PlonKup: Reconciling PlonK with plookup

Luke Pearson*[1], Joshua Fitzgerald[2], Héctor Masip[4],
Marta Bellés-Muñoz[3,5], and Jose Luis Muñoz-Tapia[4]

[1] *Polychain Capital*
[2] *Anoma*
[3] *Dusk Network*
[4] *Universitat Politècnica de Catalunya*
[5] *Pompeu Fabra University*

January, 2022

Abstract

In 2019, Gabizon, Williamson, and Ciobotaru introduced PlonK – a fast and flexible ZK-SNARK with an updatable and universal structured reference string. PlonK uses a grand product argument to check permutations of wire values, and exploits convenient interactions between multiplicative subgroups and Lagrange bases. The following year, Gabizon and Williamson used similar techniques to develop plookup– a ZK-SNARK that can verify that each element from a list of queries can be found in a public lookup table. In this paper, we present PlonKup, a fully succinct ZK-SNARK that integrates the ideas from plookup into PlonK in an efficient way.

Contents

*This work was commenced whilst the first author was at Dusk Network

1 Introduction

The first practical *zero-knowledge succinct noninteractive arguments of knowledge (ZK-SNARKs)* with constant verification and communication complexity [GGPR13, PHGR13, Gro16], needed an *structured reference string (SRS)* per each specific individual relation. The construction of the SRS requires a set of secret values that if ever gets exposed, the whole scheme gets compromised. To reduce risk, the secret values are usually computed collaboratively using *multi-party computation (MPC)* protocols that guarantee that, if at least one of the participants is honest, the final parameters are secure [BGM17]. However, each specific individual relation being proved with ZK-SNARKs required a different SRS, and hence, every change in the construction required a new MPC. Recently, new schemes like [MBKM19, CHM+20, GKM+18, GWC19], were constructed with an *updatable* SRS that could be upgraded at any point in time. Moreover, the SRS from these protocols is also *universal*, meaning that it can be used for any circuit of some bounded size. In this paper, we generalize $\mathcal{P}$lon$\mathcal{K}$ [GWC19], a universal and updatable ZK-SNARK for general arithmetic circuit satisfiability that was presented by Gabizon, Williamson and Cioboutaru in 2019.

$\mathcal{P}$lon$\mathcal{K}$ represents circuits using fan-in 2 and fan-out 1 gates that can be expressed as constraints of the form

$$q_{Li} \cdot a_i + q_{Ri} \cdot b_i + q_{Oi} \cdot c_i + q_{Mi} \cdot a_i b_i + q_{Ci} = 0. \tag{1}$$

The coefficients $q_{Li}, q_{Ri}, q_{Oi}, q_{Mi}$ and q_{Ci} are *selectors* that represent the gate's operation (addition, multiplication, or constant assignment), and the elements a_i, b_i, and c_i are wires whose values should satisfy the gate's operation, i.e. the corresponding constraint. Typically, a_i and b_i are the left and the right wire, respectively, and c_i represents the output wire. This way, we can think of an arithmetic circuit as a set of gates represented by the set of selector vectors $\{\mathbf{q_L} = (q_{Li})_{i=1}^n, \mathbf{q_R} = (q_{Ri})_{i=1}^n, \mathbf{q_O} = (q_{Oi})_{i=1}^n, \mathbf{q_M} = (q_{Mi})_{i=1}^n, \mathbf{q_C} = (q_{Ci})_{i=1}^n\}$, together with some function that connects the wires from one gate to another. At a very high level, the idea behind $\mathcal{P}$lon$\mathcal{K}$ is to use Kate polynomial commitments [KZG10] to prove that each element i of the vectors $\boldsymbol{a} = (a_i)_{i=1}^n$, $\boldsymbol{b} = (b_i)_{i=1}^n$ and $\boldsymbol{c} = (c_i)_{i=1}^n$ satisfy the i-th constraint, and also that the wires are connected to the right gates. This second condition is enforced by a *grand product argument* constructed using additional vectors representing permutations of the variables.

The restricted form of the constraints in $\mathcal{P}$lon$\mathcal{K}$, makes the implementation of some common functions very expensive. For instance, manipulation of bits, such as XOR or AND operation between bit strings, or common hash functions such as AES-128 or SHA-256, which are constructed from the repeated use of some simple computation units that are heavy on bit manipulations, are very inefficient [GW20]. There have been different ways to tackle this problem. In particular, the authors of **plookup** [GW20] propose a way to treat some of the "frequently used" computation units as precomputed lookup tables. The idea is that the whole original computations can be converted into a more general type of circuit which has, besides addition, multiplication and constant gates, a fourth type of gates, called *lookup gates.*

For now, we consider a lookup gate as a fan-in2 and fan-out 1 gate, whose operation is described in a public table. In this case, the table must have three columns, one per input/output, and each row should describe the relation between the output and the pair of inputs. When integrating this type of gate in a circuit, the prover will prove that her witness values exist in the table.

Our contributions. In this paper, we generalize $\mathcal{P}$lon$\mathcal{K}$ based on the ideas of **plookup**, and present $\mathcal{P}$lon$\mathcal{K}$up, a fully-succinct ZK-SNARK that allows proving gates of the form of Equation (1), as well as lookup gates. As a result, $\mathcal{P}$lon$\mathcal{K}$up can be easily used as an extension of $\mathcal{P}$lon$\mathcal{K}$, enhancing the performance of the original protocol in many useful cases. We assume the reader is familiar with Kate polynomial commitments [KZG10], $\mathcal{P}$lon$\mathcal{K}$ [GWC19] and plookup [GW20] protocols.

2 Terminology and Notation

We follow a notation similar to [GWC19] and [GW20].

Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_t$ groups of prime order r, and e an efficiently computable non-degenerate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$. We choose generators for these groups $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, such that $e(g_1, g_2)$ generates $\mathbb{G}_t$. We use the addition notation for $\mathbb{G}_1$ and $\mathbb{G}_2$, and the shorthand notation $[x]_1 := x \cdot g_1$ and $[x]_2 := x \cdot g_2$. The operations on $\mathbb{G}_t$ are written multiplicatively.

We consider $\mathbb{F}$ a finite field of prime order p. For a given integer n, we denote by $[n]$ the set of integers $\{1, ..., n\}$. We explicitly define the multiplicative subgroup $H \subset \mathbb{F}^*$ as the subgroup containing the n-th roots of unity in $\mathbb{F}$, where ω is a primitive n-th root of unity and a generator of H. That is,

$$H = \{\omega, \ldots, \omega^{n-1}, \omega^n = 1\}.$$

We denote by $\mathbb{F}_{<n}[X]$ the set of univariate polynomials over $\mathbb{F}$ of degree strictly smaller than n. For a polynomial $f(x) \in \mathbb{F}_{<n}[X]$ and $i \in [n]$, we sometimes denote $f_i := f(\omega^i)$. For a vector $\boldsymbol{f} \in \mathbb{F}^n$, we also denote by $f(x)$ the polynomial in $\mathbb{F}_{<n}[X]$ with $f(\omega^i) = f_i$.

