Placeholder

Yeah, that is the actual proof system name.

Alisa Cherniaeva
=nil; Foundation
a.cherniaeva@nil.foundation

Ilia Shirobokov
=nil; Foundation
i.shirobokov@nil.foundation

Mikhail Komarov
=nil; Foundation
nemo@nil.foundation

July 3, 2024

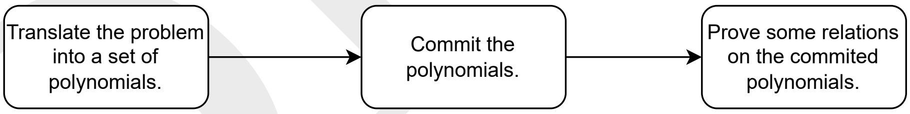
Abstract

This paper introduces Placeholder, a zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) based on $\mathcal{P}\mathfrak{lon}\mathcal{K}$*-style arithmetization. Placeholder's internal components, such as commitment schemes and types of arithmetization, are replaceable and configurable. Low-level Placeholder circuits can adapt to selected parameters, such as table size, date degree, and lookup options. These properties enable the flexible configuration of Placeholder with trade-offs between circuit parameters, trust assumptions, and efficiency of proof generation. Due to this flexibility, Placeholder can accommodate particular cases, consistently achieving efficient results.*

1 Introduction

zk-SNARK is a type of zero-knowledge proof system that allows one to prove the authenticity of a statement to a verifier without revealing any additional information beyond the statement's validity. The «succinct» and «non-interactive» aspects of zk-SNARKs refer to the fact that the proof is short and does not require any interaction between the prover and verifier beyond the initial setup.

Conceptually, general SNARK construction contains three steps.


Translate the problem into a set of polynomials.
Commit the polynomials.
Prove some relations on the commited polynomials.

Our system follows general SNARK construction and contains two «main modules»:

- **Arithmetization** defines the arithmetic representation of the proving statement. We use $\mathcal{P}\mathfrak{lon}\mathcal{K}$-based [1] representation with custom gates. The idea was introduced in TurboPLONK paper [2] and modified later in other proof systems (Halo2 [3], Plonky2 [4], Kimchi [5], etc).
- **Commitment Scheme** for polynomials obtained from the arithmetization procedure. For these purposes, we use List Polynomial Commitment scheme from [6].

A significant drawback of first zk-SNARK systems (Groth16 [7], Pinocchio [8]) in practice is the requirement to run a trusted setup for the public parameters. This procedure was necessary because every individual relation being proved with ZK-SNARKs required a different structured reference string (SRS). Recently, new schemes were constructed with an updatable and universal SRS: Marlin [9], $\mathcal{P}\mathfrak{lon}\mathcal{K}$ [10]. Because of the use of List Polynomial Commitment (LPC), the Placeholder supports a transparent setup. This type of setup makes it well suited for use in decentralized systems, a wide range of applications, including secure multi-party computations, as it allows for efficient and secure verification of computations without requiring trust in the setup of the system.

The use of an LPC scheme gives the Placeholder other benefits as well. The underlying FRI protocol is pairings-free (as opposed to Kate commitment scheme [11]). In addition, the justification of the soundness of the FRI not only does not require pairing assumptions but also works in the quantum model. However, Kate commitment scheme has a relatively small proof size and fast verification. The drawback of this scheme is the requirement for a minimal trusted setup. Placeholder allows switching between commitment schemes in order to achieve target efficiency and trust assumption.

Another drawback of the first SNARK systems is an arithmetization, which allows one type of constraint: equality of the multiplication of the sum of n variables with another sum. On the other hand, Placeholder uses PLONK-based arithmetization, which makes it possible to build custom gates. This type of arithmetization allows to roll up complex non-linear calculation in one constraint. Also, PLONK enables the using of lookups as an extension. Thus, the types of computations, which is can be efficiently expressed in Placeholder, are significantly extended. For instance, support of any logic operations can be easily added to Placeholder by using a lookup. These custom gates significantly reduce the size of the circuit required to perform a function, resulting in a faster proof construction process.

This allows you to gain performance advantages over other transparent zk-SNARKs (for example, STARK [12]).

The system focuses on the general case application. It means that we do not try to provide the best performance for some particular cases. Instead of this, we build a system that can be applied to any task with the same efficiency.

2 Preliminaries

2.1 Notation

For $n \in \mathbb{N}$ we denote by $[n]$ the set $\{0, 1, 2, \ldots, n-1\}$. We use the notation $\mathbb{F}_p^{<d}[X] = \{f \in \mathbb{F}_p[X] : \deg(f) < d\}$. The Reed-Solomon code evaluated over domain $\mathcal{D} \subseteq \mathbb{F}_p$ and rate ρ is denoted $\mathsf{RS}[\mathbb{F}_p, \mathcal{D}, \rho] = \{\varphi : \mathcal{D} \to \mathbb{F}_p \mid \exists f \in \mathbb{F}_p[X] : \ \deg(f) < \rho \cdot |\mathcal{D}| \ \wedge \ \forall x \in \mathcal{D} \ f(x) = \varphi(x)\}$.

Let $H = \langle \zeta \rangle$ with generator $\zeta \in \mathbb{F}_p^*$ and order n. We use $L_i(X) \in \mathbb{F}_p^{<n}[X]$ to denote i-th Lagrange polynomial, that satisfies $L_i(\zeta^i) = 1, \ L_i(\zeta^j) = 0$ for $i, j \in [n], \ i \neq j$.

The protocol is defined as a non-interactive one, utilizing the Fiat-Shamir heuristic. The $\mathsf{transcript}$ refers to the combination of the preprocessed common input, public input, and proof elements generated by the prover to a certain point. We use $\mathsf{transcript}.\mathit{append}()$ and $\mathsf{transcript}.\mathit{challenge}(\mathcal{D})$ to represent the update of $\mathsf{transcript}$ and generation of random challenges from domain $\mathcal{D}$ through the Fiat-Shamir method.

All vectors are denoted by bold lower case letters $\boldsymbol{q}$. Matrices are denoted by bold capital letters $\boldsymbol{\mathcal{T}}$ and we write $\boldsymbol{\mathcal{T}}_{j,i}$ for element on i-th row and j-th column. We write a matrix $\boldsymbol{\mathcal{T}}$ as $\boldsymbol{\mathcal{T}} = \left[\boldsymbol{\tau}_0^T, \ldots, \boldsymbol{\tau}_{m-1}^T\right]$ where τ_i^T is the i-th column vector of $\boldsymbol{\mathcal{T}}$. The i-th coordinate of $\boldsymbol{v}_j$ is denoted by $\boldsymbol{v}_{j,i}$ or $(\boldsymbol{v_j})_i$.

For notational simplicity, we sometimes write sequence $\{x_i\}_{i=a}^b$ as $\{x_i\}$ when range is clear from the context.

Precommitment for polynomial batch $\texttt{polynomial} = \{f_0, \ldots, f_{l-1}\}$ are denoted $\widehat{\texttt{polynomial}}$ and its commitment are denoted $\overline{\texttt{polynomial}}$.

2.2 Plonk Arithmetization

Here we describe our instantiation of PLONK with custom gates.

The computation sequence that needs to be proved is represented as *Circuit*. The Circuit is defined by fixed parameters and various sets of constraints (*Basic*, *Copy*, *Lookup*) on the *Execution trace* of the computation. Note that it is assumed that the Verifier does not know the entire Execution trace. A more formal description is as follows.

The Execution trace stores values used during computations. It is represented by a rectangular matrix $\boldsymbol{\mathcal{T}}$ (which we'll refer as *Table*) with $\mathsf{N}_{\mathsf{rows}}$ rows and $\mathsf{N}_{\mathsf{col}}$ columns:

$$\boldsymbol{\mathcal{T}} = \left[\boldsymbol{\tau}_0^T, \ldots, \boldsymbol{\tau}_{\mathsf{N}_{\mathsf{col}}-1}^T\right].$$

There are 5 types of columns $\boldsymbol{\mathcal{T}}$.

- *Witness* columns contain witness input and intermediate calculations.
  Witness columns differ between proof instances (because they depend on input). They are not known

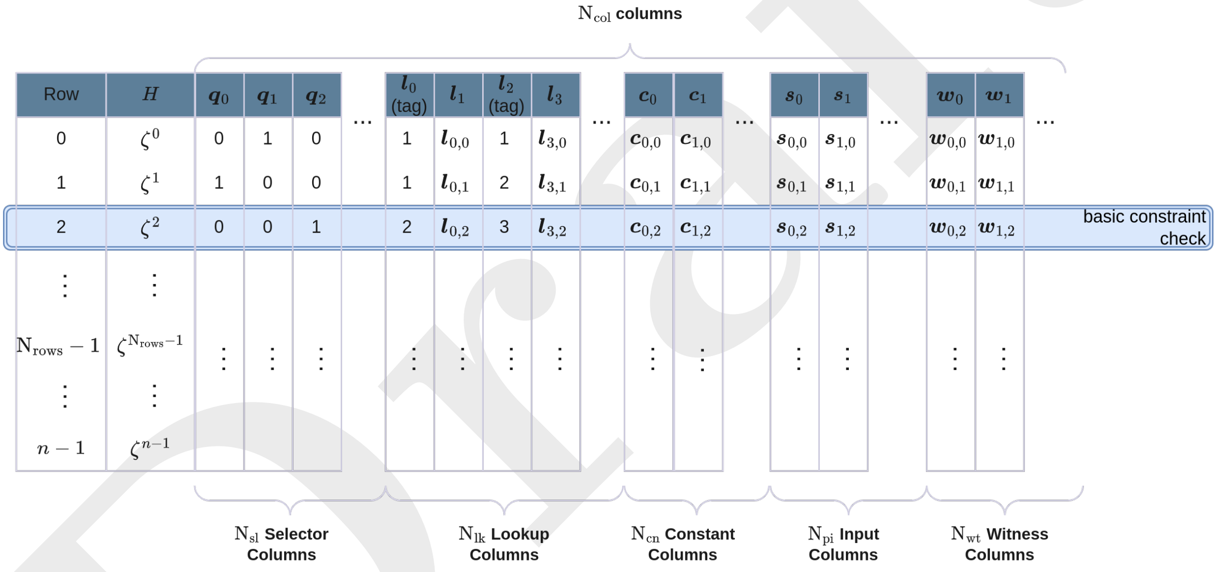
to the verifier. We denote witness columns by $\boldsymbol{w} = (\boldsymbol{w}_0, \dots, \boldsymbol{w}_{\mathsf{N_{wt}}-1})$ where $\mathsf{N_{wt}} \in \mathbb{N},\ \boldsymbol{w}_i \in \mathbb{F}^{\mathsf{N_{rows}}}$ for $i = 0, \dots, \mathsf{N_{wt}} - 1$.

- *Public* columns contain public input for computation.
  Public Columns differ between proof instances (because they depend on input). They are known to the verifier. We denote public columns by $\boldsymbol{s} = (\boldsymbol{s}_0, \dots, \boldsymbol{s}_{\mathsf{N_{pi}}-1})$ where $\mathsf{N_{pi}} \in \mathbb{N},\ \boldsymbol{s}_i \in \mathbb{F}^{\mathsf{N_{rows}}}$ for $i = 0, \dots, \mathsf{N_{pi}} - 1$.
- *Constant* columns contain circuit-depended data.
  Constant columns do not differ between proof instances. They are known to the verifier. We denote constant columns by $\boldsymbol{c} = (\boldsymbol{c}_0, \dots, \boldsymbol{c}_{\mathsf{N_{cn}}-1})$ where $\mathsf{N_{cn}} \in \mathbb{N},\ \boldsymbol{c}_i \in \mathbb{F}^{\mathsf{N_{rows}}}$ for $i = 0, \dots, \mathsf{N_{cn}} - 1$.
- *Selector* columns define which rows of the Table the basic constraint is applied.
  Selectors' values can be only ones or zeroes. We denote selector columns by $\boldsymbol{q} = (\boldsymbol{q}_0, \dots, \boldsymbol{q}_{\mathsf{N_{sl}}-1})$ where $\mathsf{N_{sl}} \in \mathbb{N},\ \boldsymbol{q}_i \in \{0,1\}^{\mathsf{N_{rows}}}$ for $i = 0, \dots, \mathsf{N_{sl}} - 1$.
- *Lookup* columns define tables for membership testing. We denote lookup columns by $\boldsymbol{l} = (\boldsymbol{l}_0, \dots, \boldsymbol{l}_{\mathsf{N_{lk}}-1})$ where $\mathsf{N_{lk}} \in \mathbb{N},\ \boldsymbol{l}_i \in \mathbb{F}_p{}^{\mathsf{N_{rows}}}$ for $i = 0, \dots, \mathsf{N_{lk}} - 1$.

Assume without loss of generality that

$$\boldsymbol{\mathcal{T}} = \left[\boldsymbol{q}_0^T, \dots, \boldsymbol{q}_{\mathsf{N_{sl}}-1}^T, \boldsymbol{l}_0^T, \dots, \boldsymbol{l}_{\mathsf{N_{lk}}-1}^T, \boldsymbol{c}_0^T, \dots, \boldsymbol{c}_{\mathsf{N_{cn}}-1}^T, \boldsymbol{s}_0^T, \dots, \boldsymbol{s}_{\mathsf{N_{pi}}-1}^T, \boldsymbol{w}_0^T, \dots, \boldsymbol{w}_{\mathsf{N_{wt}}-1}^T\right],$$

where $\mathsf{N_{sl}} + \mathsf{N_{lk}} + \mathsf{N_{cn}} + \mathsf{N_{pi}} + \mathsf{N_{wt}} = \mathsf{N_{col}}$ (see figure 1).



