Placeholder

Yeah, that is the actual proof system name.

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

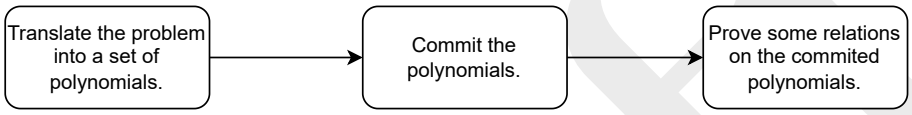
Ilia Shirobokov
=nil; Foundation
i.shirobokov@nil.foundation

Mikhail Komarov
=nil; Foundation
nemo@nil.foundation

June 19, 2022

1 Introduction

In general widespread SNARK constructions contains three steps as follows:



A proof system this paper introduces follows general SNARK construction and contains two "main modules":

- **Arithmetization** defines the arithmetic representation of the proving statement. We use PLONK-based [1] representation with custom gates. The idea was introduced in TurboPLONK paper [2] and modified later in other proof systems[1].
- **Commitment Scheme** for polynomials obtained from the arithmetization procedure. For these purposes, we use List Polynomial Commitment scheme from [3].

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.

1.1 PLONK Arithmetization with Custom Gates

Here we describe our instantiation of PLONK with custom gates.

The computation sequence that needs to be proved is called **Circuit**. **Circuit** is defined by **Table**, **Basic Constraints**, **Copy Constraints**, **Lookup Constraints**. Note that **Circuit** does not include witnesses, public input, and intermediate values.

It all starts with a rectangular matrix, which we'll refer as **Table**, represents a structure of the computations. **Rows**, **Columns** and **Cells** of this matrix are used with the conventional meanings.

There are four types of columns:

- **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.
- **Public Columns** contain public input.
  Public Columns differ between proof instances (because they depend on input). They are known to the verifier.
- **Fixed Columns** contain circuit-depended data. Fixed Columns do not differ between proof instances. They are known to the verifier.
- **Selectors** are special case of Fixed Columns. Selectors define to which rows of the Table the basic constraint is applied. Selectors' values can be only ones or zeroes. We provide details on Selectors later in this Section.

Table values are bound by three types of assertions:

[1] Halo2, Plonky, Plonky2, Kimchi, etc.

Column Type	Constant Values	Contains Public Data	Values
Witness	✗	✗	$\mathbb{F}$
Public	✗	✓	$\mathbb{F}$
Fixed	✓	✓	$\mathbb{F}$
Selectors	✓	✓	$\{1, 0\} \in \mathbb{F}$

1. Assertions that are imposed relation between Table values are referred as **Basic Constraints**. Basic Constraints are expressions (multivariate polynomials) over Cells.
2. **Copy Constraints** defines equality assertions between Cells.
3. **Lookup Constraints** assert that the chosen tuples of Cells of the Table are equal to some rows in **Lookup Table**. 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.

Table stores values used during computations, Constraints define relationships between these values.

Basic and Lookup Constraints may use Cells from the different Rows. If Constraint references values from the neighboring row, we call that reference as **Offset Reference** and the difference between constraint's row and referenced row as **Offset**. We also use **Absolute References** in Copy Constraints. **Absolute References** point to the number of the row instead of the difference between rows.

Constraint Type	Example
Basic Constraint	$2 \cdot T_{i,j} + T_{i+1,j} = 0$
Basic Constraint	$T_{i,j} \cdot T_{i+1,j} + T^2_{i+1,j+1} = 1$
Copy Constraint	$T_{i,j} = T_{1,0}$
Lookup Constraint	$(T_{i,j}, T_{i+1,j}, T_{i+2,j+1}) \in L$

Constraints examples for Table T and Lookup Table L

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.

To include Gate as an assertion to the particular Row, the value of the corresponding Selector's column of the Row has to be set to 1. Otherwise, it is set to 0.

Appendix A contains example of the circuit construction.

1.2 Public Input

Public Input takes a separate column or columns (in the case if one column is not enough, that is very unlikely) in the table. It is enforced in the witness columns via copy constraints.

2 RedShift Protocol

WIP

$N_{\texttt{rows}}$	Number of rows
$N_{\texttt{witness}}$	Number of witness columns
$N_{\texttt{perm}}$	Number of witness columns that are included in the permutation argument
$N_{\texttt{sel}}$	Number of selectors used in the circuit
$N_{\texttt{lookup}}$	Number of lookup constraints
$N_{\texttt{c}}$	Number of constraints polynomials
N_{PI}	Number of public input columns
w_i	Witness polynomials, $0 \leq i < N_{\texttt{witness}}$
$\texttt{c}_j^{(i)}$	Constraint polynomials, $0 \leq i < N_{\texttt{sel}}$
$\texttt{gate}_i$	Gate polynomials for selector $\mathbf{q}_i(X)$ and constraints $\{\mathbf{c}_j^{(i)}\}_{j=0}^{n_i'-1}$
PI_i	Public input polynomials, $0 \leq i < N_{PI}$
$\sigma(\text{col} : i, \text{row} : j) = (\text{col} : i', \text{row} : j')$	Permutation over the table
$\mathbf{o}$	Set of all offsets (see Section 1.1)

For details on polynomial commitment scheme and polynomial evaluation scheme, we refer the reader to [3].

We need to define how gates are constructed from constraints. Remind that gates contains a set of constraints with the same selector.

Each constraint may include different columns and offsets. Let constraint polynomial $\texttt{c}_j$ includes columns $w_{j,0}, \ldots, w_{j,k-1}$ for some $k \in \mathbb{N}$ with the corresponding offsets $d_{j,0}, \ldots, d_{j,k-1}$. It has the following form:

$$\texttt{c}_j = a_{j,0} \cdot w_{j,0}(\omega^{d_{j,0}} X) + \ldots a_{j,k} \cdot w_{j,k-1}(\omega^{d_{j,k-1}}),\ a_{j,i} \in \mathbb{F}$$

Denote by k_i the number of constraints used in i-th gate. Let ν_i be the initial degree of the random challenge for i-th gate.

$$\nu_{i+1} = \nu_i + k_i$$
$$\nu_0 = 0.$$

Then we can represent i-th gate as:

$$\texttt{gate}_i(X) = q_{l_i}(X) \cdot (\theta^{k_i - 1 + \nu_i} \texttt{c}_{0_i}(X) + \cdots + \theta^{\nu_i} \texttt{c}_{k_i - 1}(X))$$

Preprocessing:

Preprocessing from table to polynomials

1. $\mathcal{L}' = (\mathbf{q}_0, ..., \mathbf{q}_{N_{\texttt{sel}}})$
2. Let ω be a $2^k = N_{\texttt{rows}}$ root of unity.
3. Let δ be a T root of unity, where $T \cdot 2^S + 1 = p$, $k \leq S$, T odd and p is a size of the field.
4. Compute $N_{\texttt{perm}}$ permutation polynomials $S_{\sigma_i}(X)$ such that $S_{\sigma_i}(\omega^j) = \delta^{i'} \cdot \omega^{j'}$
5. Compute $N_{\texttt{perm}}$ identity permutation polynomials: $S_{id_i}(X)$ such that $S_{id_i}(\omega^j) = \delta^i \cdot \omega^j$
6. Let $H = \{\omega^0, ..., \omega^{N_{\texttt{rows}}-1}\}$ be a cyclic subgroup of $\mathbb{F}^*$
7. Let A_i be a witness columns, which are used for a lookup, and S_i be a table columns for some lookup, $i = 0, .., m$.
8. $Z(X) = \prod_{a \in H}(X - a) = X^{N_{\texttt{rows}}} - 1$

interpolation from ω^0

2.1 Prover View

Denote polynomial commitment generation function as **Commit**(.) The commitment scheme is described in Section 3. Details on the commitment scheme optimizations are in Section 7.