The Lagrange polynomial $L_i(X) \in \mathbb{F}_{<n}[X]$ for $i \in [n]$ has the form

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

Thus, the zero polynomial $Z_H(X) \in \mathbb{F}_{<n+1}[X]$ is defined as

$$Z_H(X) = (X - \omega) \cdots (X - \omega^{n-1})(X - \omega^n) = X^n - 1.$$

Let $\mathcal{T} \in \mathbb{F}^{n \times 3}$ be a publicly known table of 3 columns and n rows. We denote by $\mathcal{T}_{i,j}$, the element of the table placed in column i and row j, for $i \in \{1, 2, 3\}$ and $j \in \{1, \ldots, n\}$. To compress the rows of the table, we define the vector $\boldsymbol{t} = (t_1, \ldots, t_n) \in \mathbb{F}^n$, where

$$t_i = \mathcal{T}_{1,i} + \zeta\mathcal{T}_{2,i} + \zeta^2\mathcal{T}_{3,i},$$

for all $i \in [n]$ and where $\zeta \in \mathbb{F}$ is a random element obtained from the verifier.

Let n and d be two integers, and let $f = \{f_i\}_{i=1}^n \in \mathbb{F}^n$ and $t = \{t_i\}_{i=1}^d \in \mathbb{F}^d$ be two vectors such that $\{f_i\}_{i=1}^n \subset \{t_i\}_{i=1}^d$. We say that f is sorted by t, when values appear in f the same order as they do it in t.

We assume that the number of gates in a circuit, that is, the total sum of arithmetic and lookup gates, is no more than n, and that the length of the vector $\boldsymbol{t}$ (that coincides with the number of rows in the table $\mathcal{T}$) is, as well, no more than n.

3 Our Protocol

3.1 General Overview

In this Section, we present the main ideas in our protocol that differ from the $\mathcal{P}$lon$\mathcal{K}$ and plookup protocols.

Selecting Lookup Gates

For our needs, the lookup and $\mathcal{P}lon\mathcal{K}$ portions of the protocol needed to be able to refer to the same variables, so that the output variable of a regular arithmetic gate could be an input to a plookup gate and vice-versa. For this reason, we use the same wires for both arithmetic and lookup gates. Then, we use a *selector* q_K to select only lookup gates when necessary.

Constructing the Compressed Public Lookup Table

Given a public table $\mathcal{T} \in \mathbb{F}^{n\times 3}$, we compress the columns of $\mathcal{T}$ into a single vector $\boldsymbol{t} = (t_1, \dots, t_n)$ by sampling a random $\gamma \in \mathbb{F}$ and computing

$$t_i = \mathcal{T}_{1,i} + \zeta \cdot \mathcal{T}_{2,i} + \zeta^2 \cdot \mathcal{T}_{3,i}.$$

If the circuit has m wires and T has n rows, for $m < n$, the wires should be padded with $m - n$ dummy values and the selectors padded with $m - n$ zeroes until the lengths match. If $n < m$, then the table should be padded with $n - m$ copies of one of its elements.

Constructing the Query Wire

The query vector $\boldsymbol{f} = (f_1, \dots, f_n)$ is constructed from the wires a, b, and c. When the lookup selector $q_{K\,i}$ is 0, f_i takes on a dummy value from the public table, and when $q_{K\,i}$ is 1, then f_i becomes the compression of a_i, b_i, and c_i. That is, we define f_i as

$$f_i = \begin{cases} a(\omega^i) + \zeta \cdot b(\omega^i) + \zeta^2 \cdot c(\omega^i), & \text{if the } i\text{-th gate is a lookup gate,} \\ \mathcal{T}_{1,n} + \zeta \cdot \mathcal{T}_{2,n} + \zeta^2 \cdot \mathcal{T}_{3,n}, & \text{otherwise.} \end{cases}$$

Alternating Method for the Sorted Vector

Let $\boldsymbol{f} = (f_0, ..., f_{n-1})$ be a query vector and $\boldsymbol{t} = (t_0, ..., t_{n-1})$ a table vector, both of length n. Like in plookup, the prover has to compute the vector $\boldsymbol{s} = (\boldsymbol{f}, \boldsymbol{t})$ and sort $\boldsymbol{s}$ with respect to $\boldsymbol{t}$.

Consider the randomized difference sets for $\boldsymbol{t}$ and $\boldsymbol{s}$. That is, $\Delta t = (\Delta t_0, ..., \Delta t_{n-1})$, and $\Delta s = (\Delta s_0, ..., \Delta s_{n-1})$, such that

$$\Delta t_i = \begin{cases} t_i + \delta t_{i+1} & \text{for } i \in \{0, ..., n-2\}, \\ t_i + \delta t_0, & \text{for } i = n-1, \end{cases} \qquad \Delta s_i = \begin{cases} s_i + \delta s_{i+1} & \text{for } i \in \{0, ..., n-2\}, \\ s_i + \delta s_0, & \text{for } i = n-1, \end{cases}$$

for some randomly chosen δ.

The prover's algorithm from plookup divides the $\boldsymbol{s}$ vector into lower and upper halves $\boldsymbol{h_1}$ and $\boldsymbol{h_2}$, so that $\boldsymbol{h_1} = (s_0, s_1, s_2, ..., s_{n-1})$ and $\boldsymbol{h_2} = (s_n, s_{n+1}, s_{n+2}, ..., s_{2n-1})$. Then, the expression $\Delta s_i = s_i + \delta s_{i+1}$ can be written as $h_{1i} + \delta h_{1i+1}$, and the expression $\Delta s_{n+i} = s_{n+i} + \delta s_{n+i+1}$ can be written as $h_{2i} + \delta h_{2i+1}$.

As a result, their permutation polynomial expression is of the form

$$Z(X\omega) = Z(X) \frac{(1+\delta)(\epsilon + f(X))(\epsilon(1+\delta) + t(X) + \delta t(X\omega))}{(\epsilon(1+\delta) + h_1(X) + \delta h_1(X\omega)(\epsilon(1+\delta) + h_2(X) + \delta h_2(X\omega))}.$$

The drawback of this method is that the verifier must check that $\boldsymbol{h_1}$ and $\boldsymbol{h_2}$ overlap, i.e., the verifier must check that

$$L_{n-1}(X)h_1(X) - L_0(X)h_2(X) = 0.$$

In our protocol, we have the prover divide $\boldsymbol{s}$ into alternating halves $\boldsymbol{h_1}$ and $\boldsymbol{h_2}$ so that $\boldsymbol{h_1} = (s_0, s_2, s_4, ..., s_{2n-2})$ and $\boldsymbol{h_2} = (s_1, s_3, s_5, ..., s_{2n-1})$. Now, the expression $\Delta s_{2i} = s_{2i} + \delta s_{2i+1}$ can be written as $h_{1i} + \delta h_{2i}$, and the expression $\Delta s_{2i+1} = s_{2i+1} + \delta s_{2i+2}$ can be written as $h_{2i} + \delta h_{1i+1}$. This way, our permutation polynomial expression becomes

$$Z(X\omega) = Z(X)\frac{(1+\delta)(\epsilon + f(X))(\epsilon(1+\delta) + t(X) + \delta t(X\omega))}{(\epsilon(1+\delta) + h_1(X) + \delta h_2(X)(\epsilon(1+\delta) + h_2(X) + \delta h_1(X\omega))},$$

and no extra check from the verifier is needed.

