, (X^n - 1)}{n \, (X - \omega^i)}.$$

Lagrange polynomials are convenient when specifying a point check in an protocol restricted to some domain H. Specifically, these polynomials are suitable for converting point checks to checks in the entire subgroup H, as it is shown in Claim 1.

Claim 1. *Fix $i \in [n]$ and $f, g \in \mathbb{F}[X]$. Then $L_i(x)(f(x) - g(x)) = 0$ for all $x \in H$ if and only if $f(\omega^i) = g(\omega^i)$.*

The generator ω is also convenient for specifying constraints on "neighboring values". For example, the constraint $f(X \cdot \omega^2) = f(X \cdot \omega) + f(X)$ means that f's values follow the Fibonacci sequence between three consecutive points.

2.3 Polynomial Commitment Schemes

We define polynomial commitment schemes (PCS) in a general context similarly to [BDFG20]. Specifically, we define the scheme in a batched setting, allowing to query multiple polynomials at multiple points.

Definition 1. A *d-polynomial commitment scheme* is a triplet $\text{PCS} = (\mathsf{gen}, \mathsf{com}, \mathsf{open})$ such that:

- $\mathsf{gen}(d)$: Randomized algorithm that given a positive integer d outputs a structured reference string (SRS) srs.
$$\mathsf{srs} = \mathsf{gen}(d).$$
- $\mathsf{com}(f, \mathsf{srs})$: Deterministic algorithm that given a polynomial $f \in \mathbb{F}_{<d}[X]$ and an output srs of $\mathsf{gen}(d)$ returns a commitment cm of f.
$$\mathsf{cm} = \mathsf{com}(f, \mathsf{srs}).$$
- open: Public coin protocol between parties $\mathcal{P}_{\mathsf{PCS}}$ and $\mathcal{V}_{\mathsf{PCS}}$. $\mathcal{P}_{\mathsf{PCS}}$ is given $f_1, \ldots, f_k \in \mathbb{F}_{<d}[X]$. $\mathcal{P}_{\mathsf{PCS}}$ and $\mathcal{V}_{\mathsf{PCS}}$ are both given:
  1. Positive integers $d, t = \mathsf{poly}(\lambda)$.
  2. The SRS $\mathsf{srs} = \mathsf{gen}(d)$.
  3. $\mathsf{cm}_1, \ldots, \mathsf{cm}_k$: The alleged commitments to $f_1, \ldots f_k$.
  4. A subset $T = \{z_1, \ldots, z_t\} \subset \mathbb{F}$ consisting on the evaluation points.
  5. Some subsets $S_1, \ldots, S_k \subset T$ consisting on the specific evaluation points to be disclosed by each polynomial $f_1, \ldots, f_k$, respectively. Note that we do not require these subsets to be disjoint.
  6. $\left\{r_i \in \mathbb{F}_{<|S_i|}[X]\right\}_{i\in[k]}$: The polynomials describing the alleged openings, i.e., having $r_i(z) = f_i(z)$ for each $i \in [k]$ and $z \in S_i$.

  At the end of the protocol $\mathcal{V}_{\mathsf{PCS}}$ outputs Accept or Reject.

  The open protocol satisfies the following properties:

  - **Completeness**: Fix any $d, t = \mathsf{poly}(\lambda)$, $f_1, \ldots f_k \in \mathbb{F}_{<d}[X]$, $T = \{z_1, \ldots, z_t\} \subset \mathbb{F}$, $S_1, \ldots, S_k \subset T$ and $\left\{r_i \in \mathbb{F}_{<|S_i|}[X]\right\}_{i\in[k]}$. Suppose that for each $i \in [k]$, $\mathsf{cm}_i = \mathsf{com}(f_i, \mathsf{srs})$, and $Z_{S_i} \mid (f_i - r_i)$. Then, if $\mathcal{P}_{\mathsf{PCS}}$ follows open correctly with these values, $\mathcal{V}_{\mathsf{PCS}}$ outputs Accept with probability one.
  - **Knowledge soundness in the algebraic group model**: There exists an efficient extractor E such that for any algebraic adversary $\mathcal{A}$ and any choice of $d = \mathsf{poly}(\lambda)$, the probability of $\mathcal{A}$ winning the following game is negligible over the randomness of $\mathcal{A}, \mathcal{V}_{\mathsf{PCS}}$ and gen:
    1. Given d and $\mathsf{srs} = \mathsf{gen}(d)$, $\mathcal{A}$ outputs $\mathsf{cm}_1, \ldots, \mathsf{cm}_k$.
    2. E, given access to the messages of $\mathcal{A}$ during the previous step, outputs $f_1, \ldots, f_k \in \mathbb{F}_{<d}[X]$.
    3. $\mathcal{A}$ outputs $T = \{z_1, \ldots, z_t\} \subset \mathbb{F}$, $S_1, \ldots, S_k \subset T$ and $\left\{r_i \in \mathbb{F}_{<|S_i|}[X]\right\}_{i\in[k]}$.
    4. $\mathcal{A}$ takes part of $\mathcal{P}_{\mathsf{PCS}}$ in the open protocol with the inputs $\mathsf{cm}_1, \ldots, \mathsf{cm}_k$, T, $S_1, \ldots, S_k$, and $\{r_i\}_{i\in[k]}$.