$\mathrm{N_{col}}$ columns
$\mathrm{N_{sl}}$ Selector Columns
$\mathrm{N_{lk}}$ Lookup Columns
$\mathrm{N_{cn}}$ Constant Columns
$\mathrm{N_{pi}}$ Input Columns
$\mathrm{N_{wt}}$ Witness Columns

Row	H	$\boldsymbol{q}_0$	$\boldsymbol{q}_1$	$\boldsymbol{q}_2$	…	$\boldsymbol{l}_0$ (tag)	$\boldsymbol{l}_1$	$\boldsymbol{l}_2$ (tag)	$\boldsymbol{l}_3$	…	$\boldsymbol{c}_0$	$\boldsymbol{c}_1$	…	$\boldsymbol{s}_0$	$\boldsymbol{s}_1$	…	$\boldsymbol{w}_0$	$\boldsymbol{w}_1$	…
0	ζ^0	0	1	0		1	$\boldsymbol{l}_{0,0}$	1	$\boldsymbol{l}_{3,0}$		$\boldsymbol{c}_{0,0}$	$\boldsymbol{c}_{1,0}$		$\boldsymbol{s}_{0,0}$	$\boldsymbol{s}_{1,0}$		$\boldsymbol{w}_{0,0}$	$\boldsymbol{w}_{1,0}$	
1	ζ^1	1	0	0		1	$\boldsymbol{l}_{0,1}$	2	$\boldsymbol{l}_{3,1}$		$\boldsymbol{c}_{0,1}$	$\boldsymbol{c}_{1,1}$		$\boldsymbol{s}_{0,1}$	$\boldsymbol{s}_{1,1}$		$\boldsymbol{w}_{0,1}$	$\boldsymbol{w}_{1,1}$	
2	ζ^2	0	0	1		2	$\boldsymbol{l}_{0,2}$	3	$\boldsymbol{l}_{3,2}$		$\boldsymbol{c}_{0,2}$	$\boldsymbol{c}_{1,2}$		$\boldsymbol{s}_{0,2}$	$\boldsymbol{s}_{1,2}$		$\boldsymbol{w}_{0,2}$	$\boldsymbol{w}_{1,2}$	basic constraint check
⋮	⋮																		
$\mathrm{N_{rows}} - 1$	$\zeta^{\mathrm{N_{rows}}-1}$	⋮	⋮	⋮		⋮	⋮	⋮	⋮		⋮	⋮		⋮	⋮		⋮	⋮	
⋮	⋮																		
$n-1$	ζ^{n-1}																		

Figure 1: *Structure of execution trace table* $\boldsymbol{\mathcal{T}}$

Sets of constraints define relationships between values of $\boldsymbol{\mathcal{T}}$.

- *Basic* constraints are expressions for table values of a certain row (and possibly several adjacent ones).
  Let $\mathbf{o}$ – set of offsets for the row indices involved in the constraint. Usually, $\mathbf{o} = \{-1, 0, 1\}$. The j-th constraint $(0 \leq j < \mathsf{C_{bs}})$ is given in the form of a multivariate polynomial $\mathcal{C}'_j$ of total degree $\mathsf{C_{dg}}$ over the table values:

$$\mathcal{C}'_j(\{\boldsymbol{w}_{0,i+o'}\}_{o' \in \mathbf{o}}, \dots, \{\boldsymbol{w}_{\mathsf{N_{wt}}-1,i+o'}\}_{o' \in \mathbf{o}}) = 0,\ \text{where } i - \text{number of row}, i \in [\mathsf{N_{rows}}] \tag{1}$$

An example of a basic constraint could be the following:

$$\boldsymbol{w}_{j,i} \cdot \boldsymbol{w}_{j+1,i} + \boldsymbol{w}_{j+1,i+1} - 1 = 0. \tag{2}$$

Selectors are used to include/exclude a Basic Constraint check to/from the Row. Selectors are included as a part of the assertion to do this. A set of Basic Constraints used with the same Selector is called *Gate*. A gate may contain one or more constraints. Each Row has to satisfy all Gates of the Circuit.

- *Copy* constraints defines equality assertions between Cells. Such constraints have following form $\boldsymbol{\mathcal{T}}_{i,j} = \boldsymbol{\mathcal{T}}_{i',j'}$.
- *Lookup* constraints assert that the chosen tuples of cells of the Table are equal to some rows in lookup table L. Note that Lookup Constraint does not define the precise place of the tuple in the Lookup Table. It is the main difference between Lookup and Copy constraints. Such constraints have following form $(\boldsymbol{\mathcal{T}}_{i,j}, \boldsymbol{\mathcal{T}}_{i+1,j}, \boldsymbol{\mathcal{T}}_{i+2,j+1}) \in L$.

In order to arithmetize basic constraints we need to define how gates univariate polynomials are constructed from constraints. Remind that gates contains a set of constraints with the same selector.

In practice, the table with $\mathsf{N}_{\mathsf{rows}}$ rows is padded to get the number of rows $n = 2^d$ for some $d \in \mathbb{N}$.

Let $H \subseteq \mathbb{F}_p^*$ – cyclic group with generator ζ: $H = \{\zeta^0, ..., \zeta^{n-1}\}$. The bijection $[n] \to H$ make it possible to consider $\boldsymbol{\tau}_i$ as values of some polynomial on evaluation domain H. For each table column we define interpolant polynomial from $\mathbb{F}_p^{<n}[X]$ using Lagrange basis. For j-th witness column $j \in [\mathsf{N}_{\mathsf{wt}}]$ we have

$$w_j(x) = \sum_{i=0}^{n-1} \boldsymbol{w}_{j,i} \cdot L_i(x).$$

Selector $\{s_i(x)\}$, constant $\{c_i(x)\}$, lookup-tables $\{l_i(x)\}$ and public input $\{s_i(x)\}$ polynomials are given analogously.

Each constraint $\mathcal{C}'$ (see equation 1) may be written as polynomial $\mathcal{C}_j(X) = \mathcal{C}'_j(\{w_j(\zeta^{o'} \cdot X)\}_{o' \in \mathbf{o},\ j \in [\mathsf{N}_{\mathsf{wt}}]})$, $X \in H$. Then example 2 has the following form $w_j(\zeta^i) \cdot w_{j+1}(\zeta^i) + w_{j+1}(\zeta^{i+1}) - 1 = 0$.

For constraint polynomials $\mathcal{C}_0, \ldots, \mathcal{C}_{k-1}$ used in i-th gate and corresponding random challenge θ_i, we can represent i-th gate as: $\mathcal{G}_i(X) = q_i(X) \cdot (\theta_i^0 \mathcal{C}_0(X) + \cdots + \theta_i^{k-1} \mathcal{C}_{k-1}(X))$.

For details on Copy and Lookup constraints arithmetization, we refer the reader to sections 3 and 4.

3 Permutation Argument

Here we describe the arithmetization of the copy constraints and permutation argument, which is used as part of the protocol in Section 7. Permutation argument description is based on [10].

3.1 Permutation Argument Details

Copy constraints affect only part of the program execution trace. For simplicity of notation, we will assume that the copy constraints are given for the matrix $\boldsymbol{\mathcal{T}}'$, composed of the columns $j_1, \ldots, j_m$ of the original matrix $\boldsymbol{\mathcal{T}}$:

$$\boldsymbol{\mathcal{T}}' = [\boldsymbol{\tau}_{j_1}, \ldots, \boldsymbol{\tau}_{j_m}].$$

Let $\boldsymbol{\mathcal{T}}'$ be the table representation of the part of the circuit's trace with n rows and m columns. Denote by $\boldsymbol{\mathcal{T}}'_{j,i}$ value of the cell $(i; j)$, $i \in [n], j \in [m]$ during circuit's trace computation. Copy constraints are represented by equalities of the following form

$$\boldsymbol{\mathcal{T}}'_{j,i} = \boldsymbol{\mathcal{T}}'_{j',i'},$$

and break table elements into disjoint cycles. Let the permutation $\widehat{\sigma}$ define a cycle structure on the matrix $\boldsymbol{\mathcal{T}}'$:

$$\widehat{\sigma} : [m] \times [n] \to [m] \times [n].$$

The permutation $\widehat{\sigma}$ can be obtained using the algorithm described in Section 3.2.

Let as above $H = \{\zeta^0, ..., \zeta^{n-1}\}$ be a cyclic subgroup of $\mathbb{F}_p^*$, $\{L_i(x)\}$ – Lagrange basis for H. Let $\gamma \in \mathbb{F}_p$ be a $\mathcal{B}$-th root of unity, where $\mathcal{B} \cdot 2^{d^*} + 1 = p$, M is odd. Let $\gamma^i \cdot H$ for $i = 0, \ldots, m-1$ are distinct cosets of H in $\mathbb{F}_p^*$:

$$(\gamma^i \cdot H) \cap (\gamma^j \cdot H) = \varnothing,\ i \neq j.$$

Let $H' = H \cup (\gamma \cdot H) \cup \cdots \cup (\gamma^{m-1} \cdot H)$ and permutations $e, \sigma : [m] \times [n] \to H'$:

$$e(j,i) = \gamma^j \zeta^i, \quad \sigma(j,i) = e(\widehat{\sigma}(j,i)).$$

Now we can arithmetize copy constraints by identity and permutation polynomials

$$S_j^e(X) = \sum_{i=0}^{n-1} e(j,i) L_i(X),\ 0 \le j < m,$$

$$S_j^\sigma(X) = \sum_{i=0}^{n-1} \sigma(j,i) L_i(X),\ 0 \le j < m,$$

It is easy to check that $S_j^e(\zeta^i) = \gamma^j \zeta^i$ and for each $\widehat{\sigma}(j,i) = (j', i')$ we have $S_j^\sigma(\zeta^i) = \gamma^{j'} \zeta^{i'}$.

Let columns of $\boldsymbol{\mathcal{T}}'$ are represented by polynomials $f_j \in \mathbb{F}_p[X]$ in Lagrange basis. For pair of random $\theta_1, \theta_2 \in \mathbb{F}_p$ we define polynomials

$$f_j^e(X) = f_j(X) + \theta_1 \cdot S_j^e(X) + \theta_2,\ f_j^\sigma(X) = f_j(X) + \theta_1 \cdot S_j^\sigma(X) + \theta_2.$$

Let $V^\sigma \in \mathbb{F}_p[X]$ such that $V^\sigma(\zeta^0) = 1$ and for $t = 1, \ldots, n-1$

$$V^\sigma(\zeta^t) = V^\sigma(\zeta^{t-1}) \cdot \prod_{j=0}^{m-1} \frac{f_j^e(\zeta^{t-1})}{f_j^\sigma(\zeta^{t-1})} = \prod_{i=0}^{t-1} \prod_{j=0}^{m-1} \frac{f_j^e(\zeta^i)}{f_j^\sigma(\zeta^i)} = \prod_{i=0}^{t-1} \prod_{j=0}^{m-1} \frac{f_j(\zeta^i) + \theta_1 \cdot S_j^e(\zeta^i) + \theta_2}{f_j(\zeta^i) + \theta_1 \cdot S_j^\sigma(\zeta^i) + \theta_2}.$$

The constraint system will be considered fulfilled if the following conditions are met for any $a \in H$

$$\begin{cases} L_0(a)(1 - V_\sigma(a)) = 0 \\ V^\sigma(a) \prod_{j=0}^{m-1} f_j^e(a) = V^\sigma(a \cdot \zeta) \prod_{j=0}^{m-1} f_j^\sigma(a). \end{cases}$$

The constructed polynomials are used in the main protocol described in Section 7. In practice, the table with $\mathsf{N}_{\mathsf{rows}}$ rows is padded to get the number of rows $n = 2^d$ for some $d \in \mathbb{N}$. Therefore, the specified conditions may not be valid for the last rows. To solve this problem, we add to equations special selectors polynomials. Let columns-selectors:

$$(\boldsymbol{q}^{\text{last}})_i = 1,\ \text{if } i = \mathsf{N}_{\mathsf{rows}},\ \text{and } 0,\ \text{otherwise },$$

$$(\boldsymbol{q}^{\text{pad}})_i = 1,\ \text{if } i > \mathsf{N}_{\mathsf{rows}},\ \text{and } 0,\ \text{otherwise }.$$

Then $q^{\text{last}}(x),\ q^{\text{pad}}(x)$ – corresponding polynomials in Lagrange basis.

Thus, the calculation of the permutation argument and and its verification is given by algorithms 1, 2.

Algorithm 1 PermArgument

```
    Input: $f_0, \ldots, f_{m-1} : f_i \in \mathbb{F}_p^{<n}[X],\ \{S_j^e(X)\},\ \{S_j^\sigma(X)\},\ q^{\text{last}}(x),\ q^{\text{pad}}(x),\ \mathsf{transcript}$
    Output: $F_0, F_1, F_2, V^\sigma$
$\theta_1, \theta_2 = \mathsf{transcript}.\mathit{challenge}(\mathbb{F}_p^2)$
Calculate $f^\sigma(X),\ f^e(X)$:
        $f^e(X) = \prod_{j=0}^{m-1} f_j^e(X) = \prod_{j=0}^{m-1} (f_j(X) + \theta_1 \cdot S_j^e(X) + \theta_2)$
        $f^\sigma(X) = \prod_{j=0}^{m-1} f_j^\sigma(X) = \prod_{j=0}^{m-1} (f_j(X) + \theta_1 \cdot S_j^\sigma(X) + \theta_2)$
Calculate $V^\sigma(x)$ for $x = \zeta^0, \ldots, \zeta^n$:
        $V^\sigma(\zeta) = V^\sigma(\zeta^n) = 1$
        $V^\sigma(\zeta^t) = V^\sigma(\zeta^{t-1}) \cdot \frac{f^e(\zeta^i)}{f^\sigma(\zeta^i)}$ for $0 < t < n$
Calculate permutation-related numerators of the quotient polynomial:
        $F_0(X) = L_0(X)(1 - V^\sigma(X))$
        $F_1(X) = (1 - (q^{\text{last}}(X) + q^{\text{pad}}(X))) \cdot (V^\sigma(\zeta X) \cdot f^\sigma(X) - V^\sigma(X) \cdot f^e(X))$
        $F_2(X) = q^{\text{last}}(X) \cdot (V^\sigma(X)^2 - V^\sigma(X))$
return $F_0, F_1, F_2, \overline{V^\sigma}$
```

```
Algorithm 2 PermArgumentVerify
  Input: $V^\sigma(y), V^\sigma(\zeta \cdot y), \{f_i(y)\}, \{S_i^e(y)\}, \{S_i^\sigma(y)\}, q^{\text{pad}}(y), q^{\text{last}}(y), L_0(y),$ transcript
  Output: $F_0(y), F_1(y), F_2(y)$
  $\theta_1, \theta_2 =$ transcript.$challenge(\mathbb{F}_p^2)$
  Denote $f^\sigma(y),\ f^e(y)$:
      $f^e(y) = \prod_{j=0}^{m-1} (f_j(y) + \theta_1 \cdot S_j^e(y) + \theta_2)$
      $f^\sigma(y) = \prod_{j=0}^{m-1} (f_j(y) + \theta_1 \cdot S_j^\sigma(y) + \theta_2)$
  Calculate:
      $F_0(y) = L_0(y)(1 - V^\sigma(y))$
      $F_1(y) = (1 - (q^{\text{last}}(y) + q^{\text{pad}}(y))) \cdot (V^\sigma(\zeta y) \cdot f^\sigma(y) - V^\sigma(y) \cdot f^e(y))$
      $F_2(y) = q^{\text{last}}(y) \cdot (V^\sigma(y)^2 - V^\sigma(y))$
  return $F_0(y), F_1(y), F_2(y)$
```

3.2 Permutation Construction Algorithm

We use the same algorithm as Halo [13] .