Finalising a Mega Permutation argument

Both $\mathcal{P}$lon$\mathcal{K}$ and **plookup** utilise a similar methodology for converting a computer program into a SNARK. $\mathcal{P}$lon$\mathcal{K}$ is especially well-known for the introducing the grand product argument and evaluations over a multiplicative subgroup, amongst other novelties. The permutation in the **plookup** protocol uses a polynomial s, which is a concatenated and sorted version of the table and query polynomials, to check that certain relations hold. The sorting algorithm is performed after both the table and query columns are compressed into single vectors of field elements with ascending powers of a challenge scalar. As a result the complexity class of applicable algorithms, which can sort with respect to a list of field elements, are not akin to the complexity class of the rest of the prover operations. We modify and augment the above permutation polynomial by introducing a sorted version of the preprocessed table, namely t'. This ultimately increases the efficiency, as it allows for the construction of the sorted polynomial, s, with algorithms of complexity $O(n\mathsf{log}n)$. The prover can enforce constraints from the product permutation by appending $\frac{t'_i+\theta}{t_i+\theta}$. Our mega-permutation is defined as

$$Z(X\omega) = Z(X)\frac{(1+\delta)(\varepsilon + (X))(\varepsilon(1+\delta) + t'(X) + \delta t'(X\omega))(\theta + t'(X))}{(\varepsilon(1+\delta) + h_1(X) + \delta h_2(X)(\varepsilon(1+\delta) + h_2(X) + \delta h_1(X\omega)(\theta + t(X))}.$$

3.2 Polynomials Defining a Circuit

In the following list, we define the polynomials that define a specific circuit with lookup gates.

- The table polynomial $t(X) \in \mathbb{F}_{<n}[X]$ representing the table $\mathcal{T}$:

$$t(X) = \sum_{i=1}^{n} t_i L_i(X).$$

- The selector polynomials $q_M(X), q_L(X), q_R(X), q_O(X), q_C(X) \in \mathbb{F}_{<n}[X]$, which define the arithmetization of an arithmetic circuit.

- An additional selector $q_K(X) \in \mathbb{F}_{<n}[X]$, which activates the lookup gates. More specifically:

$$q_K(\omega^i) = q_{K_i} = \begin{cases} 1, & \text{if the } i\text{-th gate is a } \textbf{lookup} \text{ gate,} \\ 0, & \text{otherwise.} \end{cases}$$

- The identity permutation polynomials $S_{\mathrm{ID}1}(X) = X, S_{\mathrm{ID}2}(X) = k_1 \cdot X, S_{\mathrm{ID}3}(X) = k_2 \cdot X$ applied to $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, with constants $k_1, k_2 \in \mathbb{F}$ chosen such that $H, k_1 \cdot H, k_2 \cdot H$ are distinct cosets in $\mathbb{F}^*$, and thus consist of $3n$ distinct elements.
- Let us denote $H' := H \cup k_1 \cdot H \cup k_2 \cdot H$. Let $\sigma\colon [3n] \to [3n]$ be a permutation. Now, identify $[3n]$ with H' via $i \to \omega^i, n+i \to k_1 \cdot \omega^i, 2n+i \to k_2 \cdot \omega^i$. Finally, define σ^* below to denote the mapping from $[3n]$ to H' derived from applying σ and then this injective mapping into H'. We encode σ^* by the three copy permutation polynomials $S_{\sigma_1}(X), S_{\sigma_2}(X), S_{\sigma_3}(X) \in \mathbb{F}_{<n}[X]$:

$$S_{\sigma_1}(X) = \sum_{i=1}^{n} \sigma^*(i) L_i(X), \quad S_{\sigma_2}(X) = \sum_{i=1}^{n} \sigma^*(n+i) L_i(X), \quad S_{\sigma_3}(X) = \sum_{i=1}^{n} \sigma^*(2n+i) L_i(X).$$

3.3 The SNARK Proof Relation

Given $\ell < n$ and fixed values for the above polynomials, we wish to prove statements of knowledge for the relation $\mathcal{R} \subset \mathbb{F}^{\ell} \times \mathbb{F}^{3n-\ell}$ containing all pairs $x = (w_i)_{i \in [\ell]}$, $w = (w_i)_{i=\ell+1}^{3n}$ such that:

1. For all $i \in [n]$:

$$\begin{aligned} &q_{Mi} w_i w_{n+i} + q_{Li} w_i + q_{Ri} w_{n+i} + q_{Oi} w_{2n+i} + q_{Ci} = 0, \\ &q_{Ki}\big(w_i + \zeta w_{n+i} + \zeta^2 w_{2n+i} - f_i\big) = 0, \\ &f_i \in \{t_1, t_2, \dots, t_n\}. \end{aligned}$$

2. For all $i \in [3n]$:

$$w_i = w_{\sigma(i)}.$$

3.4 The $\mathcal{P}\mathfrak{lon}\mathcal{K}\mathfrak{up}$ Protocol

In the following protocol we use Hash to refer to a hash function, where $\mathsf{Hash}\colon \{0,1\}^* \to \{0,1\}^{\ell}$ is an efficiently computable hash function that takes arbitrary length inputs and returns ℓ-bit outputs.

We describe the protocol below as a non-interactive protocol using the Fiat-Shamir heuristic. For this purpose we always denote by $\mathsf{transcript}$ the concatenation of the common preprocessed input, public input, and the proof elements written by the prover up to a certain point in time. We use $\mathsf{transcript}$ for obtaining random challenges via Fiat-Shamir.