Define params and how to add params to transcript

Details on transcript

1. $\texttt{transcript.}\textbf{append}(\texttt{circuit\_params})$
2. $\texttt{transcript.}\textbf{append}(\textbf{Commit}(w_i(X)))$ for $0 \leq i < N_{\texttt{witness}}$
3. Denote witness polynomials included in permutation argument and public input polynomials as follows

$$f_0 \coloneqq w_0, f_1 \coloneqq w_1, \ldots, f_{N_{\texttt{perm}}+N_{PI}-1} = PI_{N_{PI}-1}$$

4. $F_0(X), F_1(X), F_2(X) = \texttt{permutation\_argument}(\text{transcript}, f_0, \ldots, f_{N_{\texttt{perm}}+N_{PI}-1}, \texttt{circuit\_params})$
5. Denote witness polynomials included in lookup argument as follows

$$a_0 \coloneqq A_0, a_1 \coloneqq A_1, \ldots, a_{m-1} = A_{m-1}$$

6. $F_3(X), F_4(X), F_5(X), F_6(X), F_7(X) = \texttt{lookup\_argument}(\text{transcript}, a_0, \ldots, a_{m-1}, \texttt{circuit\_params})$
7. Constraint-satisfability processing:
   - 7.1 $\tau = \texttt{transcript.}\textbf{get\_challenge}()$
   - 7.2 For $i = 0, \ldots, N_{\texttt{sel}} - 1$ (the details on the gate construction are presented above):
     - 7.2.1 $\texttt{gate}_i(X) = q_{l_i}(X) \cdot (\theta^{k_i-1+\nu_i}\texttt{c}_{0_i}(X) + \cdots + \theta^{\nu_i}\texttt{c}_{k_i-1}(X))$
   - 7.3 Calculate a constraints-related numerator of the quotinent polynomial:

$$F_8(X) = \sum_{0 \leq i < N_{\texttt{sel}}} (\texttt{gate}_i(X))$$

8. Quotient polynomial calculation:
   - 8.1 $\alpha_0, \ldots, \alpha_8 = \texttt{transcript.}\textbf{get\_challenge}()$
   - 8.2 Compute quotient polynomial $T(X)$:

$$F(X) = \sum_{i=0}^{8} \alpha_i F_i(X)$$
$$T(X) = \frac{F(X)}{Z(X)}$$

   - 8.3 $N_T \coloneqq \texttt{max}(N_{\texttt{perm}} + N_{PI}, \texttt{deg}_{\texttt{gates}} - 1)$, where $\texttt{deg}_{\texttt{gates}}$ is the highest degree of the degrees of gate polynomials

     check lookup degree

   - 8.4 Split $T(X)$ into separate polynomials $T_0(X), ..., T_{N_T-1}(X)$ to fit them into commitments [2]
   - 8.5 $\texttt{transcript.}\textbf{append}(\textbf{Commit}(T_i(X)))$ for $0 \leq i < N_T - 1$
9. Run evaluation proof:
   - 9.1 $y = \texttt{transcript.}\textbf{get\_challenge\_from}(\mathbb{F}/H)$, $y \in \mathbb{F}/H$
   - 9.2 Run evaluation scheme with the committed polynomials and the corresponding points from the set $\{y, y\omega^{-1}, y\omega, y\omega^d\}$ for $d \in \mathbf{o}$
   - 9.3 The proof is $(\pi_{\texttt{comm}}, \pi_{\texttt{eval}})$, where:
     - $\pi_{\texttt{comm}} = \{w_{0,\texttt{comm}}, \ldots, w_{N_{\texttt{witness}-1},\texttt{comm}}, V_{P,\texttt{comm}}, T_{0,\texttt{comm}}, ..., T_{N_T-1,\texttt{comm}}, A_{\texttt{perm},\texttt{comm}}, S_{\texttt{perm},\texttt{comm}}, V_{L,\texttt{comm}}\}$

[2] Commit scheme supposes that polynomials should be degree $\leq n$

- $\pi_{\texttt{eval}}$ is evaluation proofs for $w_i(y), w_i(y\omega^d), V_P(y), V_P(y\omega)$, $T_0(y), \ldots, T_{N_T-1}(y), A_{\texttt{perm}}(y), A_{\texttt{perm}}(y\omega^{-1}), S_{\texttt{perm}}(y), V_L(y), V_L(y\omega)$ for all corresponding $d \in \mathbf{o}$

Algorithm 1 Permutation Argument

Input: transcript, $f_0, \ldots, f_k$, `circuit_params`
Output: F_0, F_1, F_2

1. $\beta_1, \gamma_1 = \texttt{transcript.}\textbf{get\_challenge}()$
2. For $0 \leq j \leq N_{\texttt{rows}} - 1$

$$\texttt{id\_binding}_j = \prod_{i=0}^{N_{\texttt{perm}}+N_{PI}-1} \left(f_i(\omega^j) + \beta_1 \cdot S_{id_i}(\omega^j) + \gamma_1\right)$$
$$\sigma\texttt{\_binding}_j = \prod_{i=0}^{N_{\texttt{perm}}+N_{PI}-1} \left(f_i(\omega^j) + \beta_1 \cdot S_{\sigma_i}(\omega^j) + \gamma_1\right)$$

   Remark: Note that $\texttt{id\_binding}_j, \sigma\texttt{\_binding}_j$ are elements of $\mathbb{F}$, not polynomials.

3. Calculate V_P:

$$V_P(\omega) = V_P(\omega^{N_{\texttt{rows}}}) = 1$$
$$V_P(\omega^j) = \prod_{i=0}^{j-1} \frac{\texttt{id\_binding}_i}{\sigma\texttt{\_binding}_i} \text{ for } 0 < j < N_{\texttt{rows}}$$

4. $\texttt{transcript.}\textbf{append}(\textbf{Commit}(V_P))$
5. Calculate $g_{\texttt{perm}}(X), h_{\texttt{perm}}(X)$:

$$g_{\texttt{perm}}(X) \coloneqq \prod_{i=0}^{N_{\texttt{perm}}+N_{PI}-1} (f_i(X) + \beta_1 \cdot S_{id_i}(X) + \gamma_1)$$
$$h_{\texttt{perm}}(X) \coloneqq \prod_{i=0}^{N_{\texttt{perm}}+N_{PI}-1} (f_i(X) + \beta_1 \cdot S_{\sigma_i}(X) + \gamma_1)$$

6. Calculate permutation-related numerators of the quotinent polynomial:

$$F_0(X) = L_0(X)(1 - V_P(X))$$
$$F_1(X) = (1 - (q_{\texttt{last}}(X) + q_{\texttt{blind}}(X))) \cdot (V_P(\omega X) \cdot h_{\texttt{perm}}(X) - V_P(X) \cdot g_{\texttt{perm}}(X))$$
$$F_2(X) = q_{\texttt{last}}(X) \cdot (V_P(X)^2 - V_P(X))$$

Algorithm 2 Lookup Argument

Input: transcript, $a_0, \ldots, a_{m-1}$, `circuit_params`
Output: F_3, F_4, F_5, F_6, F_7