We can split all copy constraints into a set of cycles such that cells in the same cycles are supposed to have the same circuit's trace value. For each set of equal cells $\boldsymbol{\mathcal{T}}'_{a_1,b_1} = \cdots = \boldsymbol{\mathcal{T}}'_{a_k,b_k}$ define a cycle $\boldsymbol{\iota} = (\boldsymbol{\iota}_0 = \boldsymbol{\mathcal{T}}_{a_1,b_1}, \dots, \boldsymbol{\iota}_{k-1} = \boldsymbol{\mathcal{T}}_{a_k,b_k})$. The circuit's permutation is defined as a composition of these cycles.

Further in this paragraph we use single-letter notations x for table element for simplicity. However, it is $x = (j, i)$, where j is the cell's column and i is the cell's row.

The copy state is represented by three maps:

- **aux** that returns the distinguished element of each cycle:
  if x is element of cycle $\boldsymbol{\iota}$
  $$\mathbf{aux}(x) = \boldsymbol{\iota}_0;$$
  If x, y belong to the same cycle, then $\mathbf{aux}(x) = \mathbf{aux}(y)$.
- **next** (permutation itself) that return next element in cycle
  if x is j-th element of cycle $\boldsymbol{\iota}$:
  $$\mathbf{next}(x) = \boldsymbol{\iota}_{(j+1) \mod |\boldsymbol{\iota}|};$$
- **size** that returns the size of each cycle:
  if x is element of cycle $\boldsymbol{\iota}$
  $$\mathbf{size}(\mathbf{aux}(x))) = |\boldsymbol{\iota}|;$$

The process of construction the permutation is described by algorithms 3, 4.

```
Algorithm 3 Copy State Initialization
  Input: –
  Output: initialized copy state
  for all $x$ do:                          ▷ each $x$ is one-element cycle
      next$(x) = x$
      aux$(x) = x$
      size$(x) = 1$
  end for
```

Algorithm 4 Add Copy Constraint

```
    Input:  copy constraint «$x = y$»
    Output:  updated copy state
if if $\mathbf{aux}(x) = \mathbf{aux}(y)$ then
    return                                        ▷ don't do anything if $x, y$ belong to the same cycle
end if
if $\mathbf{size}(\mathbf{aux}(\mathrm{x})) < \mathbf{size}(\mathbf{aux}(\mathrm{y}))$ then
    $\mathbf{swap}(\mathrm{x,y})$                 ▷ Let $x$ be an input with a larger cycle and $y$ the other one.
end if
$\mathbf{size}(\mathbf{aux}(x)) = \mathbf{size}(\mathbf{aux}(x)) + \mathbf{size}(\mathbf{aux}(y))$    ▷ the right cycle will be merged into the left cycle
$z = \mathbf{aux}(y)$
repeat                                            ▷ set all pointers from $y$ cycle to $x$ cycle
    $\mathbf{aux}(z) = \mathbf{aux}(x)$
    $z = \mathbf{next}(z)$
until $\mathrm{z} = \mathbf{aux}(\mathrm{y})$
$tmp = \mathbf{next}(x)$                          ▷ actually merge cycles in next
$\mathbf{next}(x) = \mathbf{next}(y)$
$\mathbf{next}(y) = tmp$
```

4 Lookup Argument

Here we describe the transformation from lookup constraints to lookup argument, which is used as part of the protocol in Section 7. We use the lookup argument proposed in Halo [14].

4.1 Arithmetization Details

Lookup constraints require that the element values of some columns of table $\boldsymbol{\mathcal{T}}$ belong to a predefined set represented by other columns in the table. Let us describe these constraints more formally. Let $j_0, \ldots, j_{m-1}$ and $j'_0, \ldots, j'_{m-1}$ – indicies of columns of $\boldsymbol{\mathcal{T}}$. The lookup constraint is satisfied if $\forall i \in [n] \;\; \exists i' \in [n]:$

$$\boldsymbol{\mathcal{T}}_{j_t, i} = \boldsymbol{\mathcal{T}}_{j'_t, i'} \text{ for } t = 0, \ldots, m-1.$$

Let denote these columns as follows.

$$\begin{aligned} \boldsymbol{A} &= [\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{m-1}] = \left[\boldsymbol{\tau}_{\boldsymbol{j_0}}, \ldots, \boldsymbol{\tau}_{\boldsymbol{j_{m-1}}}\right], \\ \boldsymbol{L} &= [\boldsymbol{l}_0, \ldots, \boldsymbol{l}_{m-1}] = \left[\boldsymbol{\tau}_{\boldsymbol{j'_0}}, \ldots, \boldsymbol{\tau}_{\boldsymbol{j'_{m-1}}}\right]. \end{aligned}$$

We call $\boldsymbol{A}$ – *Lookup Input*, and $\boldsymbol{L}$ – *Lookup Table*. Note, that $N_{\mathbf{rows}}$ is equal to the number of usable rows in $\boldsymbol{\mathcal{T}}$. We refer to $\{\boldsymbol{a}_i\}$ as input columns and to $\{\boldsymbol{l}_i\}$ as lookup columns. Both $\boldsymbol{A}$ and $\boldsymbol{L}$ can contain duplicate. If it is necessary to extend one of the sets, we extend $\boldsymbol{L}$ with duplicates and $\boldsymbol{A}$ with dummy values known to be in $\boldsymbol{L}$. The table $\boldsymbol{L}$ has not to be fixed. Any columns from $\boldsymbol{L}$ can be witness columns.

Let $\theta \in \mathbb{F}$ is the verifier's challenge. We compress the columns $\{\boldsymbol{a}_i\}$, $\{\boldsymbol{l}_i\}$ into *Compressed Lookup Input* column $\boldsymbol{a}$ and *Compressed Lookup* column $\boldsymbol{l}$ as follow:

$$\begin{aligned} \boldsymbol{a} &= \theta^{m-1}\boldsymbol{a}_0 + \cdots + \theta\boldsymbol{a}_{m-2} + \boldsymbol{a}_{m-1} \\ \boldsymbol{l} &= \theta^{m-1}\boldsymbol{l}_0 + \cdots + \theta\boldsymbol{l}_{m-2} + \boldsymbol{l}_{m-1}. \end{aligned}$$

Now we need to introduce notations for general lookup representation.

There are two parts of lookup argument similar to the original PLONK argument: permutation and assertion check. Firstly, the prover permutes $\boldsymbol{a}$, $\boldsymbol{l}$ in a such way that verification of inclusion lookup queries into $\boldsymbol{l}$ is relatively simple task. After that, the prover provides a permutation argument for the permuted columns. Finally, they prove that the values from the permuted $\boldsymbol{a}$ is subset of the values from the permuted $\boldsymbol{l}$.

4.2 Permutation

Firstly, the prover calculates two additional columns $\boldsymbol{a}^{\text{perm}}$ and $\boldsymbol{l}^{\text{perm}}$ that are permutations of $\boldsymbol{a}$ and $\boldsymbol{l}$ respectively.

The permutations for the new columns are defined by the following rules.

- All the cells of column $\boldsymbol{a}^{\text{perm}}$ are arranged so that like-valued cells are vertically adjacent to each other. The order of these like-valued groups is not matter.
- The first row in a sequence of like values in $\boldsymbol{a}^{\text{perm}}$ is the row that has the corresponding value in $\boldsymbol{l}^{\text{perm}}$. The order of the other values in $\boldsymbol{l}^{\text{perm}}$ can be arbitrary.

Similarly to Section 3, we use a grand product argument [1] to prove that $\boldsymbol{a}^{\text{perm}}$, $\boldsymbol{l}^{\text{perm}}$ are permutations of $\boldsymbol{a}$, $\boldsymbol{l}$.

4.3 Assertion Check

The permuted columns are constructed in a such way that we can assert that all elements from a^{perm} are presented in l^{perm} with the following rules:

1. $(a^{\text{perm}}(X) - l^{\text{perm}}(X)) \cdot (a^{\text{perm}}(X) - a^{\text{perm}}(\zeta^{-1}X)) = 0$
   to ensure that either $(\boldsymbol{a}^{\text{perm}})_j = (\boldsymbol{l}^{\text{perm}})_j$ or $(\boldsymbol{a}^{\text{perm}})_j = (\boldsymbol{a}^{\text{perm}})_{j-1}$;
2. $L_0(X) \cdot (a^{\text{perm}}(X) - l^{\text{perm}}(X)) = 0$
   we need it because $(a^{\text{perm}}(X) - a^{\text{perm}}(\zeta^{-1}X))$ is not a valid check on the first row.

In order to achieve zero-knowledge we use polynomials $q^{\text{last}}(X)$, $q^{\text{pad}}(X)$ as before. So we have the following constraints:

1. $(1 - (q^{\text{last}}(X) + q^{\text{pad}}(X))) \cdot (V_L(\zeta X) \cdot (a^{\text{perm}}(X) + \theta_1) \cdot (l^{\text{perm}}(X) + \theta_2) - V_L(X) \cdot (a^{\texttt{compr}}(X) + \theta_1) \cdot (l^{\texttt{compr}}(X) + \theta_2)) = 0$,
2. $(1 - (q^{\text{last}}(X) + q^{\text{pad}}(X))) \cdot (a^{\text{perm}}(X) - l^{\text{perm}}(X)) \cdot (a^{\text{perm}}(X) - a^{\text{perm}}(\zeta^{-1}X)) = 0$,
3. $q^{\text{last}}(X) \cdot (V_L(X)^2 - V_L(X)) = 0$.

4.4 Generalization

Each lookup input's cell can be any polynomial expression and use the relative references in the lookup constraint. It influences on the way how the column a_{perm} is calculated.

Let k-th lookup constraint $k = 0, \dots, \mathsf{C}_{\mathsf{lk}} - 1$ given by $\nu(k)$ column indices $j_0^k, \dots, j_{\nu(k)-1}^k$, corresponding offsets $d_0^k, \dots, d_{\nu(k)-1}^k \in \mathbf{o}$ and lookup table columns $u_0^k, \dots, u_{\nu(k)-1}^k$ determine the following equalities:

$$\exists i' \in [n] : \ \forall i \in [\nu(k)] : \ \boldsymbol{a}_{j_i^k, i+d_i^k} = \boldsymbol{l}_{u_i^k, i'}$$

To combine multiple lookup constraints into one argument, we add one more random challenge. Denote a random challenges by θ.

Thus, the lookup expression would be:

$$\texttt{lookup\_gate}_j(X) = q_{lj} \cdot \sum_{k=0}^{\mathsf{C}_{\mathsf{lk}}-1} \sum_{i=0}^{\nu(k)-1} \theta^{\mathcal{N}_k + i} a_{j_i^k}(\zeta^{d_j^k} \cdot X), \quad \text{where } \mathcal{N}_k = \sum_{i=0}^{k-1} \nu(i).$$

The compressed lookup input:

$$a^{\texttt{compr}}(\zeta^j) = \textstyle\sum_{0 \le i < \mathsf{C}_{\mathsf{lk}}} \texttt{lookup\_gate}_i(\zeta^j)$$

The compressed lookup table is computed similarly to compressed lookup input:

$$\begin{aligned}\texttt{table\_value}_j(X) &= \sum_{k=0}^{\mathsf{C_{lk}}-1} \sum_{i=0}^{\nu(k)-1} \theta^{\mathcal{N}_k+i} l_{j_i^k}(X), \quad \text{where} \;\; \mathcal{N}_k = \sum_{i=0}^{k-1} \nu(i) \\ l^{\texttt{compr}}(X) &= \sum_{0 \le i < \mathsf{N_{lk}}} \texttt{table\_value}_i(X)\end{aligned}$$

4.5 Small Tables

Note that we can arrange multiple tables in the same columns using tag column.

For instance, let $\boldsymbol{L}^{(1)} = \left[\boldsymbol{l}_0^{(1)}, \boldsymbol{l}_1^{(1)}\right]$, $\boldsymbol{L}^{(2)} = \left[\boldsymbol{l}_0^{(2)}, \boldsymbol{l}_1^{(2)}\right]$ be two lookup tables with 4 rows. Two lookup expression corresponds to $\boldsymbol{L}^{(1)}, \boldsymbol{L}^{(2)}$. These tables can be located in the separate way:

$\boldsymbol{q}_{l_1}$	$\boldsymbol{l}_0^{(1)}$	$\boldsymbol{l}_1^{(1)}$	$\boldsymbol{q}_{l_2}$	$\boldsymbol{l}_0^{(2)}$	$\boldsymbol{l}_1^{(2)}$
...	0	1	...	4	5
...	1	1	...	5	6
...	2	1	...	6	7
...	3	0	...	7	8

However, $N_{\texttt{rows}} \gg 4$ in a typical case. This means that the prover has to complete these columns to $N_{\texttt{rows}}$ and commit all of them. Instead of this, we can arrange the tables in the following way:

$\boldsymbol{q}_{l_1}$	$\boldsymbol{q}_{l_2}$	tag	$\boldsymbol{l}_0$	$\boldsymbol{l}_1$
...	...	1	0	1
...	...	1	1	1
...	...	1	2	1
...	...	1	3	0
...	...	2	4	5
...	...	2	5	6
...	...	2	6	7
...	...	2	7	8

It allows saving up to $(N_{\texttt{con\_tables}} - 1) \cdot \texttt{max\_columns} - 1$ columns, where `max_columns` is the maximum number of columns in all $N_{\texttt{con\_tables}}$ concatenated tables.