Common preprocessed input:

$$n, \left([1]_1, [x]_1, \dots, \left[x^{n+5}\right]_1\right), (\mathcal{T}_{1,i}, \mathcal{T}_{2,i}, \mathcal{T}_{3,i})_{i \in [n]}, \left(q_{M_i}, q_{L_i}, q_{R_i}, q_{O_i}, q_{C_i}, q_{K_i}\right)_{i \in [n]}, \sigma^*,$$

$$\begin{aligned} &\mathcal{T}_1(X) = \textstyle\sum_{i=1}^n \mathcal{T}_{1,i} L_i(X), && \mathcal{T}_2(X) = \textstyle\sum_{i=1}^n \mathcal{T}_{2,i} L_i(X), && \mathcal{T}_3(X) = \textstyle\sum_{i=1}^n \mathcal{T}_{3,i} L_i(X), \\ &q_M(X) = \textstyle\sum_{i=1}^n q_{M_i} L_i(X), && q_L(X) = \textstyle\sum_{i=1}^n q_{L_i} L_i(X), && q_R(X) = \textstyle\sum_{i=1}^n q_{R_i} L_i(X), \\ &q_O(X) = \textstyle\sum_{i=1}^n q_{O_i} L_i(X), && q_C(X) = \textstyle\sum_{i=1}^n q_{C_i} L_i(X), && q_K(X) = \textstyle\sum_{i=1}^n q_{K_i} L_i(X), \\ &S_{\sigma_1}(X) = \textstyle\sum_{i=1}^n \sigma^*(i) L_i(X), && S_{\sigma_2}(X) = \textstyle\sum_{i=1}^n \sigma^*(n+i) L_i(X), && S_{\sigma_3}(X) = \textstyle\sum_{i=1}^n \sigma^*(2n+i) L_i(X). \end{aligned}$$

Public input: The number of public inputs ℓ and the public input $x = (w_i)_{i \in [\ell]}$.

3.4.1 Prover Algorithm

Prover input: The pair $(x, w) = (w_i)_{i \in [3n]}$ that satisfies the circuit $\mathscr{C}$.

Round 1

1. Generate random blinding scalars $b_1, \ldots, b_6 \in \mathbb{F}$.
2. Compute the wire polynomials $a(X), b(X), c(X) \in \mathbb{F}_{<n+2}[x]$:

$$\begin{aligned} a(X) &= (b_1 X + b_2) Z_H(X) + \sum_{i=1}^{n} w_i L_i(X), \\ b(X) &= (b_3 X + b_4) Z_H(X) + \sum_{i=1}^{n} w_{n+i} L_i(X), \\ c(X) &= (b_5 X + b_6) Z_H(X) + \sum_{i=1}^{n} w_{2n+i} L_i(X). \end{aligned}$$

3. Compute $[a(x)]_1$, $[b(x)]_1$, $[c(x)]_1$.

The first output of $\mathcal{P}$ is $([a(x)]_1, [b(x)]_1, [c(x)]_1)$.

Round 2

1. Compute the compression factor $\zeta \in \mathbb{F}_p$:

$$\zeta = \mathsf{Hash}(\mathsf{transcript}).$$

2. Compute the query vector $\boldsymbol{f} = (f_1, \ldots, f_n)$ and the table vector $\boldsymbol{t} = (t_1, \ldots, t_n)$:

$$\begin{aligned} f_i &= \begin{cases} a(\omega^i) + \zeta \cdot b(\omega^i) + \zeta^2 \cdot c(\omega^i), & \text{if the } i\text{-th gate is a } \textit{lookup} \text{ gate}, \\ \mathcal{T}_{1,n} + \zeta \cdot \mathcal{T}_{2,n} + \zeta^2 \cdot \mathcal{T}_{3,n}, & \text{otherwise.} \end{cases}, \\ t_i &= \mathcal{T}_{1,i} + \zeta \cdot \mathcal{T}_{2,i} + \zeta^2 \cdot \mathcal{T}_{3,i}. \end{aligned}$$

3. Compute the sorted version of the table vector $\boldsymbol{t}$, denoted $\boldsymbol{t'} = (t'_1, \ldots, t'_n)$.
4. Generate random blinding scalars $b_7, \ldots, b_{13} \in \mathbb{F}$.
5. Compute the query polynomial $f(X) \in \mathbb{F}_{<n+2}[x]$, the table polynomial $t(X) \in \mathbb{F}_{<n}[X]$ and the sorted table polynomial $t'(X) \in \mathbb{F}_{<n}[X]$:

$$\begin{aligned} f(X) &= (b_7 X + b_8) Z_H(X) + \sum_{i=1}^{n} f_i L_i(X), \\ t(X) &= \mathcal{T}_1(X) + \zeta \mathcal{T}_2(X) + \zeta^2 \mathcal{T}_3(X), \\ t'(X) &= \sum_{i=1}^{n} t'_i L_i(X). \end{aligned}$$

6. Let $\boldsymbol{s} \in \mathbb{F}^{2n}$ be the vector that is $(\boldsymbol{f}, \boldsymbol{t'})$ sorted by $\boldsymbol{t'}$. We represent $\boldsymbol{s}$ by the vectors $\boldsymbol{h_1}, \boldsymbol{h_2} \in \mathbb{F}^n$ as follows:

$$\begin{aligned} \boldsymbol{h_1} &= (s_1, s_3, \dots, s_{2n-1}), \\ \boldsymbol{h_2} &= (s_2, s_4, \dots, s_{2n}). \end{aligned}$$

7. Compute the polynomials $h_1(X) \in \mathbb{F}_{<n+3}[X]$, and $h_2(X) \in \mathbb{F}_{<n+2}[X]$

$$\begin{aligned} h_1(X) &= (b_9 X^2 + b_{10} X + b_{11}) Z_H(X) + \sum_{i=1}^{n} s_{2i-1} L_i(X), \\ h_2(X) &= (b_{12} X + b_{13}) Z_H(X) + \sum_{i=1}^{n} s_{2i} L_i(X). \end{aligned}$$

8. Compute $[f(x)]_1$, $[t'(x)]_1$, $[h_1(x)]_1$ and $[h_2(x)]_1$.

The second output of $\mathcal{P}$ is $([f(x)]_1, [t'(x)]_1, [h_1(x)]_1, [h_2(x)]_1)$.

Round 3

1. Compute the permutation challenges $\beta, \gamma, \delta, \varepsilon, \theta \in \mathbb{F}$:

$$\begin{aligned} \beta &= \mathsf{Hash}(\mathsf{transcript} \mid\mid 1), \quad & \gamma &= \mathsf{Hash}(\mathsf{transcript} \mid\mid 2), \\ \delta &= \mathsf{Hash}(\mathsf{transcript} \mid\mid 3), \quad & \varepsilon &= \mathsf{Hash}(\mathsf{transcript} \mid\mid 4), \\ \theta &= \mathsf{Hash}(\mathsf{transcript} \mid\mid 5). \end{aligned}$$

2. Generate random blinding scalars $b_{14}, \dots, b_{19} \in \mathbb{F}$.
3. Compute the $\mathcal{P}\mathfrak{lon}\mathcal{K}$ permutation polynomial $z_1(X) \in \mathbb{F}_{<n+3}[x]$:

$$\begin{aligned} z_1(X) &= (b_{14} X^2 + b_{15} X + b_{16}) Z_H(X) + L_1(X) \\ &+ \sum_{i=1}^{n-1} \left(L_{i+1}(X) \prod_{j=1}^{i} \frac{(w_j + \beta\omega^{j-1} + \gamma)(w_{n+j} + \beta k_1 \omega^{j-1} + \gamma)(w_{2n+j} + \beta k_2 \omega^{j-1} + \gamma)}{(w_j + \beta\sigma^*(j) + \gamma)(w_{n+j} + \beta\sigma^*(n+j) + \gamma)(w_{2n+j} + \beta\sigma^*(2n+j) + \gamma)} \right). \end{aligned}$$