1. $\theta = \texttt{transcript.get\_challenge}()$
2. For $i = 0, \ldots, N_{\texttt{lookup}} - 1$ (see Section 6.3 for details):
   2.1 $\texttt{lookup\_gate}_i(X) = q_{l_i}(X) \cdot (\theta^{k_i - 1 + \nu_i} A_{0_i}(\omega^{d_{0_i}} X) + \cdots + \theta^{\nu_i} A_{k_i - 1}(\omega^{d_{k_i - 1}} X))$
   2.2 $\texttt{table\_value}_i(\omega^j) = q_{l_i}(\omega^j) \cdot (\theta^{k_i - 1 + \nu_i} S_{0_i}(\omega^j) + \cdots + \theta^{\nu_i} S_{k_i - 1}(\omega^j))$
3. Construct the input lookup compression and table compression values for $1 \leq j \leq N_{\texttt{rows}}$:

$$\mathbf{A}_{\texttt{compr}}(\omega^j) = \sum_{0 \leq i < N_{\texttt{lookup}}} \texttt{lookup\_gate}_i(\omega^j)$$
$$\mathbf{S}_{\texttt{compr}}(\omega^j) = \sum_{0 \leq i < N_{\texttt{lookup}}} \texttt{table\_value}_i(\omega^j)$$

4. Interpolate polynomials $A_{\texttt{compr}}(X), S_{\texttt{compr}}(X)$ from $\mathbf{A}_{\texttt{compr}}, \mathbf{S}_{\texttt{compr}}$
5. Produce the permutation polynomials $S_{\texttt{perm}}(X)$ and $A_{\texttt{perm}}(X)$ according to Section 6.1.
6. $\texttt{transcript.}\mathbf{append}(\mathbf{Commit}(A_{\texttt{perm}}))$, $\texttt{transcript.}\mathbf{append}(\mathbf{Commit}(S_{\texttt{perm}}))$
7. Compute $V_L(X)$ such that:

$$V_L(1) = V_L(\omega^{N_{\texttt{rows}}}) = 1$$
$$V_L(\omega^j) = \prod_{i=0}^{j-1} \frac{(A_{\texttt{compr}}(\omega^i) + \beta)(S_{\texttt{compr}}(\omega^i) + \gamma)}{(A_{\texttt{perm}}(\omega^i) + \beta)(S_{\texttt{perm}}(\omega^i) + \gamma)} \text{ for } 0 < j < N_{\texttt{rows}}$$

8. $\texttt{transcript.}\mathbf{append}(\mathbf{Commit}(V_L))$
9. $\beta_2, \gamma_2 = \texttt{transcript.get\_challenge}()$
10. Calculate $g_L(X), h_L(X)$:

$$g_L(X) = (A_{\texttt{compr}}(X) + \beta_2) \cdot (S_{\texttt{compr}}(X) + \gamma_2)$$
$$h_L(X) = (A_{\texttt{perm}}(X) + \beta_2) \cdot (S_{\texttt{perm}}(X) + \gamma_2)$$

11. Calculate lookup-related numerators of the quotinent polynomial:

$$F_3(X) = L_0(X)(1 - V_L(X))$$
$$F_4(X) = V_L(\omega X) \cdot h_L(X) - V_L(X) \cdot g_L(X)$$
$$F_5(X) = q_{\texttt{last}}(X) \cdot (V_L(X)^2 - V_L(X))$$
$$F_6(X) = L_0(X)(A_{\texttt{perm}}(X) - S_{\texttt{perm}}(X))$$
$$F_7(X) = (1 - (q_{\texttt{last}}(X) + q_{\texttt{blind}}(X))) \cdot (A_{\texttt{perm}}(X) - S_{\texttt{perm}}(X)) \cdot (A_{\texttt{perm}}(X) - A_{\texttt{perm}}(\omega^{-1} X))$$

2.2 Verifier View

1. Parse proof π into:
   - $\pi_{\texttt{comm}} = \{w_{0,\texttt{comm}}, \ldots, w_{N_{\texttt{witness}-1},\texttt{comm}}, V_{P,\texttt{comm}}, T_{0,\texttt{comm}}, ..., T_{N_T-1,\texttt{comm}}, A_{\texttt{perm},\texttt{comm}}, S_{\texttt{perm},\texttt{comm}}, V_{L,\texttt{comm}}\}$
   - $\pi_{\texttt{eval}}$ is evaluation proofs for $w_(y), w_i(y\omega^d), V_P(y), V_P(y\omega)$, $T_0(y), \ldots, T_{N_T-1}(y), A_{\texttt{perm}}(y), A_{\texttt{perm}}(y\omega^{-1}), S_{\texttt{perm}}(y), V_L(y), V_L(y\omega)$ for all corresponding $d \in \mathbf{o}$
2. $\texttt{transcript.}\mathbf{append}(\texttt{circuit\_params})$
3. $\texttt{transcript.}\mathbf{append}(w_{i,\texttt{comm}})$ for $0 \leq i < N_{\texttt{witness}}$
4. Denote witness polynomials included in permutation argument and public input polynomials as

$$f_0 \coloneqq w_0, f_1 \coloneqq w_1, \ldots, f_{N_{\texttt{perm}}+N_{PI}-1} = PI_{N_{PI}-1}$$

5. $F_0(y), F_1(y), F_2(y) = \texttt{permutation\_argument}(\text{transcript}, \texttt{circuit\_params})$

6. $F_3(y), F_4(y), F_5(y), F_6(y), F_7(y) = \texttt{lookup\_argument}(\text{transcript}, \texttt{circuit\_params})$
7. Constraints-satisfability processing:
   7.1 $\theta = \texttt{transcript.get\_challenge}()$
   7.2 For $i = 0, \ldots, N_{\texttt{sel}} - 1$:
      7.2.1 $\texttt{gate}_i(X) = q_{l_i}(X) \cdot (\theta^{k_i - 1 + \nu_i} \texttt{c}_{0_i}(X) + \cdots + \theta^{\nu_i} \texttt{c}_{k_i - 1}(X))$.
   7.3 Calculate:
$$F_8(y) = \sum_{0 \leq i < N_{\texttt{sel}}} (\texttt{gate}_i(y))$$
8. $\alpha_0, \ldots, \alpha_8 = \texttt{transcript.get\_challenge}()$
9. Evaluation proof check:
   9.1 $N_T := \texttt{max}(N_{\texttt{perm}} + N_{PI}, \texttt{deg}_{\texttt{gates}} - 1)$, where $\texttt{deg}_{\texttt{gates}}$ is the highest degree of the degrees of gate polynomials
   9.2 Let $T_{0,\texttt{comm}}, ..., T_{N_T - 1,\texttt{comm}}$ be commitments to $T_0(X), ..., T_{N_T - 1}(X)$
   9.3 $\texttt{transcript.append}(T_{i,\texttt{comm}})$ for $0 \leq i < N_T$
   9.4 $y = \texttt{transcript.get\_challenge\_from}(\mathbb{F}/H)$, $y \in \mathbb{F}/H$
   9.5 Run evaluation scheme verification with the committed polynomials and the points from the set $\{y, y\omega^{-1}, y\omega, y\omega^d\}$ for all corresponding $d \in \mathbf{o}$ to get values $w_i(y), w_i(y\omega^d), V_P(y), V_P(y\omega), T_j(y), A_{\texttt{perm}}(y), S_{\texttt{perm}}(y), V_L(y), V_L(y\omega^{-1}), V_L(y\omega)$
10. Quotient Polynomial Check:
   10.1 Check the identity:
$$\sum_{i=0}^{10} \alpha_i F_i(y) = Z(y)T(y)$$