4.6 Lookup Arguments

Algorithm 5 LookupArgument

```
Input: $a_0, \dots, a_{m-1}$, $\mathsf{transcript}$
Output: $F_3, F_4, F_5, F_6, F_7, V_L, \mathsf{lookup}^{\mathsf{perm}}, \widehat{\mathsf{lookup}^{\mathsf{perm}}}$
$\theta = \mathsf{transcript}.\mathit{challenge}(\mathbb{F})$
For $j = 0, \dots, \mathsf{N}_{\mathsf{lk}} - 1$ (see Section 4.4 for details):
    $\texttt{lookup\_gate}_j(X) = q_{lj} \cdot \sum_{k=0}^{\mathsf{C}_{\mathsf{lk}}-1} \sum_{i=0}^{\nu(k)-1} \theta^{\mathcal{N}_k+i} a_{j_i^k}(\zeta^{d_j^k} \cdot X), \quad \text{where } \mathcal{N}_k = \sum_{i=0}^{k-1} \nu(i)$
    $\texttt{table\_value}_j(X) = \sum_{k=0}^{\mathsf{C}_{\mathsf{lk}}-1} \sum_{i=0}^{\nu(k)-1} \theta^{\mathcal{N}_k+i} l_{j_i^k}(X),$
Construct the input lookup compression and table compression values for $1 \le j \le \mathsf{N}_{\mathsf{rows}}$:
    $a^{\texttt{compr}}(\zeta^j) = \sum_{0 \le i < \mathsf{N}_{\mathsf{lk}}} \texttt{lookup\_gate}_i(\zeta^j)$
    $l^{\texttt{compr}}(\zeta^j) = \sum_{0 \le i < \mathsf{N}_{\mathsf{lk}}} \texttt{table\_value}_i(\zeta^j)$
Produce the permutation polynomials $a^{\text{perm}}(X)$ and $l^{\text{perm}}(X)$ according to Section 4.2.
$\mathsf{lookup}^{\mathsf{perm}} = \{a^{\text{perm}}, l^{\text{perm}}\}$,
$\widehat{\mathsf{lookup}^{\mathsf{perm}}} = \mathsf{MT.Precommit}(\mathsf{lookup}^{\mathsf{perm}})$,
$\mathsf{transcript}.\mathit{append}(\mathsf{MT.Commit}(\widehat{\mathsf{lookup}^{\mathsf{perm}}}))$
$\beta_2, \gamma_2 = \mathsf{transcript}.\mathit{challenge}(\mathbb{F})$
Compute $V_L(X)$ such that:
    $V_L(1) = V_L(\omega^{\mathsf{N}_{\mathsf{rows}}}) = 1$
    $V_L(\omega^j) = \prod_{i=0}^{j-1} \frac{(a^{\texttt{compr}}(\omega^i) + \beta_2)(l^{\texttt{compr}}(\omega^i) + \gamma_2)}{(a^{\text{perm}}(\omega^i) + \beta_2)(l^{\text{perm}}(\omega^i) + \gamma_2)} \text{ for } 0 < j < \mathsf{N}_{\mathsf{rows}}$
Calculate $g_L(X), h_L(X)$:
    $g_L(X) = (a^{\texttt{compr}}(X) + \beta_2) \cdot (l^{\texttt{compr}}(X) + \gamma_2)$
    $h_L(X) = (a^{\text{perm}}(X) + \beta_2) \cdot (l^{\text{perm}}(X) + \gamma_2)$
Calculate lookup-related numerators of the quotient polynomial:
    $F_3(X) = L_0(X)(1 - V_L(X))$
    $F_4(X) = V_L(\omega X) \cdot h_L(X) - V_L(X) \cdot g_L(X)$
    $F_5(X) = q^{\text{last}}(X) \cdot (V_L(X)^2 - V_L(X))$
    $F_6(X) = L_0(X)(a^{\text{perm}}(X) - l^{\text{perm}}(X))$
    $F_7(X) = (1 - (q^{\text{last}}(X) + q^{\text{pad}}(X))) \cdot (a^{\text{perm}}(X) - l^{\text{perm}}(X)) \cdot (a^{\text{perm}}(X) - a^{\text{perm}}(\omega^{-1}X))$
return $F_3(X), F_4(X), F_5(X), F_6(X), F_7(X), V_L, \mathsf{lookup}^{\mathsf{perm}}, \widehat{\mathsf{lookup}^{\mathsf{perm}}}$
```

Algorithm 6 LookupArgumentVerify

```
Input: $\overline{\mathsf{lookup}^{\mathsf{perm}}}, \{a_i(y)\}, \{l_i(y)\}, a^{\mathrm{perm}}(y), a^{\mathrm{perm}}(\zeta^{-1} \cdot y), l^{\mathrm{perm}}(y), V_L(y), V_L(\zeta \cdot y), L_0(y), q^{\mathrm{last}}(y), q^{\mathrm{pad}}(y),$ transcript
Output: $F_3(y), F_4(y), F_5(y), F_6(y), F_7(y)$
Get challenge $\theta \in \mathbb{F}$ from transcript

    $\texttt{lookup\_gate}_j(y) = q_{lj} \cdot \sum_{k=0}^{\mathsf{C}_{\mathsf{lk}}-1} \sum_{i=0}^{\nu(k)-1} \theta^{\mathcal{N}_k+i} a_{j_i^k}(\zeta^{d_j^k} \cdot y), \quad \text{where } \mathcal{N}_k = \sum_{i=0}^{k-1} \nu(i)$
    $\texttt{table\_value}_j(y) = \sum_{k=0}^{\mathsf{C}_{\mathsf{lk}}-1} \sum_{i=0}^{\nu(k)-1} \theta^{\mathcal{N}_k+i} l_{j_i^k}(y),$

Construct the input lookup compression and table compression values for $1 \le j \le \mathsf{N}_{\mathsf{rows}}$:

    $a^{\texttt{compr}}(y) = \sum_{0 \le i < \mathsf{N}_{\mathsf{lk}}} \texttt{lookup\_gate}_i(y)$
    $l^{\texttt{compr}}(y) = \sum_{0 \le i < \mathsf{N}_{\mathsf{lk}}} \texttt{table\_value}_i(y)$

transcript.$append(\overline{\mathsf{lookup}^{\mathsf{perm}}})$
Get challenges $\beta_2, \gamma_2 \in \mathbb{F}$ from transcript
Denote:

    $g_L(y) = (a^{\texttt{compr}}(y) + \beta_2) \cdot (l^{\texttt{compr}}(y) + \gamma_2)$
    $h_L(y) = (a^{\mathrm{perm}}(y) + \beta_2) \cdot (l^{\mathrm{perm}}(y) + \gamma_2)$

Calculate:

    $F_3(y) = L_0(y)(1 - V_L(y))$
    $F_4(y) = (1 - (q^{\mathrm{last}}(y) + q^{\mathrm{pad}}(y))) \cdot (V_L(\zeta y) \cdot h_L(y) - V_L(y) \cdot g_L(y)$
    $F_5(y) = q^{\mathrm{last}}(y) \cdot (V_L(y)^2 - V_L(y))$
    $F_6(y) = L_0(y)(a^{\mathrm{perm}}(y) - l^{\mathrm{perm}}(y))$
    $F_7(y) = (1 - (q^{\mathrm{last}}(y) + q^{\mathrm{pad}}(y))) \cdot (a^{\mathrm{perm}}(y) - l^{\mathrm{perm}}(y)) \cdot (a^{\mathrm{perm}}(y) - a^{\mathrm{perm}}(\zeta^{-1} y))$

return $F_3(y), F_4(y), F_5(y), F_6(y), F_7(y)$
```

5 Placeholder Commitment Scheme

In this section, we describe the commitment scheme used in the Placeholder system. This scheme is based on LPC [6], which is generalization of polynomial commitment scheme. Commitment scheme is executed on a batch of polynomials $\{f_i(x)\}_{i=0}^{l-1}$ where number of polynomials l is one of scheme parameters. List polynomial commitment implies that in the query phase for point x, the values $f'_i(x)$ is returned (instead of $f_i(x)$), where

$$\Delta(f_i, f'_i) < \delta, \; f' \in \mathsf{RS}[\mathbb{F}_p, \mathcal{D}, \rho].$$

Let $d \in \mathbb{Z}$ be the minimum for which $2^d \ge \mathsf{N}_{\mathsf{rows}}$, as above. The Reed-Solomon code $\mathsf{RS}[\mathbb{F}_p, \mathcal{D}, \rho]$ is set by the parameters $\rho = 2^{-R}, \; R \in \mathbb{N}, \; |\mathcal{D}| = 2^k, \; k = d + R$.

5.1 Scheme Parameters

The binding scheme is parameterized with the following values.

- FRI localization factor m. Default value $m = 2$.
- Folding map $q(X) = X^m$, denote $q_j(X)$ a result of j times application map q applied on X. In our case it $q_j(x) = X^{2^j}$.
- $2^d = 2^{k-R}$ – max degree of polynomial for FRI protocol

- $r \in [1; \log d]$ – total number of FRI-rounds
- l – number of polynomials
- K – number of polynomials' precommitments.
- $l_0, \dots, l_{K-1} : \sum\limits_{i=0}^{K} l_i = l$ number of polynomials for each precommitment.
- Domains $D_0, \dots D_{r-1}$, such that:
  - $\mathcal{D}_i \subset \mathbb{F}$
  - $\mathcal{D}_0 = [\zeta, \dots, \zeta^n]$.
  - $\mathcal{D}_{i+1} = q(\mathcal{D}_i)$
  - $|\mathcal{D}_{i+1}| = \frac{|\mathcal{D}_i|}{m} = \frac{|\mathcal{D}_0|}{m^{i+1}}$
- Error-bound $\delta > 0$
- $\mathsf{steps_{FRI}}$ – number of FRI round proofs in FRI proof
- $r_0, \dots .r_{\mathsf{steps_{FRI}}-1} : \sum\limits_{i=0}^{\mathsf{steps_{FRI}}-1} r_i = r$, number of rounds in each step
- r_q – number FRI-queries are necessary for constructing FRI-proof;

5.2 Proof format

- LPC proof $\mathcal{P}$ contains:
  - Vector of evaluation values for each polynomial $\boldsymbol{z}_0, \dots, \boldsymbol{z}_{l-1}$
  - FRI proof π
- FRI proof π contains:
  - Merkle roots $\mathsf{fri\_root}_0, \dots, \mathsf{fri\_root}_{\mathsf{steps_{FRI}}}$ for commitments for each FRI step.
  - Query proof $\pi^{(i)}$ for $0 \le i < r_q$
  - $\texttt{final\_polynomial} = \{c_0^*, \dots, c_k^*\}$, $k = 2^{\log(d-l-r)}$
- Query proof $\pi^{(i)}$ for FRI contains:
  - Initial proof $\pi^{(i)*}$
  - Round proofs $\pi_j^{(i)}$ for $0 \le j < \mathsf{steps_{FRI}}$
- Initial proof for FRI contains:
  - Vector of evaluation values for each polynomial val_i for $0 \le i < l$. Length of each val_i equals m^{r_0}.
  - Merkle tree paths p_k for $0 \le k < K$
- Round proof for FRI contains:
  - Values for the one combined polynomial $\boldsymbol{y}$. Size of $\boldsymbol{y}$ equals 2^{r_i}
  - Merkle tree path auth

5.3 Implemented optimizations

In current version of algorithm FRI-proof π doesn't contain all r round proofs. There are only $\mathsf{steps_{FRI}}$ of them. Algorithm executes r_i FRI-rounds on i-th step to produce π_i^{FRI}. Total number of FRI-rounds is r.

Instead of committing each of the polynomials, we use the same Merkle tree for several polynomials. This leads to the decrease of the number of Merkle tree paths which are required to be provided by the prover. See [15], [6] for details.

Each $i+1$ FRI round supposes the prover to send all elements from a coset $H \in D^{(i)}$. Each Merkle leaf is able to contain the whole coset instead of separate values. See [15] for details. Similar approach is described in [6]. However, the authors of [6] use more values per leaf, that leads to better performance.

Instead of checking each commitment individually, we aggregate them for FRI. For polynomials $f_0, \dots, f_l$:

1. Get τ from transcript
2. $f(X) = \sum_{k=0}^{l-1} \tau^{l-k-1} f_k(X)$

3. Run FRI over f, using precommitments to $f_0, \dots, f_l$

Thus, we run only one FRI instance for all committed polynomials. See [6] for details.

5.4 LPC Preprocessing

In the scheme commit/opening for a polynomials $\{f_i\}$ describe a δ-list of functions f_i' such that

$$\Delta(f_i, f_i') < \delta \text{ for } i = 0, \dots, l-1,$$

where Δ is Hamming weight function.

For the polynomials that define a circuit, we require one more check. Each commit/opening should describe exactly one polynomial from the list. For that, Preprocessing step is used.

Algorithm 7 Preprocessing

```
Input: Set of polynomials $\{g_i(X)\}_{i=0}^{l-1}$ that are required to be preprocessed
Output: Set of distinguishing points $\{\xi_i, \nu_i\}_{i=1}^{l-1}$, where $\nu_i = g$
for all $g_i(X)$ $(i = 1, \dots, n)$ do
    Prover and Verifier agree on distinguishing point $x_i \in \mathbb{F}$ and value $\nu_i = g_i(x_i)$, such that:
        $\forall g' \in L_\delta(g_i) : g'(x_i) \neq g_i(x_i)$, where $L_\delta(f) := \{f' \ : \ f' \in \mathsf{RS}[\mathbb{F}_p, \mathcal{D}, \rho] \wedge \Delta(f, f') < \delta\}$.
end for
```

5.5 Merkle trees usage

To commit polynomial values protocol uses Merkle tree MT. The MT needs following algorithms:

- $\mathsf{Create}(leaves)$ – each leaf of Merkle tree commits values of polynomials $\{f_i\}_{i=0}^{l-1}$ on cosets $S \subset D$. Coset has the following structure $S = \forall s_i, s_j \in S : q_r(s_i) = q_r(s_j)$ for some r.
- $\mathsf{Proof}(tree, leaf)$ – generates Merkle proof for the leaf.
- $\mathsf{Validate}(proof, leaf)$ – checks if `proof` corresponds `leaf` data. Input of this function should contain $f_i(s) \forall i = 0, \dots, l-1, s \in S$.
- $\mathsf{Precommit}(\{f_i\}_{i=0}^{l-1}, D, r)$ – Splits domain D into cosets, and generates Merkle tree (see algorithm 8).
- $\mathsf{Commit}(T)$ – returns root of tree T.