4. Compute the mega-permutation polynomial $z_2(X) \in \mathbb{F}_{<n+3}[x]$, which results from the product of the Plookup and the sorted table permutation polynomials:

$$\begin{aligned} z_2(X) &= (b_{17} X^2 + b_{18} X + b_{19}) Z_H(X) + L_1(X) \\ &+ \sum_{i=1}^{n-1} \left(L_{i+1}(X) \prod_{j=1}^{i} \frac{(1+\delta)(\varepsilon + f_j)(\varepsilon(1+\delta) + t'_j + \delta t'_{j+1})(\theta + t'_j)}{(\varepsilon(1+\delta) + s_{2j-1} + \delta s_{2j})(\varepsilon(1+\delta) + s_{2j} + \delta s_{2j+1})(\theta + t_j)} \right). \end{aligned}$$

5. Compute $[z_1(x)]_1$, $[z_2(x)]_1$.

The third output of $\mathcal{P}$ is $([z_1(x)]_1, [z_2(x)]_1)$.

Round 4

1. Compute the quotient challenges $\alpha \in \mathbb{F}$:

$$\alpha = \mathsf{Hash}(\mathsf{transcript}).$$

2. Compute the quotient polynomial $q(X) \in \mathbb{F}_{<3n+5}[x]$:

$$q(X) = \frac{1}{Z_H(X)} \left(\begin{array}{l} (a(X)b(X)q_M(X) + a(X)q_L(X) + b(X)q_R(X) + c(X)q_O(X) + \mathrm{PI}(X) + q_C(X)) \\ +(a(X) + \beta X + \gamma)(b(X) + \beta k_1 X + \gamma)(c(X) + \beta k_2 X + \gamma)z_1(X)\alpha \\ -(a(X) + \beta S_{\sigma_1}(X) + \gamma)(b(X) + \beta S_{\sigma_2}(X) + \gamma)(c(X) + \beta S_{\sigma_3}(X) + \gamma)z_1(X\omega)\alpha \\ +(z_1(X) - 1)L_1(X)\alpha^2 \\ +q_K(X)(a(X) + \zeta b(X) + \zeta^2 c(X) - f(X))\alpha^3 \\ +z_2(X)(1+\delta)(\varepsilon + f(X))(\varepsilon(1+\delta) + t'(X) + \delta t'(X\omega)(\theta + t'(X)))\alpha^4 \\ -z_2(X\omega)(\varepsilon(1+\delta) + h_1(X) + \delta h_2(X))(\varepsilon(1+\delta) + h_2(X) + \delta h_1(X\omega))(\theta + t(X)))\alpha^4 \\ +(z_2(X) - 1)L_1(X)\alpha^5. \end{array} \right.$$

3. Split $q(X)$ into three polynomials $q_{\mathrm{low}}(X)$, $q_{\mathrm{mid}}(X)$, $q_{\mathrm{high}}(X)$ of degree at most $n+1$ such that:

$$q(X) = q_{\mathrm{low}}(X) + X^{n+2}q_{\mathrm{mid}}(X) + X^{2n+4}q_{\mathrm{high}}(X).$$

4. Compute $[q_{\mathrm{low}}(x)]_1, [q_{\mathrm{mid}}(x)]_1, [q_{\mathrm{high}}(x)]$.

The fourth output of $\mathcal{P}$ is $([q_{\mathrm{low}}(x)]_1, [q_{\mathrm{mid}}(x)]_1, [q_{\mathrm{high}}(x)])$.

Round 5

1. Compute the evaluation challenge $\mathfrak{z} \in \mathbb{F}$:

$$\mathfrak{z} = \mathsf{Hash}(\mathsf{transcript}).$$

2. Compute the opening evaluations:

$$a(\mathfrak{z}), \quad b(\mathfrak{z}), \quad c(\mathfrak{z}), \quad S_{\sigma_1}(\mathfrak{z}), \quad S_{\sigma_2}(\mathfrak{z}), \quad f(\mathfrak{z}), \quad t'(\mathfrak{z}), \quad h_2(\mathfrak{z}), \quad t(\mathfrak{z}),$$
$$z_1(\mathfrak{z}\omega), \quad t'(\omega\mathfrak{z}), \quad z_2(\omega\mathfrak{z}), \quad h_1(\omega\mathfrak{z}).$$

The fifth output of $\mathcal{P}$ is $(a(\mathfrak{z}), b(\mathfrak{z}), c(\mathfrak{z}), S_{\sigma_1}(\mathfrak{z}), S_{\sigma_2}(\mathfrak{z}), f(\mathfrak{z}), t'(\mathfrak{z}), h_2(\mathfrak{z}), t(\mathfrak{z}), z_1(\mathfrak{z}\omega), t'(\omega\mathfrak{z}), z_2(\omega\mathfrak{z}), h_1(\omega\mathfrak{z}))$.

Round 6

1. Compute the opening challenge $v \in \mathbb{F}$:

$$v = \mathsf{Hash}(\mathsf{transcript}).$$

2. Compute linearization polynomial $r(X) \in \mathbb{F}_{<n+3}[x]$:

$$\begin{aligned} r(X) = {} & a(\mathfrak{z})b(\mathfrak{z})q_M(X) + a(\mathfrak{z})q_L(X) + b(\mathfrak{z})q_R(X) + c(\mathfrak{z})q_O(X) + \mathrm{PI}(\mathfrak{z}) + q_C(X) \\ & + \alpha[(a(\mathfrak{z}) + \beta\mathfrak{z} + \gamma)(b(\mathfrak{z}) + \beta k_1\mathfrak{z} + \gamma)(c(\mathfrak{z}) + \beta k_2\mathfrak{z} + \gamma)z_1(X) \\ & - (a(\mathfrak{z}) + \beta S_{\sigma_1}(\mathfrak{z}) + \gamma)(b(\mathfrak{z}) + \beta S_{\sigma_2}(\mathfrak{z}) + \gamma)(c(\mathfrak{z}) + \beta S_{\sigma_3}(X) + \gamma)z_1(\mathfrak{z}\omega)] \\ & + \alpha^2(z_1(X) - 1)L_1(\mathfrak{z}) \\ & + \alpha^3 q_K(X)(a(\mathfrak{z}) + \zeta b(\mathfrak{z}) + \zeta^2 c(\mathfrak{z}) - f(\mathfrak{z})) \\ & + \alpha^4[z_2(X)(1+\delta)(\varepsilon + f(\mathfrak{z}))(\varepsilon(1+\delta) + t'(\mathfrak{z}) + \delta t'(\mathfrak{z}\omega))(\theta + t'(\mathfrak{z})) \\ & - z_2(\mathfrak{z}\omega)(\varepsilon(1+\delta) + h_1(X) + \delta h_2(\mathfrak{z}))(\varepsilon(1+\delta) + h_2(\mathfrak{z}) + \delta h_1(\mathfrak{z}\omega)(\theta + t(\mathfrak{z}))] \\ & + \alpha^5(z_2(X) - 1)L_1(\mathfrak{z}) \\ & - Z_H(\mathfrak{z})(q_{\mathrm{low}}(X) + \mathfrak{z}^{n+2}q_{\mathrm{mid}}(X) + \mathfrak{z}^{2n+4}q_{\mathrm{high}}(X)). \end{aligned}$$