Algorithm 3 Permutation Argument Verification

1. $\beta_1, \gamma_1 = \texttt{transcript.get\_challenge}()$
2. $\texttt{transcript.append}(V_{P,\texttt{comm}})$,
3. Denote (see Step 3 of the Prover's view for details on f_i):
$$g_{\texttt{perm}}(y) := \prod_{i=0}^{N_{\texttt{perm}} + N_{PI} - 1} (f_i(y) + \beta \cdot S_{id_i}(y) + \gamma)$$
$$h_{\texttt{perm}}(y) := \prod_{i=0}^{N_{\texttt{perm}} + N_{PI} - 1} (f_i(y) + \beta \cdot S_{\sigma_i}(y) + \gamma)$$
4. Calculate:
$$F_0(y) = L_0(y)(1 - V_P(y))$$
$$F_1(y) = (1 - (q_{\texttt{last}}(y) + q_{\texttt{blind}}(y))) \cdot (V_P(\omega y) \cdot h_{\texttt{perm}}(y) - V_P(y) \cdot g_{\texttt{perm}}(y))$$
$$F_2(y) = q_{\texttt{last}}(y) \cdot (V_P(y)^2 - V_P(y))$$

Algorithm 4 Lookup Argument Verification

1. $\theta = \texttt{transcript.get\_challenge}()$
2. $\texttt{transcript.append}(A_{\texttt{perm,comm}})$, $\texttt{transcript.append}(S_{\texttt{perm,comm}})$, $\texttt{transcript.append}(V_{L,\texttt{comm}})$
3. For $i = 0, \ldots, N_{\texttt{lookup}} - 1$ (see Section 6.3 for details):
   3.1 $\texttt{lookup\_gate}_i(y) \coloneqq q_{l_i}(y) \cdot (\theta^{k_i - 1 + \nu_i} A_{0_i}(\omega^{d_{0_i}} y) + \cdots + \theta^{\nu_i} A_{k_i - 1}(\omega^{d_{k_i - 1}} y))$
   3.2 $\texttt{table\_value}_i(y) \coloneqq q_{l_i}(y) \cdot (\theta^{k_i - 1 + \nu_i} S_{0_i}(y) + \cdots + \theta^{\nu_i} S_{k_i - 1}(y))$
4. Construct the input lookup compression and table compression:

$$A_{\texttt{compr}}(y) \coloneqq \sum_{0 \le i < N_{\texttt{lookup}}} \texttt{lookup\_gate}_i(y)$$
$$S_{\texttt{compr}}(y) \coloneqq \sum_{0 \le i < N_{\texttt{lookup}}} \texttt{table\_value}_i(y)$$

5. Denote (see Step 3 of the Prover's view for details on f_i):

$$g_L(y) = (A_{\texttt{compr}}(y) + \beta) \cdot (S_{\texttt{compr}}(y) + \gamma)$$
$$h_L(y) = (A_{\texttt{perm}}(y) + \beta) \cdot (S_{\texttt{perm}}(y) + \gamma)$$

6. $\beta_2, \gamma_2 = \texttt{transcript.get\_challenge}()$
7. Calculate:

$$F_3(y) = L_0(y)(1 - V_L(y))$$
$$F_4(y) = (1 - (q_{\texttt{last}}(y) + q_{\texttt{blind}}(y))) \cdot (V_L(\omega y) \cdot h_L(y) - V_L(y) \cdot g_L(y)$$
$$F_5(y) = q_{\texttt{last}}(y) \cdot (V_L(y)^2 - V_L(y))$$
$$F_6(y) = L_0(X)(A_{\texttt{perm}}(y) - S_{\texttt{perm}}(y))$$
$$F_7(y) = (1 - (q_{\texttt{last}}(y) + q_{\texttt{blind}}(y))) \cdot (A_{\texttt{perm}}(y) - S_{\texttt{perm}}(y)) \cdot (A_{\texttt{perm}}(y) - A_{\texttt{perm}}(\omega^{-1} y))$$

3 RedShift Commit / Evaluation Schemes

WIP

In this section, we define different structures that contains data (proofs and params). These structures only defines the data contained in the proof (commit scheme parameters). We do not specify implementation details on the data structures that are used to store the data.

3.1 Witness Polynomials

Generilize tha algorithm with η

Algorithm 5 Setup

- Field $\mathbb{F}$
- Folding map $q(X) = X^2$
- Localization factor m. Default value $m = 2$.
- Domains $D_0, \ldots D_{r-1}$, such that:
  - $D_i \subset \mathbb{F}$
  - $D_0 = [\omega, \ldots, \omega^n]$.
  - $D_{i+1} = q(D_i)$
  - $|D_{i+1}| = \frac{|D_i|}{m} = \frac{|D_0|}{m^{i+1}}$
- Error-bound $\delta > 0$
- Bound for degree of polynomial $d \in \mathbb{N}$
- The number of FRI rounds r

Details on commit

Algorithm 6 Commit

1. Calculate f over all elements of D: $\mathbf{y} = \{f(H)\}_{H \in D}$
2. Build a Merkle-tree T for set $\mathbf{y}$.
3. Root of T is commitment

Define T.`path`()

Details on `fri_params`

FRI paramerters `fri_params` are defined by the following structure:

- $D_0, \ldots D_{r-1}$
- $q(X)$
- $m = 2$
- `max_degree` $= d$
- The number of points to open k
- FRI rounds number r
- λ

LPC proof $\mathcal{P}$:

- Evaluation vales $z_0, \ldots, z_{k-1}$
- Merkle proofs $p_{z_0}, \ldots, p_{z_{k-1}}$
- FRI proofs $\texttt{fri\_proof}_0, \ldots \texttt{fri\_proof}_{\texttt{fri\_params}.\lambda - 1}$

details

FRI proof π contains:

- $\texttt{round\_proof}_i$ for $0 \le i < \texttt{fri\_params}.r - 1$
- $\texttt{final\_polynomial} = \{c_0, \ldots, c_k\}$, $k = 2^{\log d' - \texttt{fri\_params}.r}$

Algorithm 7 Proof Eval

Input: k points for evaluation g over them: $\{\xi_j\}_{j=0}^{k-1}$, commitment to g (root of Merkle tree T), `transcript`

1. Open $z_j = g(\xi_j)$ for $0 \le j < k$ from the commitment and add the along with Merkle paths p_{z_j} to the proof
2. `MultiEval`:
   - 2.1 Interpolate polynomial $U(X)$ such that $U(\xi_j) = z_j$ for $0 \le j < k$.
     Remark: Notice, that $U(X) \neq g(X)$ since $deg(U) < deg(g)$.
   - 2.2 Calculate $Q(X) = \frac{g(X) - U(X)}{\prod_{j=0}^{k-1}(X - \xi_j)}$
     Remark: $deg(Q) = d' = d - k$.
   - 2.3 for i from 0 to $\texttt{fri\_params}.\lambda - 1$:
     - 2.3.1 $\texttt{fri\_proof}_i = \texttt{FRI.Eval}(Q(X), g(X), T, \texttt{transcript}, \texttt{fri\_params})$ with rate $\rho = \frac{d'}{|D|}$ and error-dound δ
   - 2.4 $\mathcal{P} = \{z_0, \ldots, z_{k-1}, p_{z_0}, \ldots, p_{z_{k-1}}, \texttt{fri\_proof}_0, \ldots \texttt{fri\_proof}_{\lambda - 1}\}$

details

Round proof for FRI:

- $y_0, \ldots, y_m$ polynomial values
- $p_0, \ldots, p_m$ Merkle tree paths
- T Merkle tree root (commitment)
- `colinear_value`, `colinear_path`