Algorithm 8 MT.Precommit

```
Input: polynomials $\{f_i(X)\}_{i=0}^{l-1}$, domain $D$, $r$ – coset size parameter.
Output: Merkle tree $T$.
Split domain $D$ into cosets $\{S \subset D : \forall s_i, s_j \in S, q_r(s_i) = q_r(s_j)\}$. If $m = 2$ $|S| = 2^r$.
For each coset $S$ calculate $f$ over all elements of $S$ : $\texttt{leaf}_S = \{f_i(x)\}_{i=0,\dots,l-1,x \in S}$. Number of leaves is $\frac{|D|}{|S|}$
Build Merkle tree $T = \mathsf{MT.Create}(\{\texttt{leaf}_S\})$.
$T$ is precommitment.
```

5.6 Proof eval

Algorithm 9 LPC.EvalProof

```
Input:
$l$ polynomials $g_0, \dots, g_{l-1}$, splitted into $K$ sequental subsets with sizes $l_0, \dots, l_K$, where $\sum_{i=0}^{K} l_i = l$;
vector of evaluation points $\xi^{(i)}$ for each polynomial $g_i$;
Merkle trees $T_0, \dots, T_K$ where $T_k$ is precommitment for $k$-th subset $g_{\sum_{i=0}^{k} l_i}, \dots g_{(\sum_{i=0}^{k+1} l_i)-1}$ for $0 \leq k < K$;
transcript
Output: Proof $\mathcal{P}$
Calculate $\boldsymbol{z}_j^{(k)} = g_k(\xi_j^{(k)})$ for $0 \leq k < l, 0 \leq j < |\xi^{(k)}|$
MultiEval:
for all $k$ from $0$ to $l-1$ do
    Interpolate $U_k(X)$ such that $U_k(\xi_j^{(k)}) = \boldsymbol{z}_j^{(k)}$ for $0 \leq j < |\xi^{(k)}|$
                                    ▷ $U_k(X) \neq g_k(X)$ since $\deg(U_k) < \deg(g_k)$
    Calculate $Q_k(X) = \frac{g_k(X) - U_k(X)}{\prod_{j=0}^{|\xi^{(k)}|-1}(X - \xi_j^{(k)})}$ with $\deg(Q_k) \leq d' = d - |\xi^{(k)}|$
end for
$\pi$ = FRI.Proof($Q_0(X), \dots, Q_{l-1}(X), g_0(X), \dots, g_{l-1}(X), T_0, \dots, T_{K-1}$, transcript) with rate $\rho = \frac{d'}{|D|}$ and error-bound $\delta$
$\mathcal{P} = \{\boldsymbol{z}^{(0)}, \dots, \boldsymbol{z}^{(l-1)}, \pi\}$
```

Algorithm 10 FRI.Proof

```
Input: Polynomials $\{Q_k(X)\}_{k=0}^{l-1}, \{g_k(x)\}_{k=0}^{l-1}$, Merkle trees $\{T_k\}_{k=0}^{K-1}$, transcript
Output: Proof $\pi$
$\mathcal{D}^{(0)} = \mathcal{D}, \quad \mathcal{D}^{(i+1)} = q_{r_i}\left(\mathcal{D}^{(i)}\right)$, for $i = 0, \ldots, \mathsf{steps}_{\mathsf{FRI}} - 1$
Commit phase:
for all $k$ from 0 to $K - 1$ do
    transcript.append(MT.Commit($T_k$))
end for
$\tau =$ transcript.challenge($\mathbb{F}$)
Calculate polynomial $Q(X) := \sum_{k=0}^{l-1} \tau^{l-k-1} Q_k(X)$
$f_0(X) = Q(X)$
$t = 0$
for all $i = 0, \ldots, \mathsf{steps}_{\mathsf{FRI}} - 1$ do
    tree$_i$ := MT.Precommit($f_i(X), \mathcal{D}_i(X), r_i$)
    fri_root$_i$ := MT.Commit(tree$_i$)
    transcript.append(fri_root$_i$)
    $f(X) := f_i(X)$
    for all $step = 0, \ldots, r_i - 1$ do
        $\alpha_t$ := transcript.challenge($\mathbb{F}$)
        $f'(X) := \mathsf{interpolant}^f_{\alpha_t}(X)$ where $f'(X)$ is defined on $\mathcal{D}^{(t+1)}$ this way:
            $$\forall s \in \mathcal{D}^{(t+1)}, S = \{s_j \in \mathcal{D}^{(t)} | q(s_j) = s\}, |S| = m$$
            $$\mathsf{interpolant}^f_{\alpha_t}(s) = \mathsf{lagrange\_interpolation}(\{s_j, f(s_j)\}_{s_j \in S})(\alpha_t)$$
        $f(X) := f'(X), t := t + 1$
    end for
    $f_{i+1}(X) := f(X)$
end for
final_polynomial($X$) $= f_{\mathsf{steps}_{\mathsf{FRI}}}(X)$
Query phase:
for all $query = 0, \ldots, \mathsf{queries}_{\mathsf{FRI}} - 1$ do
    $x^{(0)}$ := transcript.challenge($\mathcal{D}_0$)
    $x^{(i+1)} := q_{r_i}(x^{(i)})$, for $i = 0, \ldots, \mathsf{steps}_{\mathsf{FRI}} - 1$
    Construct cosets $S^{(i)} = \{s \in \mathcal{D}^{(i)} \mid q_{r_i}(s) = x^{(i+1)}\}$,
    $S^{(\mathsf{steps}_{\mathsf{FRI}})} = \{x^{(\mathsf{steps}_{\mathsf{FRI}})}\}$ for $i = 0, \ldots, \mathsf{steps}_{\mathsf{FRI}} - 1$        ▷ $|S^{(i)}| = m^{r_i}$
    t = 0;
    for all $k = 0, \ldots, K$ do
        $\mathsf{val}^{(j)} := \{g_j(s) \ \ \forall s \in S^{(i)}\}, t \le j < t + l_k - 1$
        $p_k$ := MT.Proof($T_k, \{\mathsf{val}^{(t)}, \ldots, \mathsf{val}^{(t+l_k-1)}\}$)
        $t := t + l_k$
    end for
    Construct initial proof $\pi^* = \{\mathsf{val}^{(0)}, \ldots, \mathsf{val}^{(l-1)}, p_0, \ldots, p_{K-1}\}$
    $\boldsymbol{y}^{(0)} := \{f_0(s), \ \ \forall s \in S^{(0)}\}$        ▷ $f_0(X) = Q(X)$
    for all $i := 0, \ldots, \mathsf{steps}_{\mathsf{FRI}} - 1$ do
        auth$_i$ := MT.Proof(tree$_i, \boldsymbol{y}^{(i)}$)
        $\boldsymbol{y}^{(i+1)} := \{f_{i+1}(s), \ \ \forall s \in S^{(i+1)}\}$
        Construct round proof $\pi_i = \{\mathsf{auth}_i, \boldsymbol{y}^{(i+1)}\}$
    end for
    Construct query proof $\pi^{(round)} = \{\pi^*, \pi_0, \ldots, \pi_{\mathsf{steps}_{\mathsf{FRI}}-1}\}$
end for
Construct FRI proof $\pi = \{\mathsf{fri\_root}_0, \ldots, \mathsf{fri\_root}_{K-1}, \mathsf{final\_polynomial}, \pi^{(0)}, \ldots, \pi^{\mathsf{queries}_{\mathsf{FRI}}-1}\}$
```

5.7 Verify eval

Algorithm 11 LPC.EvalVerify

```
    Input:
proof $\mathcal{P}$,
evaluation points $\{\xi^{(k)}\}_{k=0}^{l-1}$,
roots of Merkle trees $\{\mathsf{root}_k\}_{k=0}^{K-1}$,
transcript
    Output: verification result = true/false
$\{\boldsymbol{z}^{(0)}, \dots, \boldsymbol{z}^{(l-1)}, \pi\} = \mathtt{parse}(\mathcal{P})$
Interpolate polynomials $U_k(X) = \mathsf{lagrange\_interpolation}(\{\xi_j^{(k)}, \boldsymbol{z}_j^{(k)}\})$ for $0 \leq k < l, 0 \leq j < |\xi^{(k)}|$
Compute $V_k(X) = \prod_{j=0}^{|\xi^{(k)}|-1}(X - \xi_j^{(k)})$
if FRI.Verify($\pi, \{\mathsf{root}_k\}_{k=0}^{K}, \{U_k(X)\}_{k=0}^{l-1}, \{V_k(X)\}_{k=0}^{l-1}$, transcript) = false then return false
return true
```

Algorithm 12 FRI.Verify

```
    Input: FRI proof $\pi$, Merkle roots $\{\mathsf{T\_root}_k\}_{k=0}^{K-1}$, $\{U_k(X)\}_{k=0}^{l-1}$, $\{V_k(X)\}_{k=0}^{l-1}$, transcript
    Output: verification result = true/false
$\{\mathsf{fri\_root}_0, \ldots, \mathsf{fri\_root}_{\mathsf{steps_{FRI}}-1}, \pi^{(0)}, \ldots \pi^{(r_q-1)}, \mathsf{final\_polynomial}\} = \mathtt{parse}(\pi)$
$\mathcal{D}^{(0)} = \mathcal{D}, \quad \mathcal{D}^{(i+1)} = q_{r_i}\left(\mathcal{D}^{(i)}\right)$, for $i = 0, \ldots, \mathsf{steps_{FRI}} - 2$
for all $k = 0, \ldots, L-1$ do
    $\mathsf{transcript}.\mathit{append}(\mathsf{T\_root}_k)$
end for
$\tau := \mathsf{transcript}.\mathit{challenge}(\mathbb{F})$
$t := 0$
for all $i := 0, \ldots, \mathsf{steps_{FRI}} - 1$ do
    $\mathsf{transcript}.\mathit{append}(\mathsf{fri\_root}_i)$
    for all $\mathit{step} := 0, \ldots, r_i - 1$ do
        $\alpha_t := \mathsf{transcript}.\mathit{challenge}(\mathbb{F})$
        $t := t + 1$
    end for
end for
for all $\mathit{query} = 0, \ldots, r_q - 1$ do
    $\{\pi^*, \pi_0, \ldots, \pi_{\mathsf{steps_{FRI}}-1}\} = \mathtt{parse}(\pi^{(round)})$
    $x^{(0)} = \mathsf{transcript}.\mathit{challenge}(\mathcal{D}_0)$
    $x^{(i+1)} = q_{r_i}(x^{(i)}),\ i = 0, \ldots, \mathsf{steps_{FRI}} - 1$
    Construct cosets $S^{(i)} = \{s \in \mathcal{D}^{(i)} \mid q_{r_i}(s) = x^{(i+1)}\}$, for $i = 0, \ldots, \mathsf{steps_{FRI}} - 1$        ▷ $|S^{(i)}| = m^{r_i}$
    Initial proof check
    t := 0;
    for all $k := 0, \ldots, K-1$ do
        if $\pi^*.\mathsf{auth}_k.\mathsf{root} \neq \mathsf{T\_root}_k$ then return $\mathit{false}$
        if $\mathsf{MT.Validate}(\pi^*.\mathsf{auth}_k, \{\pi^*.\mathsf{val}^{(t)}, \ldots, \pi^*.\mathsf{val}^{(t+l_k-1)}\}) = \mathit{false}$ then return $\mathit{false}$
        $t := t + l_k$
    end for
    Compute values of combined polynomial $Q$ values $\mathsf{val}$ from $\pi_k.\mathsf{val}$
        $$\mathsf{val} = \left\{ \prod_{k=0}^{l-1} \tau^{l-k-1} \frac{\pi^*.\mathsf{val}_s^{(k)} - U_k(s)}{V_k(s)} \right\}_{s \in S^{(0)}}$$
    Round proofs check
    $t := 0, S := S^{(0)}$
    for all $i := 0, \ldots, \mathsf{steps_{FRI}} - 1$ do
        if $\pi_i.\mathsf{auth.root} \neq \pi.\mathsf{fri\_root}_i$ then return $\mathit{false}$
        if $\mathsf{MT.Validate}(\pi_i.\mathsf{auth}, \mathsf{val}) = \mathit{false}$ then return $\mathit{false}$
        for all $\mathit{step} := 0, \ldots, r_i - 1$ do
            $S_{\mathsf{next}} := \{q(s)\}_{s \in S}$
            $\mathsf{interpolant}_s := \mathsf{lagrange\_interpolation}(\{s_j, \mathsf{val}_{s_j}\}_{q(s_j)=s})(\alpha_t)$ for $s \in S_{\mathsf{next}}$
            $t := t + 1, S := S_{\mathsf{next}}, \mathsf{val} := \{\mathsf{interpolant}_s\}_{s \in S_{\mathsf{next}}}$        ▷ $|S_{\mathsf{next}}| = r_i - \mathit{step} - 1$
        end for
        if $\mathsf{val} \neq \pi_i.\boldsymbol{y}_{(x^{(i+1)})}$ then return $\mathit{false}$        ▷ $|\mathsf{val}| = 1$
        $\mathsf{val} := \pi_i.\boldsymbol{y}$
    end for
    if $\mathsf{final\_polynomial}(x^{(\mathsf{steps_{FRI}})}) \neq \mathsf{val}$ then return $\mathit{false}$
end for
return true
```

5.8 Soundness

According to [16] the soundness error of LPC is defined by $\varepsilon(\delta) \leq \max\{\varepsilon_{\mathsf{FRI}}, \varepsilon_{\mathsf{IOP}}, 1/|\mathbb{F}|\}$, where

$$\varepsilon_{\mathsf{IOP}} = \left(\frac{1}{2\eta\sqrt{\rho}}\right)^6 \cdot \frac{4n}{|\mathbb{F} \setminus \mathcal{D}|},$$
$$\varepsilon_{\mathsf{FRI}}(\delta) \leq \frac{2 \cdot n}{\eta^3|\mathbb{F}|} + (\sqrt[3]{\rho} + 2\sqrt[3]{\eta} + \eta \cdot n)^{\mathsf{l_{FRI}}}.$$

6 KZG Commitment Scheme

A polynomial commitment scheme is a tuple of algorithms $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{EvalProof}, \mathsf{EvalVerify})$. The trusted setup algorithm Gen takes as input maximum supported degree bound d (and maximum number t of evaluation points in case of batched version) and outputs public *structured reference string* srs. The prover can then commit to polynomial $f \in \mathbb{F}_p^{<d}[X]$ using srs: $\mathsf{Commit}(f, srs) = cm$. Subsequently, the prover can invoke $\mathsf{EvalProof}$ to produce a proof π that convinces the verifier who runs the $\mathsf{EvalVerify}$, that the polynomial «inside» cm has the degree less than d and, moreover, evaluate to the claimed value s (values $\{s_i\}_{i=0}^{t-1}$ for batched version) at a given evaluation point z (points $\{z_i\}_{i=0}^{t-1}$ – respectively).

Polynomial commitment schemes KGZ, introduced in [11], uses a triple of groups (G_1, G_2, G_3) with an efficiently computable non-degenerate bilinear pairing $e : G_1 \times G_2 \to G_3$. Let P_i be generators of G_i for $i = 1, 2, 3$. We denote $x \cdot P_i$ by $[x]_i$ for $i = 1, 2$ and any $x \in \mathbb{F}_p$. A trusted setup Gen generates srs which contains powers of a random field element $\alpha \in \mathbb{F}_p$: $(P_1, \alpha \cdot P_1, ..., \alpha^{d-1} \cdot P_1, P_2, \alpha \cdot P_2)$. The value of α must remain secret. For any polynomial $f \in \mathbb{F}_p^{<d}[X]$, $f = \sum_{i=0}^{d-1} c_i X^i$ commitment to f defined by $\mathsf{Commit}(f) = [f(\alpha)]_1$ that can be calculated using srs: $[f(\alpha)]_1 = \left(\sum_{i=0}^{d-1} c_i \cdot \alpha^i\right) \cdot P_1 = \sum_{i=0}^{d-1} c_i \cdot \mathsf{srs}_i$. To prove that $f(z) = s$, the $\mathsf{EvalProof}$ simply outputs a commitment $\pi = [h(\alpha)]_1$ to the quotient polynomial $h = (f(X) - s)/(X - z)$. A correctly generated proof will satisfy $e(\pi, [\alpha]_2 - [z]_2) = e(h(\alpha) \cdot P_1, (\alpha - z)P_2) = e((f(\alpha) - s) \cdot P_1, P_2)$. The proof is accepted by the verifier $(\mathsf{EvalVerify})$ if and only if $e([f(\alpha)]_1 - [s]_1, [1]_2) = e(\pi, [\alpha - z]_2)$.