3. Compute the opening proof polynomial $W_{\mathfrak{z}}(X) \in \mathbb{F}_{<n+2}[X]$:

$$W_{\mathfrak{z}}(X) = \frac{1}{X - \mathfrak{z}} \left(\begin{array}{l} r(X) \\ +v(a(X) - a(\mathfrak{z})) \\ +v^2(b(X) - b(\mathfrak{z})) \\ +v^3(c(X) - c(\mathfrak{z})) \\ +v^4(S_{\sigma 1}(X) - S_{\sigma_1}(\mathfrak{z})) \\ +v^5(S_{\sigma 2}(X) - S_{\sigma_2}(\mathfrak{z})) \\ +v^6(f(X) - f(\mathfrak{z})) \\ +v^7(t'(X) - t'(\mathfrak{z})) \\ +v^8(h_2(X) - h_2(\mathfrak{z})) \\ +v^9(t(X) - t(\mathfrak{z})). \end{array} \right)$$

4. Compute the opening proof polynomial $W_{\mathfrak{z}\omega}(X) \in \mathbb{F}_{<n+2}[X]$:

$$W_{\mathfrak{z}\omega}(X) = \frac{1}{X - \mathfrak{z}\omega} \left(\begin{array}{l} z_1(X) - z_1(\mathfrak{z}\omega) \\ +v(t'(X) - t'(\mathfrak{z}\omega)) \\ +v^2(z_2(X) - z_2(\mathfrak{z}\omega)) \\ +v^3(h_1(X) - h_1(\mathfrak{z}\omega)) \end{array} \right).$$

5. Compute $[W_{\mathfrak{z}}(x)]_1$ and $[W_{\mathfrak{z}\omega}(x)]_1$.

The sixth output of $\mathcal{P}$ is $([W_{\mathfrak{z}}(x)]_1, [W_{\mathfrak{z}\omega}(x)]_1)$.

The complete proof is:

$$\pi_{\mathcal{P}\mathfrak{lon}\mathcal{K}\mathfrak{up}} = \left(\begin{array}{c} [a(x)]_1, [b(x)]_1, [c(x)]_1, [f(x)]_1, [t'(x)]_1, [h_1(x)]_1, [h_2(x)]_1, [z_1(x)]_1, [z_2(x)]_1, \\ [q_{\mathrm{low}}(x)]_1, [q_{\mathrm{mid}}(x)]_1, [q_{\mathrm{high}}(x)], [W_{\mathfrak{z}}(x)]_1, [W_{\mathfrak{z}\omega}(x)]_1, \\ a(\mathfrak{z}), b(\mathfrak{z}), c(\mathfrak{z}), S_{\sigma_1}(\mathfrak{z}), S_{\sigma_2}(\mathfrak{z}), f(\mathfrak{z}), t'(\mathfrak{z}), h_2(\mathfrak{z}), t(\mathfrak{z}), \\ z_1(\mathfrak{z}\omega), t'(\omega\mathfrak{z}), z_2(\omega\mathfrak{z}), h_1(\omega\mathfrak{z}). \end{array} \right)$$

Compute multipoint evaluation challenge $u \in \mathbb{F}$:

$$u = \mathsf{Hash}(\mathsf{transcript}).$$

We now describe the verifier algorithm in a way that minimizes the number of $\mathbb{G}_1$ scalar multiplications.

3.4.2 Verifier Algorithm

Verifier preprocessed input:

$$[q_M(x)]_1, \quad [q_L(x)]_1, \quad [q_R(x)]_1, \quad [q_O(x)]_1, \quad [q_C(x)]_1, \quad [q_K(x)]_1,$$
$$[S_{\sigma_1}(x)]_1, \quad [S_{\sigma_2}(x)]_1, \quad [S_{\sigma_3}(x)]_1, \quad [\mathcal{T}_1(x)], \quad [\mathcal{T}_2(x)], \quad [\mathcal{T}_3(x)], \quad [x]_2.$$

$\mathcal{V}((w_i)_{i\in[\ell]}, \pi_{\mathcal{P}\mathfrak{lon}\mathcal{K}\mathfrak{up}})$:

1. Validate that $[a(x)]_1$, $[b(x)]_1$, $[c(x)]_1$, $[f(x)]_1$, $[t'(x)]_1$, $[h_1(x)]_1$, $[h_2(x)]_1$, $[z_1(x)]_1$, $[z_2(x)]_1$, $[q_{\text{low}}(x)]_1$, $[q_{\text{mid}}(x)]_1$, $[q_{\text{high}}(x)]$, $[W_{\mathfrak{z}}(x)]_1$, $[W_{\mathfrak{z}\omega}(x)]_1 \in \mathbb{G}_1$.
2. Validate that $a(\mathfrak{z}), b(\mathfrak{z}), c(\mathfrak{z}), S_{\sigma_1}(\mathfrak{z}), S_{\sigma_2}(\mathfrak{z}), f(\mathfrak{z}), t'(\mathfrak{z}), h_2(\mathfrak{z}), t(\mathfrak{z}), z_1(\mathfrak{z}\omega), t'(\omega\mathfrak{z}), z_2(\omega\mathfrak{z}), h_1(\omega\mathfrak{z}) \in \mathbb{F}$.
3. Validate that $(w_i)_{i\in[\ell]} \in \mathbb{F}^\ell$.
4. Compute the challenges $\zeta, \beta, \gamma, \delta, \varepsilon, \theta, \alpha, \mathfrak{z}, v, u \in \mathbb{F}$ as in prover description, from the common preprocessed inputs, public input, and elements of $\pi_{\mathcal{P}\mathfrak{lon}\mathcal{K}\mathfrak{up}}$.
5. Compute the zero polynomial evaluation $Z_H(\mathfrak{z}) = \mathfrak{z}^n - 1$.
6. Compute the Lagrange polynomial evaluation $L_1(\mathfrak{z}) = \frac{\omega\,(\mathfrak{z}^n-1)}{n\,(\mathfrak{z}-\omega)}$.
7. Compute the public input polynomial evaluation $\mathrm{PI}(\mathfrak{z}) = \sum_{i=1}^{\ell} w_i L_i(\mathfrak{z})$.
8. Compute the public table commitment $[t(x)]_1 = [\mathcal{T}_1(x)]_1 + \zeta[\mathcal{T}_2(x)]_1 + \zeta^2[\mathcal{T}_3(x)]_1$.
9. To save a verifier scalar multiplication, we split $r(X)$ into its constant and non-constant terms. Compute $r(X)$'s constant term:
$$\begin{aligned} r_0 := \ & \mathrm{PI}(\mathfrak{z}) - \alpha(a(\mathfrak{z}) + \beta S_{\sigma_1}(\mathfrak{z}) + \gamma)(b(\mathfrak{z}) + \beta S_{\sigma_2}(\mathfrak{z}) + \gamma)(c(\mathfrak{z}) + \gamma) z_1(\mathfrak{z}\omega) - \alpha^2 L_1(\mathfrak{z}) \\ & - \alpha^4 z_2(\mathfrak{z}\omega)(\varepsilon(1+\delta) + \delta h_2(\mathfrak{z}))(\varepsilon(1+\delta) + h_2(\mathfrak{z}) + \delta h_1(\mathfrak{z}\omega))(\theta + t(\mathfrak{z})) - \alpha^5 L_1(\mathfrak{z}), \end{aligned}$$
and let $r'(X) := r(X) - r_0$.
10. Compute the first part of the batched polynomial commitment $[D]_1 := [r'(x)] + u([z_1(x)]_1 + v^2\,[z_2(x)]_1 + v^3\,[h_1(x)]_1)$:
$$\begin{aligned} [D]_1 := \ & a(\mathfrak{z})b(\mathfrak{z})\,[q_M(x)]_1 + a(\mathfrak{z})\,[q_L(x)]_1 + b(\mathfrak{z})\,[q_R(x)]_1 + c(\mathfrak{z})\,[q_O(x)]_1 + [q_C(x)]_1 \\ & + ((a(\mathfrak{z}) + \beta\mathfrak{z} + \gamma)(b(\mathfrak{z}) + \beta k_1\mathfrak{z} + \gamma)(c(\mathfrak{z}) + \beta k_2\mathfrak{z} + \gamma)\alpha + L_1(\mathfrak{z})\alpha^2 + u)\,[z_1(x)]_1 \\ & - (a(\mathfrak{z}) + \beta S_{\sigma_1}(\mathfrak{z}) + \gamma)(b(\mathfrak{z}) + \beta S_{\sigma_2}(\mathfrak{z}) + \gamma)\alpha\beta z_1(\mathfrak{z}\omega)\,[S_{\sigma_3}(x)]_1 \\ & + (a(\mathfrak{z}) + \zeta b(\mathfrak{z}) + \zeta^2 c(\mathfrak{z}) - f(\mathfrak{z}))\alpha^3\,[q_K(x)]_1 \\ & + ((1+\delta)(\varepsilon + f(\mathfrak{z}))(\varepsilon(1+\delta) + t'(\mathfrak{z}) + \delta t'(\mathfrak{z}\omega))(\theta + t'(\mathfrak{z}))\alpha^4 + L_1(\mathfrak{z})\alpha^5 + uv^2)\,[z_2(x)]_1 \\ & + (uv^3 - z_2(\mathfrak{z}\omega)(\varepsilon(1+\delta) + h_2(\mathfrak{z}) + \delta h_1(\mathfrak{z}\omega))(\theta + t(\mathfrak{z}))\alpha^4)\,[h_1(x)]_1 \\ & - Z_H(\mathfrak{z})([q_{\text{low}}(x)]_1 + \mathfrak{z}^{n+2} \cdot [q_{\text{mid}}(x)]_1 + \mathfrak{z}^{n+4} \cdot [q_{\text{high}}(x)]_1). \end{aligned}$$

11. Compute the full batched polynomial commitment $[F]_1$:

$$\begin{aligned}[F]_1 := \ & [D]_1 + v \cdot [a(x)]_1 + v^2 \cdot [b(x)]_1 + v^3 \cdot [c(x)]_1 + v^4 \cdot [S_{\sigma_1}(x)]_1 + v^5 \cdot [S_{\sigma_2}(x)]_1 \\ & + v^6 \cdot [f(x)]_1 + v^7 \cdot [t'(x)]_1 + v^8 \cdot [h_2(x)]_1 + v^9 \cdot [t(x)]_1 + u(v \cdot [t'(x)]_1).\end{aligned}$$

12. Compute the group-encoded batch evaluation $[E]_1$:

$$[E]_1 := \left[\begin{array}{l} -r_0 + v \cdot a(\mathfrak{z}) + v^2 \cdot b(\mathfrak{z}) + v^3 \cdot c(\mathfrak{z}) + v^4 \cdot S_{\sigma_1}(\mathfrak{z}) + v^5 \cdot S_{\sigma_2}(\mathfrak{z}) + v^6 \cdot f(\mathfrak{z}) \\ +v^7 \cdot t'(\mathfrak{z}) + v^8 \cdot h_2(\mathfrak{z}) + v^9 \cdot t(\mathfrak{z}) + u(z_1(\mathfrak{z}\omega) + v \cdot t'(\mathfrak{z}\omega) + v^2 \cdot z_2(\mathfrak{z}\omega) + v^3 \cdot h_1(\mathfrak{z}\omega)) \end{array} \right]_1 .$$

13. Finally, batch validate all evaluations:

$$e\left([W_{\mathfrak{z}}(x)]_1 + u \cdot [W_{\mathfrak{z}\omega}(x)]_1, [x]_2\right) \overset{?}{=} e\left(\mathfrak{z} \cdot [W_{\mathfrak{z}}(x)]_1 + u\mathfrak{z}\omega[W_{\mathfrak{z}\omega}(x)]_1 + [F]_1 - [E]_1, [1]_2\right).$$

4 Variations of the Protocol

In this Section, we propose a solution for circuits with multiple lookup tables, and a different way of hiding the wire polynomials.

4.1 Multiple Lookup Tables

Let $\mathcal{T}_1, \dots, \mathcal{T}_s \in \mathbb{F}^{n \times 3}$ be lookup tables associated to different functions. We want to prove that for each tuple (a_i, b_i, c_i) of a lookup gate, there exists a $j \in \{1, \dots, s\}$ such that $(a_i, b_i, c_i) \in \mathcal{T}_j$. Note that this happens if and only if for any $\delta_1, \delta_2, \delta_3$ it holds that

$$a_i + \delta_1 b_i + \delta_2 c_i + \delta_3 j \in \mathcal{T}',$$

where

$$\mathcal{T}' = \{(\mathcal{T}_l[k][1] + \delta_1 \mathcal{T}_l[k][2] + \delta_2 \mathcal{T}_l[k][3] + \delta_3 l)\}_{l \in [s]},$$

and k goes for all rows of $\mathcal{T}_j$. Let f_t be an interpolant of degree n of $[j_0, j_1, \dots, j_{n-1}] : f_t(g^i) = j_i$.