Algorithm 8 FRI.Eval

Input: $Q(X), T, \texttt{transcript}, \texttt{fri\_params}$
Output: π

1. $f(X) \coloneqq Q(X)$, $f(X)$ can be represented as $\sum_{i=0}^{d-1} c_i X^i$
2. $x = \texttt{transcript}.\textbf{get\_challenge}()$
3. $r = \texttt{fri\_params}.r$
4. for $i = 0..r-1$:
   - 4.1 $\alpha = \texttt{transcript}.\textbf{get\_challenge\_from}(\mathbb{F})$, $\alpha \in \mathbb{F}$
   - 4.2 $x_{\texttt{next}} = \texttt{fri\_params}.q(x)$
   - 4.3 $d = \textbf{deg}(f(X))$
     Remark: $d \leq \texttt{fri\_params.max\_degree}$
   - 4.4 Fold an intermediate polynomial (here for $\texttt{fri\_params}.m = 2$):
     $$f_{\texttt{even}}(X^2) = \sum_{i=0}^{\frac{d+1}{2}-1} c_{2i} X^{2i}$$
     $$f_{\texttt{odd}}(X^2) = \sum_{i=0}^{\frac{d+1}{2}-1} c_{2i+1} X^{2i}$$
     $$f_{\texttt{next}}(X) = f_{\texttt{even}}(X^2) + \alpha \cdot f_{\texttt{odd}}(X^2)$$
   - 4.5 Get points for interpolation:
     - 4.5.1 Find all s_j from coset $S = \{s_j \in D_i : \texttt{fri\_params}.q(s_j) = x_{\texttt{next}}\}$, $|S| = \texttt{fri\_params}.m$
       Remark: For the case $\texttt{fri\_params}.m = 2$, all s_j can be found from the equation $x_{\texttt{next}} - X^2 = 0$. In other words, $s_0 = x$, $s_1 = -x$.
     - 4.5.2 $y_j = f(s_j)$ for $0 \leq j < \texttt{fri\_params}.m$
     - 4.5.3 Get paths to the openings:
       - if $i = 0$:
         $$p_j = T.\textbf{path}(g(s_j)) \text{ for } 0 \leq j < m$$
         Remark: During the first iteration, there is not commitment for $Q(X)$, only for $g(X)$.
       - Otherwise:
         $$p_j = T.\textbf{path}(y_j) \text{ for } 0 \leq j < m$$
   - 4.6 if $i < r - 2$:
     - 4.6.1 $T_{\texttt{next}} = \textbf{Commit}(f_{\texttt{next}}(X))$, the commit is calculated over domain $\texttt{fri\_params}.D_{i+1}$
     - 4.6.2 $\texttt{transcript}.\textbf{append}(T_{next})$
   - 4.7 Form a round proof:
     - 4.7.1 if $i < r - 1$:
       - $\texttt{colinear\_value} = f_{\texttt{next}}(x_{\texttt{next}})$
       - $\texttt{colinear\_path} = T_{\texttt{next}}.\textbf{path}(\texttt{colinear\_value})$
       - $\texttt{round\_proof}_i = \{y_0, \ldots, y_m, p_0, \ldots, p_m, T, \texttt{colinear\_value}, \texttt{colinear\_path}\}$
     - 4.7.2 else:
       - $d' = \textbf{deg}(Q(X))$
       - $\texttt{final\_polynomial} = \{c_0, \ldots, c_{2^{\log d' - r}}\}$, where $f_{\texttt{next}}(X) = \sum_{i=0}^{2^{\log d' - r}} c_i X^i$
   - 4.8 $x = x_{\texttt{next}}$
   - 4.9 $f = f_{\texttt{next}}$
   - 4.10 $T = T_{\texttt{next}}$
5. $\pi = \{\texttt{round\_proof}_0, \ldots \texttt{round\_proof}_{r-2}, \texttt{final\_polynomial}\}$

Algorithm 9 Verify Eval

Input: queries $\xi_0, \dots \xi_{k-1}$, proof $\mathcal{P}$, `fri_params`, `transcript`

1. Check Merkle proofs for $\mathcal{P}.p_{z_i}$ for $0 \le i < k$
2. `MultiEvalVerify`:
   2.1 Interpolate polynomial $U(X)$ such that $U(\xi_j) = z_j$ for $0 \le j < k$
   2.2 $V(X) = \prod_{j=0}^{k-1}(X - \xi_j)$
   2.3 for i from 0 to $\texttt{fri\_params}.\lambda - 1$:
      2.3.1 Abort if $\texttt{FRI.Verify}(\mathcal{P}.\texttt{fri\_proof}_i, \texttt{transcript}, \texttt{fri\_params}, U(X), V(X))$ returns 0

Algorithm 10 FRI.Verify

Input: FRI proof $\pi, \texttt{transcript}, \texttt{fri\_params}, U(X), V(X)$

1. $x = \texttt{transcript}.\textbf{get\_challenge}()$
2. $r = \texttt{fri\_params}.r$
3. for $i = 0..r-2$:
   3.1 $\alpha = \texttt{transcript}.\textbf{get\_challenge\_from}(\mathbb{F})$, $\alpha \in \mathbb{F}$
   3.2 $x_{\texttt{next}} = q(x)$
   3.3 Find all $s_j \in S = \{s_j \in D_i : \texttt{fri\_params}.q(s_j) = x_{\texttt{next}}\}$, $|S| = \texttt{fri\_params}.m$
   Remark: For the case $\texttt{fri\_params}.m = 2$, all s_j can be found from the equation $x_{\texttt{next}} - X^2 = 0$. In other words, $s_0 = x$, $s_1 = -x$.
   3.4 $\pi.\texttt{round\_proof}_i.T.\textbf{verify}(\pi.\texttt{round\_proof}_i.p_j)$ for $0 \le j < m$
   3.5 Get the polynomial values for $0 \le j < \texttt{fri\_params}.m$:
   - if $i = 0$:
   $$y_j = \frac{\pi.\texttt{round\_proof}_0.y_j - U(s_j)}{V(s_j)}$$
   Remark: $\pi.\texttt{round\_proof}_0.y_j$ are values of the original polynomial, not $Q(X)$. For this reason, we need to recompute them.
   - Otherwise:
   $$y_j = \pi.\texttt{round\_proof}_i.y_j$$
   3.6 if $i < r - 2$:
      3.6.1 $\texttt{transcript}.\textbf{append}(\pi.\texttt{round\_proof}_{i+1}.T)$
   3.7 Colinearity check:
      3.7.1 Interpolate $\texttt{interpolant}(X)$ from (s_j, y_j)
      3.7.2 $\pi.\texttt{round\_proof}_{i+1}.T.\textbf{verify}(\pi.\texttt{round\_proof}_i.\texttt{colinear\_path})$
      3.7.3 Check that $\texttt{interpolant}(\alpha) = \pi.\texttt{round\_proof}_i.\texttt{colinear\_value}$
   3.8 $x = x_{\texttt{next}}$
4. for the last round $r - 1$:
   4.1 Check that $\pi.\texttt{final\_polynomial}$ contains $2^{\log d' - r}$ elements
   4.2 $f(X) \coloneqq \sum_{i=0}^{2^{\log d' - r}} \pi.\texttt{final\_polynomial}.c_i \cdot X^i$
   4.3 Check that $f(x) = \pi.\texttt{round\_proof}_{r-2}.\texttt{colinear\_value}$

3.2 Circuit Polynomials

In the previous scheme commit/opening for a polynomial f describe a δ-list of functions f' such that

$$\Delta(f, f') < \delta,$$

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 11 Preprocessing

Input: Set of polynomials $g_i(X)$ that are required to be preprocessed

1. Prover and Verifier agree on separation points $x_i \in \mathbb{F}$ and values $\nu_i = g_i(\mu_i)$, such that:

$$\forall g_i' \in L_\delta(g_i) : g_i'(\mu_i) \neq g_i(\mu_i)$$

During the Proof Eval / Verify Eval algorithms for $g_i(X)$, additional pair (μ_i, ν_i) should be added to the list of evaluation points (ξ_j, z_j).