For the performance of the Placeholder, it is highly desirable to use a version of the protocol that allows it to query multiple committed polynomials at multiple points at a time. An efficient batch version of the KZG is described in [17]. We refer to this scheme as $\mathsf{KZG1}$. In algorithms 13, 14, 15 we provide a detailed description of the scheme.

Algorithm 13 KZG1.Gen

```
Input: $d, t$
Output: srs
$\alpha \in_R \mathbb{F}_p$
srs $= ([1]_1, [\alpha]_1, \ldots, [\alpha^{d-1}]_1, [1]_2, [\alpha]_2, \ldots, [\alpha^t]_2)$
```

Algorithm 14 KZG1.EvalProof

```
Input: srs, $T = \{z_0, \ldots, z_{t-1}\}$, $\{S_i \subset T\}_{i \in [k]}$, $\{f_i\}_{i \in [k]}$, transcript
Output: $\pi$
$\theta =$ transcript.$challenge(\mathbb{F}_p)$
Calculate polynomials $\{r_i(X)\}_{i \in [k]}$:
    $$r_i \in \mathbb{F}_p^{<|S_i|}[X] : \ \forall z \in S_i : \ r_i(z) = f_i(z)$$
Calculate polynomial $h(X)$:
    $$h(X) = \sum_{i=0}^{k-1} \theta^i \cdot \frac{f_i(X) - r_i(X)}{Z_{S_i}(X)}$$
Compute $\pi = [h(\alpha)]_1$
transcript.$append(\pi)$
return $\pi$
```

Algorithm 15 KZG1.EvalVerify

```
Input: $\pi$, srs, $T = \{z_0, \ldots, z_{t-1}\}$, $\{S_i \subset T\}_{i \in [k]}$, $\{f_i\}_{i \in [k]}$, $\{r_i\}_{i \in [k]}$, $\{cm_i = [f_i(\alpha)]_1\}_{i \in [k]}$, transcript
Output: true/false
Get challenges $\theta \in \mathbb{F}_p$ from transcript
Calculate $\{Z_i\}_{i \in [k]}$:
    $$Z_i = [Z_{T \setminus S_i}(\alpha)]_2$$
Calculate $F$:
    $$F = \prod_{i=0}^{k-1} e(\theta^i \cdot (cm_i - [r_i(\alpha)]_1), Z_i)$$
if $F = e(\pi, [Z_T(\alpha)]_2)$ then
    return true
else
    return false
end if
```

We also allow using another batch version of KZG, as described in [17]. We refer to it as KZG2. The evaluation proof of this scheme consists of two group elements, but it has better verifier complexity. We present the KZG2 algorithms in 17, 16, 18. Notice that the commit algorithm is again defined in the standard way: KZG2.Commit$(f, \text{srs}) = [f(\alpha)]_1$.

Algorithm 16 KZG2.EvalProof

```
Input: srs, $\{f_i\}_{i\in[k]}$, $T = \{z_0, \dots, z_{t-1}\}$, $\{S_i \subset T\}_{i\in[k]}$, transcript
Output: $\boldsymbol{\pi}$
$\theta = \mathsf{transcript}.challenge(\mathbb{F}_p)$
Calculate polynomials $\{r_i(X)\}_{i\in[k]}$:
    $r_i \in \mathbb{F}_p^{<|S_i|}[X] : \ \forall z \in S_i : \ r_i(z) = f_i(z)$
Calculate polynomial $f(X)$:
    $f(X) = \sum_{i=0}^{k-1} \theta^i \cdot Z_{T\setminus S_i}(X) \cdot (f_i(X) - r_i(X))$
$\boldsymbol{\pi}_1 = [(f/Z_T)(\alpha)]_1$
transcript.append($\boldsymbol{\pi}_1$)
$\theta_2 = \mathsf{transcript}.challenge(\mathbb{F}_p)$
Calculate polynomial $L(X)$:
    $L(X) = \sum_{i=0}^{k-1} \theta^i \cdot Z_{T\setminus S_i}(\theta_2) \cdot (f_i(X) - r_i(\theta_2)) - $
    $- Z_T(\theta_2) \cdot \left( \frac{f(X)}{Z_T(X)} \right)$
Compute $\boldsymbol{\pi}_2 = [(L/(X - \theta_2))(\alpha)]_1$
transcript.append($\boldsymbol{\pi}_2$)
return $\boldsymbol{\pi} = (\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$
```

Algorithm 17 KZG2.Gen

```
Input: $d$
Output: srs
$\alpha \in_R \mathbb{F}_p$
$\mathsf{srs} = ([1]_1, [\alpha]_1, \dots, [\alpha^{d-1}]_1, [1]_2, [\alpha]_2)$
```

Algorithm 18 KZG2.EvalVerify

```
Input: srs, $\{f_i\}_{i\in[k]}$, $T = \{z_0, \dots, z_{t-1}\}$, $\{S_i \subset T\}_{i\in[k]}$, $\pi$, $\{r_i\}_{i\in[k]}$, transcript
Output: true/false
Get challenges $\theta, \theta_2 \in \mathbb{F}_p$ from transcript
Calculate $F$:
    $F = \sum_{i=0}^{k-1} \theta^i \cdot Z_{T\setminus S_i}(\theta_2) \cdot (cm_i - [r_i(\theta_2)]_1) -$
    $- Z_T(\theta_2) \cdot \boldsymbol{\pi}_1$
if $e(F, [1]_2) = e(\boldsymbol{\pi}_2, [\alpha - \theta_2]_2)$ then
    return true
else
    return false
end if
```

7 Placeholder Protocol

7.1 Protocol Parameters

Here we summarize system parameters described earlier.

Parameter	Meaning
$\mathsf{N}_{\mathsf{rows}}$	Number of rows $\boldsymbol{\mathcal{T}}$
$\mathsf{N}_{\mathsf{col}}$	Number of columns $\boldsymbol{\mathcal{T}}$
N_{sl}	Number of selector columns
N_{lk}	Number of lookup-table columns
N_{cn}	Number of constant columns
N_{pi}	Number of public input columns
N_{wt}	Number of witness columns
$\boldsymbol{q}_i, q_i(x)$	i-th selector column of $\boldsymbol{\mathcal{T}}$ and corresponding interpolant polynomial
$\boldsymbol{l}_i, l_i(x)$	i-th lookup-table column of $\boldsymbol{\mathcal{T}}$ and corresponding interpolant polynomial
$\boldsymbol{c}_i, c_i(x)$	i-th constant column of $\boldsymbol{\mathcal{T}}$ and corresponding interpolant polynomial
$\boldsymbol{s}_i, s_i(x)$	i-th public input column of $\boldsymbol{\mathcal{T}}$ and corresponding interpolant polynomial
$\boldsymbol{w}_i, w_i(x)$	i-th witness column of $\boldsymbol{\mathcal{T}}$ and corresponding interpolant polynomial
$\mathsf{N}_{\mathsf{perm}}$	Number of witness columns that are included in the permutation argument
C_{bs}	Number of constraints polynomials
C_{dg}	Max total degree of basic constraints polynomials
C_{lk}	Number of lookup constraints
$\mathbf{o}$	Set of all offsets
$\mathcal{C}_i(x)$	Constraint polynomials, $0 \le i < \mathsf{C}_{\mathsf{bs}}$
$\mathcal{G}_i(x)$	Gate polynomials, $0 \le i < \mathsf{C}_{\mathsf{gt}}$ for selector $q_i(X)$
$\widehat{\sigma}(\text{col}: j, \text{row}: i) = (\text{col}: j', \text{row}: i')$	Permutation over the table

Table 1: *System parameters*

At the begining of the algorithm we prepare some polynomials and their precommitments to be sent to commtiment scheme. This is the table of all polynomails we need and necessary evaluation points for each polynomial.

Polynomials	Evaluation points
variable	
$\{w_i(X)\}_{i=0}^{\mathsf{N}_{\mathsf{wt}}-1}$	$y, \zeta^d \cdot y$ for all correcponding $d \in \mathbf{o}$
$\{s_i(X)\}_{i=0}^{\mathsf{N}_{\mathsf{pi}}-1}$	$y, \zeta^d \cdot y$ for all correcponding $d \in \mathbf{o}$
V_polynomials	
$V^{\sigma}(X)$	$y, \zeta \cdot y$
$V_L(X)$	$y, \zeta \cdot y$
$\text{lookup}^{\text{perm}}$	
$a^{\text{perm}}(X)$	$y, \zeta^{-1} \cdot y$
$l^{\text{perm}}(X)$	y
quotient	
$\{T_i(X)\}_{i=0}^{N_T}$	y
fixed	
$\{S_i^e(X), S_i^{\sigma}(X)\}_{i=0}^{\mathsf{N}_{\mathsf{perm}}-1}, q^{\text{last}}(X), q^{\text{pad}}(X), L_0(X)$	y
$\{c_i(X)\}_{i=0}^{\mathsf{N}_{\mathsf{cn}}-1}$, $\{l_i(X)\}_{i=0}^{\mathsf{N}_{\mathsf{lk}}-1}$, $\{1_i(X)\}_{i=0}^{\mathsf{N}_{\mathsf{sl}}-1}$	$y, \zeta^d \cdot y$ for all correcponding $d \in \mathbf{o}$

Table 2: *Evaluation points*

For details on polynomial commitment scheme and polynomial evaluation scheme, we refer the reader to [6].

7.2 Preprocessing

Algorithm 19 Prove

```
    Output: preprocessed_data
Init transcript with public data
Let $\zeta \in \mathbb{Z}_q$ be a primitive $2^d$-th root of unity, $H = \{\zeta^0, ..., \zeta^{n-1}\}$ be a cyclic subgroup of $\mathbb{F}^*$
Calculate public polynomials from public table $\boldsymbol{\mathcal{T}}$columns:
    $\{l_0(x), \ldots, l_{\mathsf{N}_{\mathsf{lk}}-1}(X), q_0(x), ..., q_{\mathsf{N}_{\mathsf{sl}}-1}(x), s_0(x), \ldots, s_{\mathsf{N}_{\mathsf{cn}}-1}(x)\}$
Let $d \in \mathbb{Z}_q$ be the minimum for which this $n = 2^d \geq \mathsf{N}_{\mathsf{rows}}$.
Let $\gamma$ be a $\mathcal{B}$ root of unity, where $\mathcal{B} \cdot 2^{d^*} + 1 = p$, $d \leq d^*$, $\mathcal{B}$ odd and $p$ is a size of the field.
Compute $\mathsf{N}_{\mathsf{perm}}$ identity permutation polynomials: $S_i^e(X)$ such that $S_i^e(\zeta^j) = \gamma^i \cdot \zeta^j$
Compute $\mathsf{N}_{\mathsf{perm}}$ permutation polynomials $S_i^\sigma(X)$ such that $S_i^\sigma(\zeta^j) = \gamma^{i'} \cdot \zeta^{j'}$
Compute polynomial $q^{\text{last}}(X)$, an interpolation of vector $\boldsymbol{q}_i^{\text{last}} : \boldsymbol{q}_i^{\text{last}} = 1$ for $i = \mathsf{N}_{\mathsf{rows}}$ and 0 overwise.
Compute polynomial $q^{\text{pad}}(X)$ , an interpolation of vector $\boldsymbol{q}_i^{\text{pad}} : \boldsymbol{q}_i^{\text{last}} = 1$ for $i > \mathsf{N}_{\mathsf{rows}}$ and 0 overwise.
Compute lagrange polynomial $L_0(X)$.
Accumulate a batch of polynomials that are known by prover and verifier and are not dependent of public input. They will be sent to the commitment scheme together.
    $\mathsf{fixed} = \{S_0^e, \ldots, S_{\mathsf{N}_{\mathsf{perm}}-1}^e, S_0^\sigma, \ldots, S_{\mathsf{N}_{\mathsf{perm}}-1}^\sigma, c_0, \ldots, c_{\mathsf{N}_{\mathsf{cn}}-1}, q_0, \ldots, q_{\mathsf{N}_{\mathsf{sl}}-1}, l_0, \ldots, l_{\mathsf{N}_{\mathsf{lk}}-1}, q^{\text{last}}, q^{\text{last}}, L_0\}$
$\widehat{\mathsf{fixed}} = \mathsf{MT.Commit}(\mathsf{fixed})$
$Z(X) = \prod_{a \in H}(X - a) = X^{N_{\text{rows}}} - 1$
return preprocessed_data = $\{\mathsf{fixed}, \widehat{\mathsf{fixed}}, Z(X)\}$
```

7.3 Prover View

The commitment scheme is described in Section 5. Details on the optimizations are in Section 8.