5. $\mathcal{A}$ wins if:
   * $\mathcal{V}_{\mathsf{PCS}}$ outputs Accept at the end of the protocol.
   * For some $i \in [k]$, $Z_{S_i} \nmid (f_i - r_i)$.

3 Polynomial Protocols

In this section, we will define a variant of polynomial protocol defined in [GWC19]. Recall that the idea underneath the definition of a polynomial protocol is to cleanly capture and abstract the use of polynomial commitment schemes such as [KZG10] and [BBHR18]. In this protocol, the prover sends polynomials to a trusted party $\mathcal{I}$. The verifier may then ask $\mathcal{I}$ whether certain identities hold between the prover's polynomials, and additional predefined polynomials known to the verifier.

Compared to [GWC19], our polynomial protocols need to allow for the prover to interact directly with the verifier by sending messages from $\mathbb{F}$ that will be dependent on at least one of the messages send previously by the prover to $\mathcal{I}$. Moreover, each of the identities that the verifier ask $\mathcal{I}$ can be over distinct domains.

Definition 2. Fix positive integers d, D, ℓ, th. A (d, D, ℓ, t, h)-*polynomial protocol* over $\mathbb{F}$ is a multi-round protocol between a prover $\mathcal{P}_{\mathsf{poly}}$, a verifier $\mathcal{V}_{\mathsf{poly}}$ and an ideal (trusted) party $\mathcal{I}$ that proceeds as follows.

1. The protocol definition includes a set of preprocessed polynomials $g_1, \ldots, g_\ell \in \mathbb{F}_{<d}[X]$.
2. The messages of $\mathcal{P}_{\mathsf{poly}}$ that are sent to $\mathcal{I}$ are of the form f, for $f \in \mathbb{F}_{<d}[X]$. If $\mathcal{P}_{\mathsf{poly}}$ sends a message not of this form, the protocol is aborted.
3. The messages of $\mathcal{P}_{\mathsf{poly}}$ that are sent to $\mathcal{V}_{\mathsf{poly}}$ are of the form K, for $K \in \mathbb{F}$. If $\mathcal{P}_{\mathsf{poly}}$ sends a message not of this form, the protocol is aborted.
4. The messages of $\mathcal{V}_{\mathsf{poly}}$ that are sent to $\mathcal{P}_{\mathsf{poly}}$ are random coins.
5. At the end of the protocol, suppose $f_1, \ldots, f_t$ are the polynomials that were sent from $\mathcal{P}_{\mathsf{poly}}$ to $\mathcal{I}$. Also suppose $K_1, \ldots, K_h$ are the elements from $\mathbb{F}$ that were sent from $\mathcal{P}_{\mathsf{poly}}$ to $\mathcal{V}_{\mathsf{poly}}$. $\mathcal{V}_{\mathsf{poly}}$ may ask $\mathcal{I}$ if certain polynomial identities holds between $\{g_1, \ldots, g_\ell, f_1, \ldots, f_t\}$ and $\{K_1, \ldots, K_h\}$. Each identity is of the form:
$$F(X) := G(X, h_1(v_1(X)), \ldots, h_M(v_M(X)), K_1, \ldots, K_h) \equiv 0,$$
for some $h_i \in \{g_1, \ldots, g_\ell, f_1, \ldots, f_t\}$, $G \in \mathbb{F}[X, X_1, \ldots, X_M]$, $v_1, \ldots, v_M \in \mathbb{F}_{<d}[X]$ such that $F \in \mathbb{F}_{<D}[X]$ for every choice of $f_1, \ldots, f_t$ made by $\mathcal{P}_{\mathsf{poly}}$ when following the protocol correctly.
6. After receiving the answers from $\mathcal{I}$ regarding the identities, $\mathcal{V}_{\mathsf{poly}}$ outputs Accept if all identities hold, and outputs Reject otherwise.