4 Zero Knowledge

4.1 Cosets

As a part of modifications to achieve zero-knowlege property, [3] proposes to use a cosets of the sub-domains $D^{(i)}$ introduced in Section 3. Let $h \in \mathbb{F}^*/D$. Define domains $D^{(0)'} = hD^{(0)}, \ldots, D^{(r)'} = hD^{(r)}$. FRI protocol works with new domains in the same way as described in Section 3.

4.2 Hiding Commitments

We use Merkle tree commitments with a pivacy adjustements from [4]. Each Merkle tree leaf contrains concatenation of the original leaf data and a random value of the size 2λ for the given security parameter λ.

introduce λ earlier

4.3 Random Rows

Use citation

We use the same approach as Mina[3] and Halo[4]. The zero-knowledge adjustment is already included in the protocol in Section 2. 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 and $q_{\texttt{last}}$ is equal to zero elsewhere.

The new selectors and corresponding calculations are included in the protocol in Section 2.

Details on how to calculate t

5 Permutation Argument Details

Here we describe the transformation from copy constraints to permutation argument described as a part of the protocol in Section 2.

5.1 Cells as Permutation Cycles

Let $c_{i,j}, c_{i',j'}$ be two cells of the table representation of the circuit's trace. Denote by $\texttt{value}(c)$ value of the cell c during circuit's trace computation. Copy constraint $\texttt{Cp}(c_{i,j}, c_{i',j'})$ asserts that $\texttt{value}(c_{i,j}) = \texttt{value}(c_{i',j'})$.

[3] https://minaprotocol.com/blog/a-more-efficient-approach-to-zero-knowledge-for-plonk
[4] https://zcash.github.io/halo2/design/proving-system/lookup.html#zero-knowledge-adjustment

Using copy constraints, we define permutation over the table's cells. The permutation can be presresented as a set of cycles[5]. Note that distinct cycles are disjoint. For each set of equal cells $\{c_i, \ldots, c_{i+k}\}$ define a cycle $C = (c_i, \ldots, c_k)$. C is 'sub-permutation' δ_C such that $\delta_C(c_j) = c_{j+1}$ for $i \leq j < k$ and $\delta_C(c_k) = c_i$.

Thus, 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. The circuit's permutation is defined as a composition of these cycles.

Example

5.2 Permutation Construction Algorithm

We use the same algorithm as Halo[6].
The state is represented as:

- a map **mapping** for the permutation itself;
- a map **aux** that keeps track of a distinguished element of each cycle;
- a map **sizes** that keeps track of the size of each cycle.

If x, y belong to the same cycle, then $\textbf{aux}(x) = \textbf{aux}(y)$. $\textbf{sizes}(\textbf{aux}(x))$ containts the size of the cycle containing x.

Remark: Here, we use one label for the element of the permutation for simplicity. However, it is $x = (i, j)$, where i is the cell's column and j is the cell's row.

Algorithm 12 Copy State Initialization

1. For all x (each x is one-element cycle):
   1.1 $\textbf{mapping}(x) = x$
   1.2 $\textbf{aux}(x) = x$
   1.3 $\textbf{sizes}(x) = 1$

Algorithm 13 Add Copy Constraint $\texttt{Cp}(x, y)$

1. `if` $\textbf{aux}(x) = \textbf{aux}(y)$:
   1.1 `return` // don't do anything if x, y belong to the same cycle
2. Let `left` be an input with a larger cycle (defined by **sizes**) and `right` the other one.
3. $\textbf{sizes}(\textbf{aux}(\texttt{left})) = \textbf{sizes}(\textbf{aux}(\texttt{left})) + \textbf{sizes}(\textbf{aux}(\texttt{right}))$ // the right cycle will be merged into the left cycle
4. $z = \textbf{aux}(\texttt{right})$
5. `do`: // set all pointers from `right` cycle to `left` cycle
   5.1 $\textbf{aux}(z) = \textbf{aux}(\texttt{left})$
   5.2 $z = \textbf{mapping}(z)$

   `while`$(z \neq \textbf{aux}(\texttt{right}))$
6. $\texttt{tmp} = \textbf{mapping}(\texttt{left})$ // actually merge cycles in mapping
7. $\textbf{mapping}(\texttt{left}) = \textbf{mapping}(\texttt{right})$
8. $\textbf{mapping}(\texttt{right}) = tmp$

[5] https://en.wikipedia.org/wiki/Permutation#Cycle_notation
[6] https://zcash.github.io/halo2/design/proving-system/permutation.html#algorithm

Example

5.3 Permutation Polynomial

The algorithm 13 outputs permutation σ as a resulting copy state. Now we need to transform the permutation to the permutation polynomials.

Let $\sigma(\text{col} : i, \text{row} : j) = (\text{col} : i', \text{row} : j')$. We can construct it using **mappring**$(x = (i, j)) = (i', j')$. Let ω be a 2^k root of unity, δ be a T root of unity, where $T \cdot 2^S + 1 = p$, $k \leq S$, T odd and p is a size of the field.

Define k

Now we can interpolate permutation polynomials as:

$$S_{id_i}(\omega^j) = \delta^i \cdot \omega^j \text{ for } i = 0, \ldots, N_{\texttt{perm}} - 1$$
$$S_{\sigma_i}(\omega^j) = \delta^{i'} \cdot \omega^{j'} \text{ for } i = 0, \ldots, N_{\texttt{perm}} - 1$$

$S_{id_i}(X)$ is called identity permutation polynomials, and $S_{\sigma_i}(X)$ is called permutation polynomials.

The constructed polynomials are used in the main protocol described in Section 2.

Do we need a separate explanation for permutation argument?

6 Lookup Argument

Here we describe the transformation from lookup constraints to lookup argument described as a part of the protocol in Section 2. We use the lookup argument proposed in Halo[7].

Let T be a PLONK-trace table. Let $\mathbf{S} = S_0, \ldots, S_{m-1}$ be a table with m columns and $N_{\mathbf{rows}}$ rows. Note, that $N_{\mathbf{rows}}$ is equal to the number of usable rows in T. For `lookup input` cells $(T_{i_0,j_0}, \ldots, T_{i_{m-1},j_{m-1}})$, lookup constraint allow to assert that $\mathbf{S}$ contains a row `lookup value` with values of these cells.

Denote columns of T that are included in lookup argument as $\mathbf{A} = A_0, \ldots, A_{m-1}$. We refer to A_i as input columns and to S_i as lookup columns.

The $\mathbf{A}$ and $\mathbf{S}$ contain the same number of rows. Moreover, each value in $\mathbf{A}$ has to be presented in $\mathbf{S}$ (we'll). Both $\mathbf{A}$ and $\mathbf{S}$ can contain duplicate. If it is necessary to extend one of the sets, we extend $\mathbf{S}$ with duplicates and $\mathbf{A}$ with dummy values known to be in $\mathbf{S}$.