Algorithm 20 Prove

```
Output: $\pi_{\mathsf{Placeholder}}$, preprocessed_data
Accumulate table columns which depends on public input into batch
    $$\mathsf{variable} = \{w_0, \dots, w_{\mathsf{N_{wt}}-1}, s_0, \dots, s_{\mathsf{N_{pi}}}\}$$
$\widehat{\mathsf{variable}} = \mathsf{MT.Precommit}(\mathsf{variable})$,   $\overline{\mathsf{variable}} = \mathsf{MT.Commit}(\widehat{\mathsf{variable}})$,   $\mathsf{transcript}.\mathit{append}(\overline{\mathsf{variable}})$
Denote polynomials included in permutation argument as $f_0, \dots, f_{\mathsf{N_{perm}}-1}$
    $$F_0(X), F_1(X), F_2(X), V^{perm} = \mathsf{PermArgument}(f_0, \dots, f_{\mathsf{N_{perm}}-1}, \{S_j^e(X)\}, \{S_j^\sigma(X)\},\ q^{\text{last}}(x),\ q^{\text{pad}}(x),\ \mathsf{transcript})$$
Denote polynomials included in lookup argument as $a_0, \dots, a_{\mathsf{N_{lk}}-1}$
    $$F_3(X), F_4(X), F_5(X), F_6(X), F_7(X), V_L, \mathsf{lookup}^{\mathsf{perm}}, \widehat{\mathsf{lookup}^{\mathsf{perm}}} = \mathsf{LookupArgument}(a_0, \dots, a_{\mathsf{N_{lk}}-1}, \mathsf{transcript})$$
$\mathsf{V\_polynomials} = \{V^\sigma, V_L\}$
$\widehat{\mathsf{V\_polynomials}} = \mathsf{MT.Precommit}(\mathsf{V\_polynomials})$
$\overline{\mathsf{V\_polynomials}} = \mathsf{MT.Commit}(\widehat{\mathsf{V\_polynomials}})$,   $\mathsf{transcript}.\mathit{append}(\overline{\mathsf{V\_polynomials}})$
$\theta = \mathsf{transcript}.\mathit{challenge}(\mathbb{F})$
For $i = 0, \dots, \mathsf{N_{sl}} - 1 :$   $g_i(X) = q_i(X) \cdot (\theta^{k_i-1+\nu_i}\mathcal{C}_{i_0}(X) + \dots + \theta^{\nu_i}\mathcal{C}_{i_{k-1}}(X))$
Calculate a constraints-related numerator of the quotient polynomial:
    $$F_8(X) = \sum_{0 \le i < \mathsf{N_{sl}}} (g_i(X))$$
$\alpha_0, \dots, \alpha_8 = \mathsf{transcript}.\mathit{challenge}(\mathbb{F})$
Compute quotient polynomial $T(X)$:
    $$F(X) = \sum_{i=0}^{8} \alpha_i F_i(X), \quad T(X) = \frac{F(X)}{Z(X)}$$
$N_T := \mathtt{max}(\mathsf{N_{perm}}, \mathtt{deg}_{\mathtt{gates}} - 1)$, where $\mathtt{deg}_{\mathtt{gates}}$ is the highest degree of the degrees of gate polynomials
Split $T(X)$ into separate polynomials $\mathsf{quotient} = \{T_0(X), ..., T_{N_T-1}(X)\}$
$\widehat{\mathsf{quotient}} = \mathsf{MT.Precommit}(\mathsf{quotient})$
$\overline{\mathsf{quotient}} = \mathsf{LPC.Commit}(\widehat{\mathsf{quotient}})$,   $\mathsf{transcript}.\mathit{append}(\overline{\mathsf{quotient}})$
$y = \mathsf{transcript}.\mathit{challenge}(\mathbb{F}/H)$, $y \in \mathbb{F}/H$
The proof is $\pi_{\mathsf{Placeholder}} = (\pi_{\mathtt{comm}}, \pi_{\mathtt{eval}})$, where:
    $$\pi_{\mathtt{comm}} = \{\ \overline{\mathsf{variable}},\ \overline{\mathsf{V\_polynomials}},\ \overline{\mathsf{lookup}^{\mathsf{perm}}},\ \overline{\mathsf{quotient}},\ \overline{\mathsf{fixed}}\}$$
    $$\pi_{\mathtt{eval}} = \mathsf{LPC.EvalProof}($$
        $$\mathsf{variable}, \mathsf{V\_polynomials}, \mathsf{lookup}^{\mathsf{perm}}, \mathsf{quotient}, \mathsf{fixed},$$
        $$\mathsf{evaluation\_points},$$
        $$\widehat{\mathsf{variable}}, \widehat{\mathsf{V\_polynomials}}, \widehat{\mathsf{lookup}^{\mathsf{perm}}}, \widehat{\mathsf{quotient}}, \widehat{\mathsf{fixed}},$$
        $$\mathsf{transcript})$$
Evaluation points for each polynomials are defined in table 2.
```

7.4 Verifier View

Algorithm 21 Verify

```
Input: $\pi_{\mathsf{Placeholder}}$, preprocessed_data, transcript
Output: true/false
Parse proof $\pi_{\mathsf{Placeholder}}$ into:
$\pi_{\mathsf{comm}} = \{\ \overline{\mathsf{variable}},\ \overline{\mathsf{V\_polynomials}},\ \overline{\mathsf{lookup}^{\mathsf{perm}}},\ \overline{\mathsf{quotient}},\ \overline{\mathsf{fixed}}\}$
$\pi_{\mathtt{eval}}$ is evaluation proofs for
polynomial_evaluations $= \{$
    $w_i(y), w_i(\zeta^d \cdot y), i = 0, \dots, \mathsf{N}_{\mathsf{wt}} - 1, \quad s_i(y), s_i(\zeta^d \cdot y), i = 0, \dots, \mathsf{N}_{\mathsf{pi}} - 1$
        for all corresponding $d \in \mathbf{o}$,
    $V^\sigma(y), V^\sigma(\zeta \cdot y), a^{\mathrm{perm}}(y), a^{\mathrm{perm}}(\zeta^{-1} \cdot y), l^{\mathrm{perm}}(y), V_L(y), V_L(\zeta \cdot y),$
    $\{T_i(y)\}, i = 0, \dots, N_T - 1$
    $c_i(y), c_i(\zeta^d \cdot y), i = 0, \dots, \mathsf{N}_{\mathsf{cn}} - 1, \quad l_i(y), l_i(\zeta^d \cdot y), i = 0, \dots, \mathsf{N}_{\mathsf{lk}} - 1, \quad q_i(y), q_i(\zeta^d \cdot y), i = 0, \dots, \mathsf{N}_{\mathsf{sl}} - 1$
        for all corresponding $d \in \mathbf{o}$,
    $q^{\mathrm{last}}(y), q^{\mathrm{pad}}(y), L_0(y)\}$
Verify Permutation Argument:
    Denote polynomials included in permutation argument as $f_0, \dots, f_{\mathsf{N}_{\mathsf{perm}}-1}$
    Get values $\{f_i(y)\}, \{S_i^e(y)\}, \{S_i^\sigma(y)\}, V^\sigma(y), V^\sigma(\zeta \cdot y), L_0(y), q^{\mathrm{last}}(y), q^{\mathrm{pad}}(y)$ from $\pi_{\mathsf{Placeholder}}$
    Calculate
        $F_0(y), F_1(y), F_2(y) = \mathsf{PermArgumentVerify}(y, V^\sigma(y), V^\sigma(\zeta \cdot y), \{f_i(y)\},$
            $\{S_i^e(y)\}, \{S_i^\sigma(y)\}, q^{\mathrm{pad}}(y), q^{\mathrm{last}}(y), L_0(y),\ \mathsf{transcript})$
Verify Lookup Argument:
    Denote polynomials included in lookup argument as $a_0, \dots, a_{\mathsf{N}_{\mathsf{lk}}-1}$
    Get values $\{a_i(y)\}, \{l_i(y)\}, a^{\mathrm{perm}}(y), a^{\mathrm{perm}}(\zeta^{-1} \cdot y), l^{\mathrm{perm}}(y), V_L(y), V_L(\zeta \cdot y), L_0(y), q^{\mathrm{last}}(y), q^{\mathrm{pad}}(y)$ from $\pi_{\mathsf{Placeholder}}$
    Calculate:
        $F_3(y), F_4(y), F_5(y), F_6(y), F_7(y) = \mathsf{LookupArgumentVerify}($
            $\overline{\mathsf{lookup}^{\mathsf{perm}}}, \{a_i(y)\}, \{l_i(y)\},$
            $a^{\mathrm{perm}}(y), a^{\mathrm{perm}}(\zeta^{-1} \cdot y), l^{\mathrm{perm}}(y),$
            $V_L(y), V_L(\zeta \cdot y), L_0(y), q^{\mathrm{last}}(y), q^{\mathrm{pad}}(y),\ \mathsf{transcript})$
$\mathsf{transcript}.append(\overline{\mathsf{V\_polynomials}})$
Verify Basic Constraints:
    For $i = 0, \dots, \mathsf{N}_{\mathsf{sl}} - 1$
        $g_i(X) = q_i(X) \cdot (\theta^{k_i - 1 + \nu_i} \mathcal{C}_{i_0}(X) + \dots + \theta^{\nu_i} \mathcal{C}_{i_{k-1}}(X))$
    Calculate a constraints-related numerator of the quotient polynomial $F_8(y) = \sum_{0 \le i < \mathsf{N}_{\mathsf{sl}}} (g_i(y))$
Quotient polynomial check:
    if $\sum_{i=0}^{8} \alpha_i F_i(y) \neq Z(y)T(y)$ then return $false$
    Get challenges $\{\alpha_i \in \mathbb{F}\}_{i=0}^{8},\ \theta \in \mathbb{F},\ y \in \mathbb{F} \setminus H$ from transcript
    $\mathsf{transcript}.append(\overline{\mathsf{quotient}})$
Evaluation proof check
    if LPC.EvalVerify(polynomials_evaluations, $\pi_{comm}$, transcript) $= false$ then return $false$
```

8 Optimizations

8.1 Batched FRI

Instead of checking each commitment individually, it is possible to aggregate them for FRI. For polynomials $f_0, \dots, f_k$:

1. Get θ from transcript
2. $f = f_0 \cdot \theta^{k-1} + \dots + f_k$
3. Run FRI over f, using oracles to $f_0, \dots, f_k$

Thus, we can run only one FRI instance for all committed polynomials.

See [6] for details.

8.2 Hash By Column

Instead of committing each of the polynomials, it is possible to use the same Merkle tree for several polynomials. This leads to the decrease of the number of Merkle tree paths which are required to be provided by the prover.

See [15], [6] for details.

8.3 Hash By Subset

Each $i+1$ FRI round supposes the prover to send all elements from a coset $H \in D^{(i)}$. Each Merkle leaf is able to contain the whole coset instead of separate values.

See [15] for details. Similar approach is described in [6]. However, the authors of [6] use more values per leaf, that leads to better performance.

8.4 Grinding

$\mathsf{Placeholder}$ combines FRI with the grinding optimization. Grinding adds a special nounce to the proof, which is used as an additional parameter of the Fiat-Shamir technique and leads to a challenge that meets the desired criteria (for example, the first $\mathcal{B}$ bits are equal 0). This requires the honest prover to produce a proof-of-work witness before output a FRI proof. The advantage of this optimization is that it reduces the probability of accepting a fake proof by a factor of about $2^{\mathcal{B}}$, thus allowing for a smaller proof size.

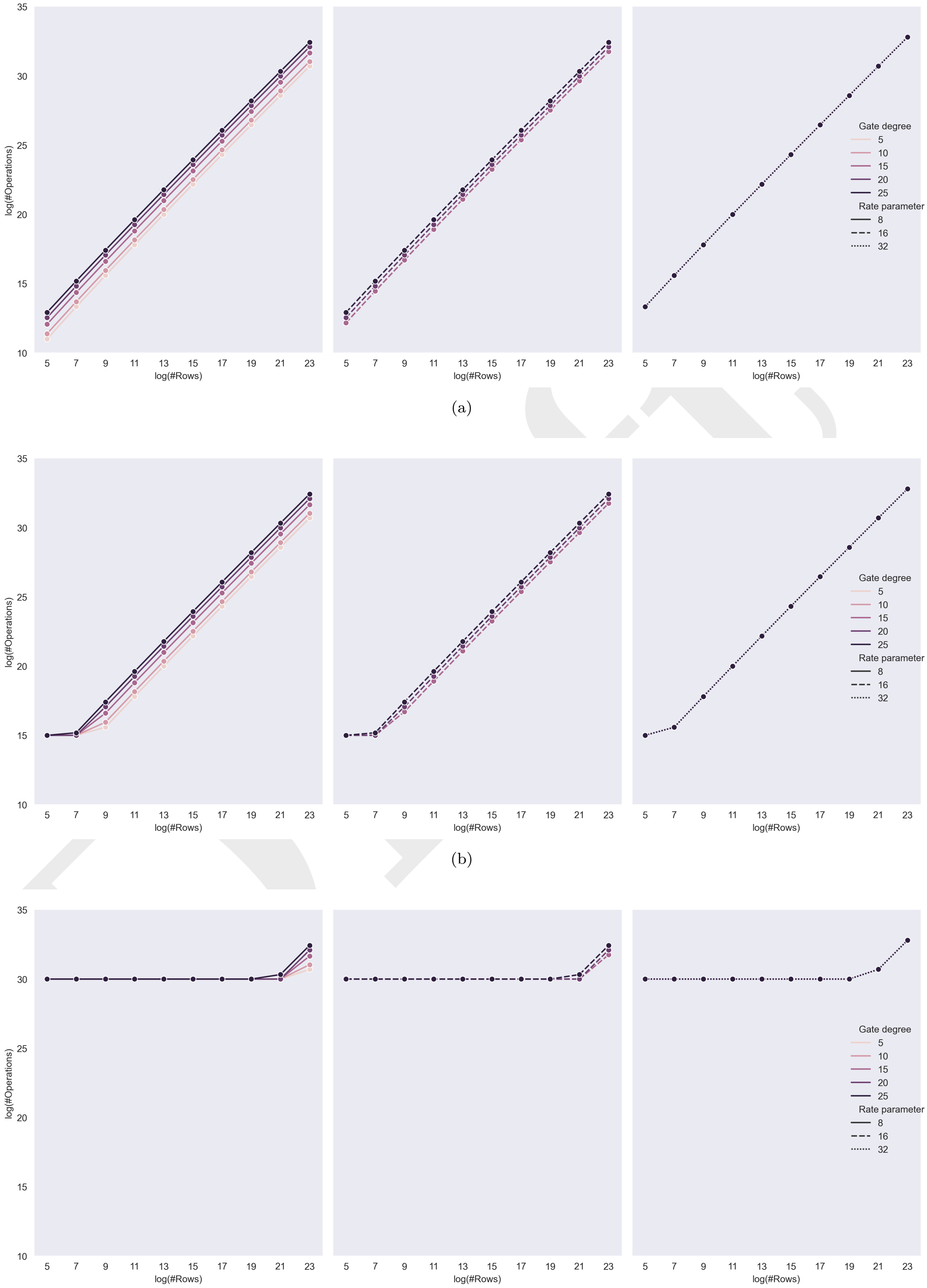
See [18] for details.

In Appendix B, we provide lower bounds for the parameter $\mathsf{queries}_{\mathsf{FRI}}$ for various levels of conjectured security and the grinding.

9 On the choice of parameters and asymptotic complexity

In this paragraph, we describe some nuances of choosing Placeholder's parameters for practical use. Choosing the code rate ($\rho = 1/2^{\mathcal{R}}$) allows you to achieve a tradeoff between the prover time and the size of the proof you get. Increasing the value of $\mathcal{R}$ reduces the number of repetitions of the Query phase of the FRI protocol and proof size. At the same time, the running time will be increased, since the polynomial for the commit will have degree $2^{d+\mathcal{R}}$. Placeholder actively uses custom gates, allowing more compact circuits for specific programs. However, using custom gates is not free. The quotient polynomial has degree $2^{d+\mathsf{C}_{\mathsf{gt}}}$, where C_{gt} is the maximum degree of the gate. Finally, the use of grinding introduces additional computational costs for proof generation. The complexity of this stage is estimated by the complexity of obtaining the preimage of the hash function for a given pattern of values.

Summarizing the above, in figure 2 we present a graphic representation of the dependence of the asymptotic complexity of the prover on the chosen parameters.



(a)

(b)

(c)

Figure 2: *Prover's asymptotic complexity with (a) 0 grinding bits (b) 15 grinding bits (c) 30 grinding bits*

10 Zero Knowledge

10.1 Cosets

As a part of modifications to achieve zero-knowledge property, [6] proposes to use a cosets of the sub-domains $D^{(i)}$ introduced in Section 5. Let $h \in \mathbb{F}^*/D$. Define domains $D^{(0)'} = hD^{(0)}, \dots, D^{(r)'} = hD^{(r)}$. FRI protocol works with new domains in the same way as described in Section 5.

10.2 Hiding Commitments

We use Merkle tree commitments with a pivacy adjustments from [19]. Each Merkle tree leaf contains concatenation of the original leaf data and a random value of the size 2λ for the given security parameter λ.