We also define polynomial protocols for relations in the natural way.

Definition 3. Given a relation $\mathcal{R}$, a *polynomial protocol for* $\mathcal{R}$ is a polynomial protocol with the following additional properties.

1. At the beginning of the protocol, $\mathcal{P}_{\mathsf{poly}}$ and $\mathcal{V}_{\mathsf{poly}}$ are both additionally given an input x. The description of $\mathcal{P}_{\mathsf{poly}}$ assumes possession of ω such that $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$.
2. **Perfect Completeness**: If $\mathcal{P}_{\mathsf{poly}}$ follows the protocol correctly using a witness w for x, $\mathcal{V}_{\mathsf{poly}}$ accepts with probability one.

3. **Knowledge Soundness**: There exists an efficient extractor E, that given access to the messages of $\mathcal{P}_{\mathsf{poly}}$ to $\mathcal{I}$ and $\mathcal{V}_{\mathsf{poly}}$ outputs w such that, for any strategy of $\mathcal{P}_{\mathsf{poly}}$, the probability of the following event is $\mathsf{negl}(\lambda)$.

   (a) $\mathcal{V}_{\mathsf{poly}}$ outputs Accept at the end of the protocol.

   (b) $(\mathsf{x}, \mathsf{w}) \notin \mathcal{R}$.

3.1 Polynomial Protocols on Multiple Domains

In our protocols, $\mathcal{V}_{\mathsf{poly}}$ in fact needs to check if certain polynomial identities hold on certain subsets of values from $\mathbb{F}$, rather than in the entire $\mathbb{F}$. Hence, for subsets $H_1, \ldots, H_k \subset \mathbb{F}$, we define a *multi-ranged* (d, D, ℓ, t, h, k)*-polynomial protocol over* $H_1, \ldots, H_k$ identically to a (d, D, ℓ, t, h)-polynomial protocol, but:

1. The integer k satisfies $k \leq t$.
2. Each of the identities that the verifier asks to $\mathcal{I}$ in the last step of the protocol are only over one of $H_1, \ldots, H_k$ rather than the entire $\mathbb{F}$.

It is shown in Section 4 of [GWC19] how an H-ranged polynomial protocol can be converted to a polynomial protocol over $\mathbb{F}$, which only incurs in an additional preprocessed polynomial and polynomials sent from $\mathcal{P}_{\mathsf{poly}}$ to $\mathcal{I}$. This construction still holds true in the case of multiple ranges.

Lemma 1. *Let $\mathscr{P}$ be a multi-ranged (d, D, ℓ, t, h, k)-polynomial protocol over $H_1, \ldots, H_k$. Let $\overline{h} = \max\{|H_1|, \ldots, |H_k|\}$ and $\underline{h} = \min\{|H_1|, \ldots, |H_k|\}$. Then we can construct a $(\max\{d, \overline{h}, D - \underline{h}\}, D, \ell + k, t + k, h)$-polynomial protocol $\mathscr{P}^*$ over $\mathbb{F}$.*

Proof. We construct the protocol $\mathscr{P}^*$. The set of preprocessed polynomials in $\mathscr{P}^*$ are the same as in $\mathscr{P}$ with the addition of vanishing polynomials $Z_{H_i} := \prod_{a \in H_i}(X - a)$. $\mathscr{P}^*$ proceeds exactly[3] as $\mathscr{P}$ until the point where $\mathcal{V}_{\mathsf{poly}}$ ask about identities over $H_1, \ldots, H_k$. Suppose that the m identities the verifier asks about are $F_1(X), \ldots, F_m(X)$. Group them depending over which H_i they are asked. W.l.o.g., assume there the existence of positive integers $m_1, \ldots, m_k$ such that $m_1 + \cdots + m_k = m$ and are such that $F_1(X), \ldots, F_{m_1}(X)$ are asked over H_1, $F_{m_1+1}(X), \ldots, F_{m_1+m_2}(X)$ are asked over H_2 and so on. $\mathscr{P}^*$ now proceeds as follows:

- $\mathcal{V}_{\mathsf{poly}}$ sends uniform $a_1, \ldots, a_m \in \mathbb{F}$ to $\mathcal{P}_{\mathsf{poly}}$.
- $\mathcal{P}_{\mathsf{poly}}$ computes the polynomials:

$$T_1 := \frac{\sum_{i \in [m_1]} a_i \cdot F_i(X)}{Z_{H_1}}, \ldots, T_k := \frac{\sum_{i \in [m - m_k, m]} a_i \cdot F_i(X)}{Z_{H_k}}$$

- $\mathcal{P}_{\mathsf{poly}}$ sends $T_1, \ldots, T_k$ to $\mathcal{I}$.
- $\mathcal{V}_{\mathsf{poly}}$ queries the identities:

$$\sum_{i \in [m_1]} a_i \cdot F_i(X) \equiv T_1 \cdot Z_{H_1}, \ldots, \sum_{i \in [m - m_k, m]} a_i \cdot F_i(X) \equiv T_k \cdot Z_{H_k}.$$

The proof finish by using Claim 4.6 from [GWC19], noticing that our result holds except with probability $k/|\mathbb{F}|$. ∎

[3] Since $H_i \subset \mathbb{F}$ for all $i \in [k]$, sending messages of the form K for $K \in H_i$ for some $i \in [k]$ is equivalent to sending messages of the same form, but for $K \in \mathbb{F}$.

3.2 From Polynomial Protocols to Protocols Against Algebraic Adversaries

Regarding the compilation from polynomial protocols to protocols in the Algebraic Group Model (AGM), this compilation also holds true in our scenario with no major changes. The compilation is similar to the one in [GW21] in the sense that we can not assume the verifier only checks one polynomial identity. As we have seen in Lemma 1, the verifier in our protocols will end up asking for one identity per each range that we consider (in our protocols, it will be used two ranges).

In the following, we describe the claim about knowledge in the AGM. Let $\mathscr{P}$ be a (d, D, ℓ, t, h)-polynomial protocol. We construct a protocol $\mathscr{P}^*$ with knowledge soundness in the AGM. To this end, we must describe the extractor E for $\mathscr{P}^*$. Let $E_{\mathscr{P}}$ be the extractor of the protocol $\mathscr{P}$ and let E_{PCS} be the extractor for the knowledge soundness game as in Def. 1.

Begin by assuming that an algebraic adversary $\mathcal{A}$ is taking the role of $\mathcal{P}$ in $\mathscr{P}^*$.

1. E sends the commitments $\mathsf{cm}_1, \ldots, \mathsf{cm}_t$ and t to E_{PCS} and receives in return $f_1, \ldots, f_t \in \mathbb{F}_{<d}[X]$.

2. E plays the role of $\mathcal{I}$ in interaction with $E_{\mathscr{P}}$, sending him the received polynomials $f_1, \ldots, f_t$.

3. When $E_{\mathscr{P}}$ outputs w, E also outputs w.

Now let us define two events:

1. We think of an adversary $\mathcal{A}_{\mathscr{P}}$ participating in $\mathscr{P}$, and using the polynomials $f_1, \ldots, f_t$ as their messages to $\mathcal{I}$. We define A to be the event that one of the identities F_i held, but $(\mathsf{x}, \mathsf{w}) \notin \mathcal{R}$. By the KS of $\mathscr{P}$, $\Pr(A) = \mathsf{negl}(\lambda)$.

2. We let B be the event that for some $i \in [k], j \in [M], h_{i,j}(v_{i,j}(x)) \neq s_{i,j}$, and at the same time $\mathcal{V}$ has output Accept when open was run as a subroutine. By the KS of $\mathscr{P}$, $\Pr(B) = \mathsf{negl}(\lambda)$.

Now look at the event C that $\mathcal{V}$ outputs Accept, but E failed in the sense that $(\mathsf{x}, \mathsf{w}) \notin \mathcal{R}$. We split C into two events.