Let $\theta \in \mathbb{F}$ is the verifier's challenge. We compress the columns A_i, S_i into two columns A, S as follow:

$$A_{\texttt{compr}} = \theta^{m-1} A_0 + \cdots + \theta A_{m-2} + A_{m-1}$$
$$S_{\texttt{compr}} = \theta^{m-1} S_0 + \cdots + \theta S_{m-2} + S_{m-1}$$

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

6.1 Permutation

Firstly, the prover calculates two additional columns $A_{\texttt{perm}}$ and $S_{\texttt{perm}}$ that are permutations of $A_{\texttt{compr}}$ and $S_{\texttt{compr}}$ respectively.

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

- All the cells of column $A_{\texttt{perm}}$ 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 $A_{\texttt{perm}}$ is the row that has the corresponding value in $S_{\texttt{perm}}$. The order of the other values in $S_{\texttt{perm}}$ can be arbitrary.

Similarly to Section 5, we use a grand product argument [1] to prove that $A_{\texttt{perm}}$, $S_{\texttt{perm}}$ are permutations of $A_{\texttt{compr}}$, $S_{\texttt{compr}}$ in the step **??**.

[7] https://zcash.github.io/halo2/design/proving-system/lookup.html

6.2 Assertion Check

The permuted columns are constructed in a such way that we can assert that all elements from $A_{\texttt{perm}}$ are presented in $S_{\texttt{perm}}$ with the following rules:

1. $(A_{\texttt{perm}}(X) - S_{\texttt{perm}}(X)) \cdot (A_{\texttt{perm}}(X) - A_{\texttt{perm}}(\omega^{-1}X))$ to ensure that either $A_{\texttt{perm}}[j] = S_{\texttt{perm}}[j]$ or $A_{\texttt{perm}}[j] = A_{\texttt{perm}}[j-1]$
2. $L_1(X) \cdot (A_{\texttt{perm}}(X) - S_{\texttt{perm}}(X))$. We need it because $(A_{\texttt{perm}}(X) - A_{\texttt{perm}}(\omega^{-1}X))$ is not a valid check on the first row.

In order to archieve zero-knowledge we use the following constraints:

1. $(1 - (q_{\texttt{last}}(X) + q_{\texttt{blind}}(X))) \cdot (V_L(\omega X) \cdot (A_{\texttt{perm}}(X) + \beta) \cdot (S_{\texttt{perm}}(X) + \gamma) - V_L(X) \cdot (A_{\texttt{compr}}(X) + \beta) \cdot (S_{\texttt{compr}}(X) + \gamma)) = 0$
2. $(1 - (q_{\texttt{last}}(X) + q_{\texttt{blind}}(X))) \cdot (A_{\texttt{perm}}(X) - S_{\texttt{perm}}(X)) \cdot (A_{\texttt{perm}}(X) - A_{\texttt{perm}}(\omega^{-1}X))$
3. $q_{\texttt{last}}(X) \cdot (V_L(X)^2 - V_L(X)) = 0$

6.3 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 $A_{\texttt{compr}}$ is calculated.

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

Now we need to introduce notations for general lookup representation.

- **Lookup Table**: a table of values with columns S_i
- **Lookup Input**: a set of cells of the PLONK table of the form $(a_0, \ldots, a_k)$
- **Compressed Lookup Table**: a column that represents jointed columns of all Lookup Tables.
- **Compressed Lookup Input**: a column that represents jointed column of all lookup inputs.
- **Lookup Constraint**: $(a_0, \ldots, a_{k_i-1}) \in S$ for some lookup table S
- **Lookup Expression**: Polynomial representation of the lookup constraint

Suppose the circuit C contains $N_{\texttt{tables}}$ lookup tables and $N_{\texttt{lookup}}$ lookup constraints.

Each lookup constraint may contain different width of input. Denote by k_i the number of elements in the lookup input of the i-th lookup constraint. Let d_{j_i} be the rotation of each element in the lookup constraint (i.e. shift by d_{j_i} rows). Let ν_i be the initial degree of the random challenge for i-th lookup constraint.

$$\nu_{i+1} = \nu_i + k_i$$
$$\nu_0 = 0.$$

Thus, the lookup expression would be:

$$\texttt{lookup\_gate}_i(X) = (\theta^{k_i-1+\nu_i} A_{0_i}(\omega^{d_{0_i}} X) + \cdots + \theta^{\nu_i} A_{k_i-1}(\omega^{d_{k_i-1}} X))$$

The compressed lookup input:

$$A_{\texttt{compr}}(\omega^j) = \textstyle\sum_{0 \le i < N_{\texttt{lookup}}} \texttt{lookup\_gate}_i(\omega^j)$$

Note that each column $A_{i,j}$, which represents one of the columns in table T, is not necessarily different from each other.

Let ν_i for a table value be defined in the same way as for lookup inputs. The compressed lookup table is computed similarly to compressed lookup input:

$$\texttt{table\_value}_i(\omega^j) = (\theta^{k_i-1+\nu_i} S_{0_i}(\omega^j) + \cdots + \theta^{\nu_i} S_{k_i-1}(\omega^j))$$
$$S_{\texttt{compr}}(\omega^j) = \textstyle\sum_{0 \le i < N_{\texttt{lookup}}} \texttt{table\_value}_i(\omega^j)$$

We need to prove the security of the lookup aggregation

6.4 Small Tables

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

For instance, let $\mathbf{S}_1 = S_{1_0}, S_{1_1}$, $\mathbf{S}_2 = S_{2_0}, S_{2_1}$ be two lookup tables with 4 rows. Two lookup expression corresponds to $\mathbf{S}_1, \mathbf{S}_2$. These tables can be located in the separate way:

q_{l_1}	S_{1_0}	S_{1_1}	q_{l_2}	S_{2_0}	S_{2_1}
...	0	1	...	4	5
...	1	1	...	5	6
...	2	1	...	6	7
...	3	0	...	7	8

However, $N_{\texttt{rows}} >> 4$ 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:

q_{l_1}	q_{l_2}	`tag`	S_0	S_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.

6.5 Non-Fixed Lookup Tables

The table $\mathbf{S}$ has not to be fixed. Any columns from $\mathbf{S}$ can be witness columns. This does not change the argument above.

7 Optimizations

WIP

7.1 Batched FRI

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

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

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

See [3] for details.

7.2 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 [**?**], [3] for details.

7.3 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 [**?**] for details. Similar approach is described in [3]. However, the authors of [3] use more values per leaf, that leads to better performance.

7.4 FRI PoW

WIP

8 Redshift Parameters

In this section, we discuss Redshift parameters and their influence on the protocol security and performance.

8.1 Circuit Influence

Let C be a circuit that should be proven. Recall some notations from Section 2.

$N_{\texttt{rows}}$	Number of rows (unpadded to power of two) in C
$N_{\texttt{witness}}$	Number of witness columns
$N_{\texttt{perm}}$	Number of witness columns that are included in the permutation argument
$N_{\texttt{sel}}$	Number of selectors used in the circuit
$N_{\texttt{lookups}}$	Number of lookups

PADDING TO POWER OF TWO

FAKE ROWS

8.2 FRI Parameters

Let $\mathbf{RS}[\mathbb{F}, D, \rho]$ be Reed-Solomon code family. Here $|D| = n = 2^k$, $\rho = 2^{-R}(k, R\mathbb{N})$. This implies degree bound for commiting polynomials $d = 2^{k-R}$. Fix $r \in [1, \log d = n]$ — the number of FRI inner rounds. Let l be a repetition parameter.

Prover: $\mathcal{O}(n)$, Verifier: $\mathcal{O}(\log n)$.