10.3 Random Rows

We use the same approach as Mina [20] and Halo [21]. The zero-knowledge adjustment is already included in the protocol in Section 7. In this section, we provide details on a PLONK-trace table preprocessing.

The basic idea is to fill the last t rows of the table with uniformly distributed random values. In this case, the values of the polynomials constructed during the protocol are uniformly distributed random values as well. The same is true for the last t evaluations of permutation and lookup polynomials. However, this change affects the permutation and lookup arguments.

Denote the number of usable rows by $N_{\texttt{usable}} = N_{\texttt{rows}} - t - 1$. Now we introduce two additional selectors:

- $q_{\texttt{blind}}$, $q_{\texttt{blind}}(\omega^i) = 1$ for $N_{\texttt{usable}} < i \leq N_{\texttt{rows}}$ and $q_{\texttt{blind}}$ is equal to zero elsewhere.
- $q_{\texttt{last}}$, $q_{\texttt{last}}(\omega^{N_{\texttt{usable}}}) = 1$ and $q_{\texttt{last}}$ is equal to zero elsewhere.

The new selectors and corresponding calculations are included in the protocol in Section 7.

References

1. *Gabizon A.*, *Williamson Z. J.*, *Ciobotaru O.* PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. — 2019. — https://ia.cr/2019/953. Cryptology ePrint Archive, Report 2019/953.
2. *Gabizon A.*, *Williamson Z. J.* Proposal: The Turbo-PLONK program syntax for specifying SNARK programs. — https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf.
3. The Halo 2 Proving System. — 2022. — URL: https://halo2.dev/ (visited on 02/06/2023).
4. Introducing Plonky2 — Polygon | Blog. — 2022. — URL: https://polygon.technology/blog/introducing-plonky2 (visited on 02/06/2023).
5. Kimchi: The latest update to Mina's proof system. — 2022. — URL: https://minaprotocol.com/blog/kimchi-the-latest-update-to-minas-proof-system (visited on 02/06/2023).
6. *Kattis A.*, *Panarin K.*, *Vlasov A.* RedShift: Transparent SNARKs from List Polynomial Commitment IOPs. — 2019. — https://ia.cr/2019/1400. Cryptology ePrint Archive, Report 2019/1400.
7. *Groth J.* On the Size of Pairing-based Non-interactive Arguments. — 2016. — URL: https://eprint.iacr.org/2016/260 (visited on 01/30/2023) ; Report Number: 260.
8. Pinocchio: Nearly Practical Verifiable Computation / B. Parno [et al.]. — 2013. — URL: https://eprint.iacr.org/2013/279 (visited on 01/30/2023) ; Report Number: 279.
9. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS / A. Chiesa [et al.]. — 2019. — URL: https://eprint.iacr.org/2019/1047 (visited on 01/27/2023) ; Report Number: 1047.
10. *Gabizon A.*, *Williamson Z. J.*, *Ciobotaru O.* PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. — 2019. — URL: https://eprint.iacr.org/2019/953 (visited on 01/03/2023) ; Report Number: 953.
11. *Kate A.*, *Zaverucha G. M.*, *Goldberg I.* Constant-Size Commitments to Polynomials and Their Applications // Advances in Cryptology - ASIACRYPT 2010. Vol. 6477 / ed. by D. Hutchison [et al.]. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. — P. 177–194. — DOI: 10.1007/978-3-642-17373-8_11. — URL: http://link.springer.com/10.1007/978-3-642-17373-8_11 (visited on 02/21/2023) ; Series Title: Lecture Notes in Computer Science.
12. Scalable, transparent, and post-quantum secure computational integrity / E. Ben-Sasson [et al.]. — 2018. — URL: https://eprint.iacr.org/2018/046 (visited on 01/30/2023) ; Report Number: 046.
13. Permutation argument - The halo2 Book. — 2021. — URL: https://zcash.github.io/halo2/design/proving-system/permutation.html#algorithm (visited on 01/20/2023).
14. Lookup argument - The halo2 Book. — 2021. — URL: https://zcash.github.io/halo2/design/proving-system/lookup.html (visited on 01/20/2023).
15. *Chiesa A.*, *Ojha D.*, *Spooner N.* Fractal: Post-Quantum and Transparent Recursive Proofs from Holography. — 2019. — URL: https://eprint.iacr.org/2019/1076 (visited on 01/20/2023) ; Report Number: 1076.
16. *Kattis A.*, *Panarin K.*, *Vlasov A.* RedShift: Transparent SNARKs from List Polynomial Commitments. — 2019. — URL: https://eprint.iacr.org/2019/1400 (visited on 01/20/2023) ; Report Number: 1400.
17. Efficient polynomial commitment schemes for multiple points and polynomials / D. Boneh [et al.]. — 2020. — URL: https://eprint.iacr.org/2020/081 (visited on 02/21/2023) ; Report Number: 081.
18. *StarkWare.* ethSTARK Documentation. — 2021. — URL: https://eprint.iacr.org/2021/582 (visited on 02/19/2023) ; Report Number: 582.
19. *Ben-Sasson E.*, *Chiesa A.*, *Spooner N.* Interactive Oracle Proofs. — 2016. — https://ia.cr/2016/116. Cryptology ePrint Archive, Report 2016/116.
20. A More Efficient Approach to Zero Knowledge for PLONK. — 2020. — URL: https://minaprotocol.com/blog/a-more-efficient-approach-to-zero-knowledge-for-plonk (visited on 01/20/2023).

21. Lookup argument - The halo2 Book. — 2021. — URL: https://zcash.github.io/halo2/design/proving-system/lookup.html#zero-knowledge-adjustment (visited on 01/20/2023).

22. Fast Reed-Solomon Interactive Oracle Proofs of Proximity : tech. rep. / E. Ben-Sasson [et al.]. — 2017. — No. 134. — URL: https://eccc.weizmann.ac.il/report/2017/134/ (visited on 02/16/2023).

23. *Ben-Sasson E.*, *Kopparty S.*, *Saraf S.* Worst-Case to Average Case Reductions for the Distance to a Code //. — Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 2018. — 23 pages. — DOI: 10.4230/LIPICS.CCC.2018.24. — URL: http://drops.dagstuhl.de/opus/volltexte/2018/8865/ (visited on 02/28/2023) ; Artwork Size: 23 pages Medium: application/pdf.

24. DEEP-FRI: Sampling Outside the Box Improves Soundness / E. Ben-Sasson [et al.]. — 2019. — URL: https://eprint.iacr.org/2019/336 (visited on 02/16/2023).

25. Proximity Gaps for Reed-Solomon Codes / E. Ben-Sasson [et al.]. — 2020. — URL: https://eprint.iacr.org/2020/654 (visited on 02/16/2023) ; Report Number: 654.

A FRI Soundness estimation

FRI (Fast Reed-Solomon Code Interactive Oracle Proof of Proximity) checks whether a received word f belongs to the RS code or it is δ-far in relative Hamming distance from all codewords. The rejection probability $\varepsilon(\delta)$ of the word that is δ-far from code is called soundness error. The following estimations are based on recent soundness analysis [22], [23], [24], [25].

Parameters

We start by recalling the parameters of the FRI protocol.

Parameter	Meaning
$\mathcal{R}$	Rate parameter of the RS code ($\rho = 2^{-\mathcal{R}}$)
$\mathcal{D},\ \lvert\mathcal{D}\rvert = 2^n$	Evaluation domain of the RS code
d	Maximum polynomial degree of $RS:\ d < \rho \cdot \lvert\mathcal{D}\rvert = 2^{n-R}$
μ	Localization parameter
m	Correlated Agreement parameter
$\mathsf{r}_{\mathsf{FRI}}$	Number of rounds $\mathsf{r}_{\mathsf{FRI}} \leq \lfloor (n-R)/m \rfloor$
$\mathsf{l}_{\mathsf{FRI}}$	Repetition parameter of Query phase
δ	Proximity parameter
ε	Error bound
κ	Number of output bits of the hash function used to construct the Merkle trees
λ	Desired level of bit-security

Table 3: *FRI Parameters*

The initial analysis

Let $\rho \cdot |\mathcal{D}| > 16$. Following [22] we can estimate error bound ($\delta^{(0)}$-far $f^{(0)}$ is rejected with probability at least):

$$\varepsilon(\delta^{(0)}) \geq 1 - \left(\frac{3 \cdot |\mathcal{D}|}{|\mathbb{F}|} + \left(1 - \min\left\{ \delta^{(0)}, \frac{1 - 3\rho - 2^{\mu}/\sqrt{|\mathcal{D}|}}{4} \right\} \right)^{\mathsf{l}_{\mathsf{FRI}}} \right),$$

where $\delta^{(0)} = \Delta(f^{(0)}, \mathsf{RS}[\mathbb{F}, \mathcal{D}, \rho])$. Consider the case of f that is maximally far from RS ($\delta^{(0)} \approx 1 - \rho$). Let $\mu = 1,\ \delta_T = (1-3\rho)/4 - 2^{-n/2-1}$ than for λ-bit soundness and far enough f we have to have

$$\begin{aligned}(1 - \varepsilon(\delta^{(0)})) &= \frac{3 \cdot 2^n}{|\mathbb{F}|} + (1 - \min\{\delta^{(0)}, \delta_T\})^{\mathsf{l}_{\mathsf{FRI}}} = \\ &= \frac{3 \cdot 2^n}{|\mathbb{F}|} + (1 - \delta_T)^{\mathsf{l}_{\mathsf{FRI}}} = \frac{3 \cdot 2^n}{|\mathbb{F}|} + \left(\frac{3}{4}(1+\rho) + 2^{-\frac{n}{2}-1} \right)^{\mathsf{l}_{\mathsf{FRI}}} \leq 2^{-\lambda}.\end{aligned}$$

Worst-case to average case reductions

For the case $\rho \cdot |\mathcal{D}| > 16$ following [23] we can estimate error bound:

$$\varepsilon(\delta^{(0)}) = 1 - \left(\frac{2 \cdot \log|\mathcal{D}|}{\eta^3 |\mathbb{F}|} + \left(1 - \min\left\{ \delta^{(0)}, J_\eta(J_\eta(1-\rho)) \right\} + \eta \log|\mathcal{D}| \right)^{\mathsf{l}_{\mathsf{FRI}}} \right),$$

For λ-bit soundness and far enough f we have to have

$$\begin{aligned}(1 - \varepsilon(\delta^{(0)})) &= \frac{2 \cdot \log|\mathcal{D}|}{\eta^3 |\mathbb{F}|} + \left(1 - \min\left\{ \delta^{(0)}, J_\eta(J_\eta(1-\rho)) \right\} + \eta \log|\mathcal{D}| \right)^{\mathsf{l}_{\mathsf{FRI}}} = \\ &= \frac{2 \cdot n}{\eta^3 |\mathbb{F}|} + (1 - J_\eta(J_\eta(1-\rho)) + \eta \cdot n)^{\mathsf{l}_{\mathsf{FRI}}} = \frac{2 \cdot n}{\eta^3 |\mathbb{F}|} + (\sqrt[4]{\rho} + 2\sqrt[4]{\eta} + \eta \cdot n)^{\mathsf{l}_{\mathsf{FRI}}} \leq 2^{-\lambda}.\end{aligned}$$

There is improvement [24] of the above equation

$$\sqrt[4]{\rho} \to \sqrt[3]{\rho}: \text{ which is optimal for } 2^n \approx q.$$

Correlated agreement

For the case $\rho \cdot |\mathcal{D}| > 16$ following [25] we can estimate error bound:

$$\begin{aligned}\varepsilon &= 1 - (\varepsilon_C + \varepsilon_Q^{\mathsf{l}_{\mathsf{FRI}}}) \\ &= 1 - \left(\frac{(m+\frac{1}{2})^7 \cdot |\mathcal{D}|^2}{2\rho^{3/2}|\mathbb{F}|} + \frac{(2m+1)\cdot(|D|+1)}{\rho} \cdot \frac{\sum_{i=1}^{r} l^{(i)}}{|\mathbb{F}|} + \left(\sqrt{\rho}\cdot\left(1+\frac{1}{2m}\right)\right)^{\mathsf{l}_{\mathsf{FRI}}} \right)\end{aligned}$$

For λ-bit soundness and far enough f we have to have

$$\begin{aligned}(1-\varepsilon(\delta^{(0)})) &= \frac{(m+\frac{1}{2})^7 \cdot |\mathcal{D}|^2}{2\rho^{3/2}|\mathbb{F}|} + \frac{(2m+1)\cdot(|D|+1)}{\rho} \cdot \frac{\sum_{i=1}^{r} l^{(i)}}{|\mathbb{F}|} + \left(\sqrt{\rho}\cdot\left(1+\frac{1}{2m}\right)\right)^{\mathsf{l}_{\mathsf{FRI}}} = \\ &= \frac{(m+\frac{1}{2})^7 \cdot 2^{2n}}{2\rho^{3/2}|\mathbb{F}|} + \frac{(2m+1)\cdot(2^n+1)}{\rho} \cdot \frac{2^n}{|\mathbb{F}|} + \left(\sqrt{\rho}\cdot\left(1+\frac{1}{2m}\right)\right)^{\mathsf{l}_{\mathsf{FRI}}} \leq 2^{-\lambda}.\end{aligned}$$

Method	Required number of repetitions of Query phase of FRI
Initial analysis [22]	$\mathsf{l}_{\mathsf{FRI}} \approx 2.4 \cdot \lambda$
Worst-case to average case reductions [23]	$\mathsf{l}_{\mathsf{FRI}} \approx (4/\mathcal{R})\lambda$
Improvements from [24]	$\mathsf{l}_{\mathsf{FRI}} \approx (3/\mathcal{R})\lambda$
Correlated agreement [25]	$\mathsf{l}_{\mathsf{FRI}} \approx (2/\mathcal{R})\lambda$
Aggressive conjecture [25]	$\mathsf{l}_{\mathsf{FRI}} \approx (1/\mathcal{R})\lambda$

Table 4: *FRI Soundness estimations*

B FRI Queries

Sec level	Grinding bits	Blowup factor $(1/\rho)$	Lower bound for $\mathsf{queries}_{\mathsf{FRI}}$
128	32	16	24
128	32	32	20
128	32	64	16
128	32	128	14
128	32	256	12
100	32	16	17
100	32	32	14
100	32	64	12
100	32	128	10
100	32	256	9
90	32	16	15
90	32	32	12
90	32	64	10
90	32	128	9
90	32	256	8
128	0	16	32
128	0	32	26
128	0	64	22
128	0	128	19
128	0	256	16
100	0	16	25
100	0	32	20
100	0	64	17
100	0	128	15
100	0	256	13
90	0	16	23
90	0	32	18
90	0	64	15
90	0	128	13
90	0	256	12

Table 5: $\mathsf{queries}_{\mathsf{FRI}}$ *estimations for conjectured security levels*