1. A or B also happened - this has $\mathsf{negl}(\lambda)$ probability.

2. C happened but not A or B. This means for some $i \in [k]$, F_i is not the zero polynomial, but $F_i(x) = 0$; which happens w.p. $\mathsf{negl}(\lambda)$.

Notice that E also has access to the messages sent directly from $\mathcal{P}_{\mathsf{poly}}$ to $\mathcal{V}_{\mathsf{poly}}$, but it does not need them to be able to obtain the "incorrect" witness ω. Hence, E can simply ignore them and continue as in the original proof.

4 Main Protocol

This section is entirely devoted to our protocol. For the description of our protocol, we follow a similar approach to the one detailed in [GW20].

In Section 4.1 we explain our the tools used in our protocol and provide an intuition on how it works. In Section 4.2 we provide a description and the soundness proof of our protocol in the trusted party $\mathcal{I}$ setting.

4.1 Grand-Product Polynomials

Fix integers m, n, let ψ be a primitive $(m+1)$-th root of unity and let ω be a primitive $(n+1)$-th root of unity. Subsequently, let $H_f = \langle\psi\rangle$ and $H_g = \langle\omega\rangle$ be the multiplicative subgroups generated by ψ and ω, respectively. Finally, let our input vectors be $f \in \mathbb{F}^m$ and $g \in \mathbb{F}^n$.

Recall that the way we want to show that some values from f are contained in g is through the grand-product polynomials Z_f and Z_g. Therefore, we have to be able to handle the "evolving" and "non-evolving" scenarios. This means that Z_f has to remain unchanged at the i-th step if the i-th value of f is not contained in g, and contribute to the grand-product otherwise. The polynomial Z_g is defined in a similar fashion.

Let γ be some element from $\mathbb{F}$.

- The computation of both Z_f and Z_g begins with $Z_f(\psi) = Z_g(\omega) = 1$.
- For $i \in [2, m+1]$, the polynomial $Z_f \in \mathbb{F}_{<m+1}[X]$ must satisfy:

$$Z_f(\psi^i) = \begin{cases} Z_f(\psi^{i-1}) \cdot (f(\psi^{i-1}) + \gamma), & \text{if there exists } j \in [n] \text{ s.t. } f_{i-1} = g_j \\ Z_f(\psi^{i-1}), & \text{otherwise} \end{cases} \tag{1}$$

- For $i \in [2, n+1]$, the polynomial $Z_g \in \mathbb{F}_{<n+1}[X]$ must satisfy:

$$Z_g(\omega^i) = \begin{cases} Z_g(\omega^{i-1}) \cdot (g(\omega^{i-1}) + \gamma), & \text{if there exists } j \in [m] \text{ s.t. } g_{i-1} = f_j \\ Z_g(\omega^{i-1}), & \text{otherwise} \end{cases} \tag{2}$$

In order to build constraints that relate the neighbor elements of those polynomials we define two selector vectors $f^{\mathsf{sel}} \in \mathbb{F}^m$, $g^{\mathsf{sel}} \in \mathbb{F}^n$. By selectors we understand polynomials that evaluate to 0 or 1 over their respective evaluation domain. The idea of the selectors is to select the elements from f and g that are supposed to be included in the permutation check. Hence, we define them under the same conditions of the polynomials Z_f and Z_g.

- For $i \in [m]$, the polynomial $f^{\mathsf{sel}} \in \mathbb{F}_{<m}[X]$ must satisfy:

$$f^{\mathsf{sel}}(\psi^i) = \begin{cases} 1, & \text{if there exists } j \in [n] \text{ s.t. } f_i = g_j \\ 0, & \text{otherwise} \end{cases}. \tag{3}$$

- For $i \in [n]$, the polynomial $g^{\mathsf{sel}} \in \mathbb{F}_{<n}[X]$ must satisfy:

$$g^{\mathsf{sel}}(\omega^i) = \begin{cases} 1, & \text{if there exists } j \in [m] \text{ s.t. } g_i = f_j \\ 0, & \text{otherwise} \end{cases}. \tag{4}$$

Now, following Eqs. (1)-(4) and in the spirit of the grand-product polynomial definitions in [GWC19] or [GW20], we can define the polynomials Z_f and Z_g in a non-recursive manner.