For every $\epsilon \in (0, 1]$, let $J_\epsilon : [0, 1] \to [0, 1]$ be the function

$$J_\epsilon(X) = 1 - \sqrt{1 - X(1 - \epsilon)}$$

USE THE LATEST SOUND-NESS BOUND

Suppose that $\Delta(f, \mathbf{RS}) = \delta > 0$, then soundness error is bounded by:

$$\mathbf{err}(\delta) = \frac{2 \log |D|}{\epsilon^3 |\mathbb{F}|} + \left(1 - \mathtt{min}\{\delta_0, J_\epsilon(J_\epsilon(1 - \rho))\} + \epsilon \log |D|\right)^l$$

8.3 RedShift Parameters

Now we can apply the circuit parameters to FRI commitments.

Let d be the smallest power of two such that $d \geq N_{\texttt{rows}}$. In RedShift, d defines the highest degree of polynomials that can be committed by FRI instance.

CAN WE JUST USE NOTATION $N_{\texttt{rows}}$ FOR PADDED ROWS?

Let ω be d root of unity. Recall that d has a form $d = 2^{\hat{n}}$ and $\hat{n} \geq N_{\texttt{rows}}$.

We use $\mathbf{RS}[\mathbb{F}, D, \rho]$ for FRI commitments, where

- $\mathbb{F}$ is the same field that is used in PLONK arithmetization.
- D is the domain of $n = 2^k = 2^{\hat{n}+R}$ root of unity.
- $\rho = 2^{-R}$ is the parameter that can be adjusted.

Let f be a polynomial used during RedShift proof. To interpolate f, we use the powers of ω. To commit f, we need to build low degree extension $f|_D$.

An additional root of unity is used for permutation polynomials interpolation. Let δ be T root of unity where $T \cdot 2^S + 1 = p$, T odd and $k \leq S$

CHANGE THE NOTATION

δ^i are all distinct quadratic non-residues

IT'S FOR PASTA CURVES

LOOKUP PARAMETERS SHOULD BE HERE

m for lookup

I suppose, we need also fix $\epsilon \in (0, 1]$?! - ϵ-ball is defined for LPC $L_\delta(f)$ is δ-list of f. $\Delta(f, g) < \delta$. $\delta > 0$ - error-bound

"distinguishing" point z

9 Circuit Performance Estimation

Recall and update notations:

n	Number of rows
$N_{\texttt{witness}}$	Number of witness columns ('advice columns')
$N_{\texttt{perm}}$	Number of witness columns that are included in the permutation argument
$N_{\texttt{sel}}$	Number of selectors used in the circuit
$N_{\texttt{const}}$	Number of constant columns
$\mathbf{f}_i$	Witness polynomials, $0 \leq i < N_{\texttt{witness}}$
$\mathbf{f}_{c_i}$	Constant-related polynomials, $0 \leq i < N_{\texttt{const}}$
$\mathbf{gate}_i$	Gate polynomials, $0 \leq i < N_{\texttt{sel}}$
$\sigma(\text{col} : i, \text{row} : j) = (\text{col} : i', \text{row} : j')$	Permutation over the table
H_c	Commitment hash
H_r	Random Oracle hash
l_{H_c}	Number of bits in commitment hash
l_{H_r}	Number of bits in random oracle hash

Public data:

- ???

9.1 Proof Size

Proof contains: ALICE CHECK LOOKUP

- $f_{0,\texttt{comm}}, \ldots, f_{N_{\texttt{witness}}-1,\texttt{comm}}$ - commitments for witness polynomials
- $A'_{\texttt{comm}}, S'_{\texttt{comm}}$ - lookup commitments
- $P_{\texttt{comm}}, Q_{\texttt{comm}}$
- $V_{\texttt{comm}}$ - lookup related
- $T_{0,\texttt{comm}}, ..., T_{N_{\texttt{perm}}-1,\texttt{comm}}$
- Values and paths with size $\log n$:
  - $f_i(y)$ for $i \in [0, N_{\texttt{witness}} - 1]$
  - $P(y), P(y\omega), Q(y), Q(y\omega)$
  - $T_j(y)$ for $j \in [0, N_{\texttt{perm}} - 1]$
  - $A'(y), S'(y), V(y), A'(y\omega^{-1}), V(y\omega)$
  - Gate-depending $f_i(y\omega^\mu)$
- For circuit polynomials: distinguishing point values
- Evaluation proof for the values above (l times):
  - for $i \in [0, r-1]$: // i = 0 makes sense because it's for U polynomial,
    - $m + 1$ values
    - m merkle paths of size $\log n - i$

– in r round: $\log n - r$ values

Firstly, we define the size of basic structures in the proof. Each commitment has size l_{H_c} as a Merkle tree root. Values are taken from the field $\mathbb{F}$ with a bit length l_p.

FRI proofs size depends on FRI parameters. Fix RS code familty $\text{RS}[\mathbb{F}, D, \rho]$, where $|D| = n = 2^k$ and rate $\rho = 2^{-R}$. This implies that degree bound d is 2^{k-R}

Let $J_\epsilon(X) = 1 - \sqrt{1 - X(1-\epsilon)}$

For any $\epsilon \in (0, 1])$, FRI soundness error $\mathbf{err}(\delta)$ is bounded by:

$$\frac{2\log|D|}{\epsilon^3|\mathbb{F}|} + (1 - min\{\delta_0, J_\epsilon(J_\epsilon(1-\rho))\} + \epsilon\log|D|)^l$$

Here l us repetition parameter. The soundness bound was improved in [5] (something around $1/2x$). CHECK IT LATER

Proof size (original):

1. Commitments: $\texttt{comm\_n} = N_{\texttt{witness}} + 2 + N_{\texttt{perm}}$
2. LookUp Commitments: $\texttt{lookup\_comm\_n} = 3$
3. Evaluations: $\texttt{eval\_n} = N_{\texttt{witness}} + 4 + N_{\texttt{perm}} + GATES$
4. Evaluations Merkle paths: $(\log n \cdot |H_c|) \cdot \texttt{eval\_n}$
5. LookUp Evaluations: 5
6. Evaluations proofs: $l \cdot [r \cdot (m+1) \cdot |p| + \sum_{i=0}^{r-1}((m+1) \cdot (\log n - i) \cdot |H_c|) + (\log n - r) \cdot |p|]$

Appendices

A Circuit Example

B Get It All Together

Protocol description with optimizations, zk, etc.

References

1. Gabizon A., Williamson Z. J., Ciobotaru O. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953. 2019. https://ia.cr/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. Kattis A., Panarin K., Vlasov A. RedShift: Transparent SNARKs from List Polynomial Commitment IOPs. Cryptology ePrint Archive, Report 2019/1400. 2019. https://ia.cr/2019/1400.
4. Ben-Sasson E., Chiesa A., Spooner N. Interactive Oracle Proofs. Cryptology ePrint Archive, Report 2016/116. 2016. https://ia.cr/2016/116.
5. Ben-Sasson E., Carmon D., Ishai Y. et al. Proximity Gaps for Reed-Solomon Codes. Cryptology ePrint Archive, Report 2020/654. 2020. https://ia.cr/2020/654.
6. Fast Reed-Solomon interactive oracle proofs of proximity / E. Ben-Sasson, I. Bentov, Y. Horesh et al. // 45th international colloquium on automata, languages, and programming (icalp 2018) / Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.
7. Gabizon A., Williamson Z. J. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315. 2020. https://ia.cr/2020/315.

8. PLONKish Arithmetization - The halo2 book. https://zcash.github.io/halo2/concepts/arithmetization.html.

9. Lookup argument - The halo2 book. https://zcash.github.io/halo2/design/proving-system/lookup.html.