Then:

1. $\mathcal{P}$ sends polynomials f_a, f_b, f_c to $\mathcal{I}$.
2. $\mathcal{V}$ sends challenges $\delta_1, \delta_2, \delta_3$ to $\mathcal{P}$.
3. $\mathcal{P}$ computes table $\mathcal{T}' = \{(\mathcal{T}_l[k][1] + \delta_1 \mathcal{T}_l[k][2] + \delta_2 \mathcal{T}_l[k][3] + \delta_3 l)\}$.
4. $\mathcal{P}$ computes Z' using $x = a_i + \delta_1 b_i + \delta_2 c_i + \delta_3 j_i$ and sends it to $\mathcal{I}$.
5. $\mathcal{V}$ asks the same 4 identities as for regular tables using $f = f_a + \delta_1 f_b + \delta_2 f_c + \delta_3 f_t$.

4.2 Options for Hiding Polynomials

In $\mathcal{P}\mathfrak{lon}\mathcal{K}$, vectors of private prover inputs are interpolated into polynomials and blinded by adding to each a random multiple of the vanishing polynomial over a particular domain. For a polynomial opened k times, a k-th degree multiple is required to ensure hiding under the discrete logarithm. The proof was not originally included in $\mathcal{P}\mathfrak{lon}\mathcal{K}$, and we add a proof for completion.

Hiding Using the Vanishing Polynomial

Suppose we have a commitment scheme $\mathrm{com} : \mathbb{F}[x] \to \mathbb{G}$ that is hiding under the discrete logarithm, and a multiplicative supgroup $H = \{1, g, g^2, ..., g^{n-1}\}$ for some $g \in \mathbb{F}$. We can construct a Lagrange basis $\{L_i\}_{i=0}^{n-1}$ over H where $L_i(g^i) = 1$ but takes the value 0 for all other elements of H. Additionally, we can construct an n-degree polynomial Z_H that vanishes on H. For a wire of private inputs $\{w_1, ..., w_d\}$, we compute

$$w(x) = \sum_{i \in [d]} w_i L_i(x).$$

A wire polynomial opened k times can be hidden by adding a random k-th degree multiple of Z_H to w:

$$a(x) = w(x) + (b_0 + b_1 x + ... + b_k x^k) Z_H(x).$$

Now, $a(x)$ agrees with $w(x)$ on H, and can be used in place of $w(x)$ in the proving protocol. The prover can reveal a commitment $\mathrm{com}(a)$ and k openings $\{a(z_i)\}_{i \in [k]}$ with $z_i \notin H$ without leaking any information about w, provided that the discrete logarithm assumption holds in $\mathbb{G}$.

Proposition 1. *The blinding scheme above is hiding under the discrete logarithm assumption.*

Proof. Let $b(x) = b_0 + b_1 x + ... + b_k x^k$ be the kth-degree blinding polynomial. With $z_i \notin H$, an adversary A with a guess $\hat{w}$ for w can compute k points on the blinding polynomial b using the k openings:

$$b(z_1) = \frac{a(z_1) - \hat{w}(z_1)}{Z_H(z_1)}$$

$$\vdots$$

$$b(z_k) = \frac{a(z_k) - \hat{w}(z_k)}{Z_H(z_k)}$$

There is a unique polynomial b' of degree $k-1$ passing through these k points. Define $K = \{z_1, ..., z_k\}$ and Z_K the degree-k vanishing polynomial on K. The blinding polynomial b agrees with b' on K, implying that

$$b(x) = b'(x) + cZ_K(x)$$

for some $c \in \mathbb{F}$.

To confirm their guess for w the adversary must find d satisfying the equation:

$$\mathrm{com}(a) - \mathrm{com}(\hat{w}) - \mathrm{com}(b' Z_H) = d \cdot \mathrm{com}(Z_K Z_H),$$

which is equivalent to solving the discrete logarithm problem. □

Next, we propose an alternative method for blinding private input by appending $k+1$ random inputs to the vector before interpolation. This method provides the same security and may be more practical for certain implementations. A proof of this method's equivalency to the vanishing polynomial method is included.

Adding Random Wire Values

Instead of hiding the wire polynomial by adding a degree k multiple of the vanishing polynomial, we can instead append $k+1$ random elements to the wire and achieve the same security.

For wire values $\{w_i\}_{i\in[d]}$ and corresponding selector values $\{q_i\}_{i\in[d]}$, define the polynomials

$$w(x) = \sum_{i\in[d]} w_i L_i(x), \quad q(x) = \sum_{i\in[d]} q_i L_i(x),$$

where $\{L_i\}_{i\in[n]}$ is a Lagrange basis for $H = \{1, g, g^2, ..., g^{n-1}\}$.

For a polynomial opened k times, choose $k+1$ blinding factors $b_1, ..., b_{k+1}$, and similarly define

$$a(x) = w(x) + \sum_{j=1}^{k+1} b_j L_{d+j}(x).$$

Note that $a(x)$ is the Lagrange interpolation of $\{w_1, ..., w_d, b_1, ..., b_{k+1}\}$ over H.

Now, $a(x)$ agrees with $w(x)$ on $\{g^i\}_{i=0}^{d-1}$. The selector q is zero on $\{g^i\}_{i=d}^{d+k}$, so $a(x)$ can be used in place of $w(x)$ during the proving rounds.

Proposition 2. *The blinding scheme above is also hiding under the discrete logarithm assumption.*

Proof. The sum $\sum_{j=1}^{k+1} b_j L_{d+j}(x)$ vanishes on $D = \{1, g, ..., g^d\}$, and so $a(x)$ can be rewritten as:

$$a(x) = w(x) + b(x)Z_D(x),$$

where $b(x)$ is the appropriate k-degree polynomial. From here, we can proceed similarly to previous proof. □

5 Conclusions

In this article we presented our work on $\mathcal{P}$lon$\mathcal{K}$up, a ZK-SNARK that integrates **plookup** into $\mathcal{P}$lon$\mathcal{K}$ in an efficient way. To do so, we introduced a new selector q_K that activates or switches off the lookup gates. In order to check the equations from **plookup** and $\mathcal{P}$lon$\mathcal{K}$ simultaneously, we use dummy values of the table vector when the lookup gates are activated. On the other hand, to make the protocol efficient, we sort the table vector in a way that the sorted version by this vector can be computed very efficiently. The polynomials $h_1(X)$ and $h_2(X)$ are chosen by separating even an odd indexes of s. Notice that this differs from the original structure used by **plookup**, but allows us to use less identities for checking our grand product. The protocol is explained so that its implementation can be done following the rounds described. In this context, we would like to mention that the implementation of $\mathcal{P}$lon$\mathcal{K}$up is currently a work in progress available at [Net21].

References

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random Beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. Available online: http://eprint.iacr.org/2017/1050.

[CHM$^+$20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. *Marlin: Preprocessing zkSNARKs with universal and updatable SRS*, pages 738–768. 05 2020.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 626–645, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[GKM$^+$18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In *Advances in Cryptology – CRYPTO 2018*, pages 698–728, Cham, 2018. Springer International Publishing.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

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

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

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2111–2128, New York, NY, USA, 2019. Association for Computing Machinery.

[Net21] Dusk Network. Pure rust implementation of the PlonK proving system over BLS12-381. GitHub, 2021. Available online: https://github.com/dusk-network/plonk.

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