For $i \in [2, m+1]$ we have:

$$Z_f(\psi^i) = \prod_{1 \le j < i} \left(f^{\mathsf{sel}}(\psi^j)\left(f(\psi^j) + \gamma - 1\right) + 1\right).$$

For $i \in [2, n+1]$ we have:

$$Z_g(\omega^i) = \prod_{1 \le j < i} \left(g^{\mathsf{sel}}(\omega^j)\left(g(\omega^j) + \gamma - 1\right) + 1\right).$$

We are now ready to proceed to the description of the protocol.

4.2 Protocol Description and Soundness Proof

Based on the definitions in Section 4.1, we obtain the following protocol.

Input: $f \in \mathbb{F}_{<m}[X]$ and $g \in \mathbb{F}_{<n}[X]$.

Protocol:

1. $\mathcal{P}_{\mathsf{poly}}$ computes the polynomials $f^{\mathsf{sel}} \in \mathbb{F}_{<m}[X]$ and $g^{\mathsf{sel}} \in \mathbb{F}_{<n}[X]$ defined as in Eq. (3), (4) and sends them to $\mathcal{I}$.

2. $\mathcal{V}_{\mathsf{poly}}$ samples random $\gamma \in \mathbb{F}$ and sends it to $\mathcal{P}_{\mathsf{poly}}$.

3. $\mathcal{P}_{\mathsf{poly}}$ computes the polynomials $Z_f \in \mathbb{F}_{<m+1}[X]$ and $Z_g \in \mathbb{F}_{<n+1}[X]$ such that $Z_f(\psi) = Z_g(\omega) = 1$; and for $i \in [2, m+1]$ and $k \in [2, n+1]$:

$$Z_f(\psi^i) = \prod_{1 \le j < i} \left(f^{\mathsf{sel}}(\psi^j) \left(f(\psi^j) + \gamma - 1 \right) + 1 \right),$$
$$Z_g(\omega^k) = \prod_{1 \le j < k} \left(g^{\mathsf{sel}}(\omega^j) \left(g(\omega^j) + \gamma - 1 \right) + 1 \right).$$

   Let K denote the evaluation[4] of Z_g at ω^{n+1}, so that $K = Z_g(\omega^{n+1})$.

4. $\mathcal{P}_{\mathsf{poly}}$ sends Z_f, Z_g to $\mathcal{I}$ and K to $\mathcal{V}_{\mathsf{poly}}$.

5. $\mathcal{V}_{\mathsf{poly}}$ checks that both Z_f, Z_g are of the form described above. $\mathcal{V}_{\mathsf{poly}}$ also checks that both polynomials evaluate to the same element at ψ^{m+1} and ω^{n+1}, respectively. In other words, the same product is obtained in both in the very last step. Specifically, $\mathcal{V}_{\mathsf{poly}}$ asks to $\mathcal{I}$ for the following identities for all $x \in H_f$ and $y \in H_g$:

   (a) $L_1(x)\left(Z_f(x) - 1\right) = 0$.

   (b) $L_1(y)\left(Z_g(y) - 1\right) = 0$.

   (c) $Z_f(x \cdot \psi) = Z_f(x)\left(f^{\mathsf{sel}}(x)\left(f(x) + \gamma - 1\right) + 1\right)\left(1 - L_{m+1}(x)\right) + L_{m+1}(x)$.

   (d) $Z_g(y \cdot \omega) = Z_g(y)\left(g^{\mathsf{sel}}(y)\left(g(y) + \gamma - 1\right) + 1\right)\left(1 - L_{n+1}(y)\right) + L_{n+1}(y)$.

   (e) $L_{m+1}(x)\left(Z_f(x) - K\right) = 0$.

   (f) $L_{n+1}(y)\left(Z_g(y) - K\right) = 0$.

   outputting Accept if all checks hold.

Remark 1. To avoid notation overloading we have abused notation for the Lagrange polynomials with respect to H_f and H_g. So, for instance, the polynomial L_1 in (a) is the 1st Lagrange polynomial for H_f, but the polynomial L_1 in (b) is the 1st Lagrange polynomial for H_g.

Lemma 2. *Fix $f \in \mathbb{F}_{<m}[X]$ and $g \in \mathbb{F}_{<n}[X]$. Suppose that the sets $f = \{f_i\}_{i\in[m]}$ and $g = \{g_i\}_{i\in[n]}$ do not contain do not contain a permutation of each other. Then, for any strategy of $\mathcal{P}_{\mathsf{poly}}$, the probability of $\mathcal{V}_{\mathsf{poly}}$ outputting Accept in the above protocol is $\mathsf{negl}(\lambda)$.*

Proof. For the proof we mainly use the following simple claim. A proof of it can be found in Appendix A of [GWC19].

[4] We equivalently could have took the evaluation of Z_f at ψ^{m+1}.

Claim 2. *If the following holds with non-negligible probability over random $\gamma \in \mathbb{F}$:*

$$\prod_{i=1}^{n}(a_i + \gamma) = \prod_{i=1}^{n}(b_i + \gamma),$$

then the entries in the tuple $(a_1, \ldots, a_n)$ equal the entries in the tuple $(b_1, \ldots, b_n)$, but not necessarily in the same order.

By Claim 2, the following holds with overwhelming probability over the choice of $\gamma \in \mathbb{F}$:

$$\prod_{i\in[m]} f_i^{\mathsf{sel}} \cdot (f_i + \gamma) \neq \prod_{k\in[n]} g_k^{\mathsf{sel}} \cdot (g_k + \gamma). \tag{5}$$

Consequently, if Eq. (5) holds then the following also holds with the same probability[5]:

$$A_f := \prod_{i\in[m]} \left(f_i^{\mathsf{sel}}\left(f_i + \gamma - 1\right) + 1\right) \neq \prod_{k\in[n]} \left(g_k^{\mathsf{sel}}\left(g_k + \gamma - 1\right) + 1\right) =: A_g \tag{6}$$

From check (a) we known that $Z_f(\psi) = 1$. From check (c) we can show inductively, that for each $i \in [m]$:

$$Z_f(\psi^{i+1}) = \prod_{1\le j\le i} \left[\left(f^{\mathsf{sel}}(\psi^j)\left(f(\psi^j) + \gamma - 1\right) + 1\right)\left(1 - L_{m+1}(\psi^j)\right) + L_{m+1}(\psi^j)\right].$$

In particular, $Z_f(\psi^{m+1}) = A_f$.

Similarly, from check (b) we know that $Z_g(\omega) = 1$ and from check (d) we can show inductively that for each $i \in [n]$:

$$Z_g(\omega^{i+1}) = \prod_{1\le j\le i} \left[\left(g^{\mathsf{sel}}(\omega^j)\left(g(\omega^j) + \gamma - 1\right) + 1\right)\left(1 - L_{n+1}(\omega^j)\right) + L_{n+1}(\omega^j)\right].$$

In particular, $Z_g(\omega^{n+1}) = A_g$.

Moreover, the Lagrange polynomials L_{m+1}, L_{n+1} make checks (c) and (d) be valid over all $x \in H_f$ and $y \in H_g$, respectively. In particular, $1 = Z_f(\psi) = Z_f(\psi^{m+2})$ and $1 = Z_g(\omega) = Z_g(\omega^{n+2})$.

However, checks (e) and (f) enforces that $Z_f(\psi^{m+1}) = Z_g(\omega^{n+1})$, which leads to a contradiction with Eq. (6). ■

[5]Note that the value of these products remain unchanged if we add the values for $i = m + 1$ and $k = n + 1$, respectively.

References

[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.

[BCG+18] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. Nearly linear-time zero-knowledge proofs for correct program execution. Cryptology ePrint Archive, Report 2018/380, 2018. https://eprint.iacr.org/2018/380.

[BCTV13] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive arguments for a von neumann architecture. Cryptology ePrint Archive, Report 2013/879, 2013. https://eprint.iacr.org/2013/879.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Report 2020/081, 2020. https://eprint.iacr.org/2020/081.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, Heidelberg, May 2013.

[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.

[GW21] Ariel Gabizon and Zachary J. Williamson. fflonk: a fast-fourier inspired verifier efficient version of PlonK. Cryptology ePrint Archive, Report 2021/1167, 2021. https://eprint.iacr.org/2021/1167.

[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.

[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.

[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.