

A simplified polynomial protocol for look-up tables

Ariel Gabizon
Aztec

Zachary J. Williamson
Aztec

March 12, 2020

Abstract

We present a protocol for checking the values of a committed polynomial $f \in \mathbb{F}_{<n}[X]$ over a multiplicative subgroup $H \subset \mathbb{F}$ of size n , are contained in the values of a different committed polynomial $t \in \mathbb{F}_{<d}[X]$ over the same subgroup H . Our protocol can be viewed as a simplification of one from Bootle et. al [BCG⁺] for a similar problem, with potential efficiency improvements. In particular, [BCG⁺]'s protocol requires committing to several auxiliary polynomials of degree $d \cdot \log n$, whereas ours requires only one commitment to a single auxiliary polynomial of degree $d+n$, which is much smaller in the case $d \sim n$.

One common use case of this primitive in the zk-SNARK setting is a “batched range proof”, where one wishes to check all of f ’s values on H are in a range $[0, \dots, M-1]$. We present a slightly optimized protocol for this special case, and pose improving it as an open problem.

1 Introduction

When wanting to use zk-SNARKs to prove statements involving standard primitives like AES-128 or SHA-256, one runs into the problem that the operations involved in these primitives are “SNARK unfriendly”, in the sense of having large representations in the native SNARK language, which typically corresponds to low degree equations over a large prime field. Examples of operations with large overhead, are ones involving bit decomposition like bitwise XOR or AND. This has lead to extensive research into STARK and SNARK friendly hash functions that are based solely on native field operations.

We investigate an alternative approach, where for commonly used operations we precompute a look-up table of the legitimate (input, output) combinations; and the prover argues the witness values exist in this table.

In polynomial language, this ultimately boils down to proving polynomials are the same “up to multiplicities”. That is, suppose that the values in the look-up table are

$\{t_i\}_{i \in [d]}$ and the values in the witness are $\{f_i\}_{i \in [n]}$. We want to show that the polynomials

$$F(X) := \prod_{i \in [n]} (X - f_i), G(X) := \prod_{i \in [d]} (X - t_i)$$

have the same roots, ignoring multiplicities, i.e. that for non-negative integers $\{e_i\}_{i \in [d]}$ we have $F(X) = \prod_{i \in [d]} (X - t_i)^{e_i}$. Bootle et. al [BCG⁺] gave an algorithm for this exact problem, also in the context of efficient SNARK arithmetization of a common operation (in their case, repeatedly checking that field elements correspond to certain convenient sparse representations of boolean strings).

Their algorithm requires committing to a vector of length $d \log n$ that contains for each $i \in [d], j \in [\log n]$ the value $t_i^{2^j}$. They also commit to the binary decomposition of the $\{e_i\}$, and using the two, prove that F is of the desired form.

We present here an arguably simpler protocol for the same problem, that doesn't require explicitly representing the multiplicities. Let us use the notation $f \subset t$ as shorthand for $\{f_i\}_{i \in [n]} \subset \{t_i\}_{i \in [d]}$.

The idea is to look at a sorted version $\{s_i\}$ of the values $\{f_i\}$, and compare the set of non-zero *differences* in $\{s_i\}$ and $\{t_i\}$. Note that if $f \subset t$, and every element of t appears at least once in f , we indeed have that these sets of differences are the same. However, the converse is not true: We can create a sequence of values $\{s_i\}$ having the same difference set as $\{t_i\}$, but with the differences appearing in *different order*; and in this case we won't have $s \subset t$. As an illustrating example, take

$$t = \{1, 4, 8\}, s = \{1, 1, 4, 8, 8, 8\}, s' = \{1, 5, 5, 5, 8, 8\}$$

All three sets have the same difference set $\{3, 4\}$; but since those differences appear in different order in s' and t , we don't have $s' \subset t$. We also didn't address checking the starting point is the same in s and t . Both these issues can be solved by comparing *randomized* difference sets: We choose random $\beta \in \mathbb{F}$, and compare the elements in the sequences $\{t_i + \beta t_{i+1}\}_{i \in [d-1]}, \{s_i + \beta s_{i+1}\}_{i \in [n-1]}$.

Seemingly, we have taken “one step forward and one back” by adding the random β , as now pairs (s_i, s_{i+1}) with $s_i = s_{i+1}$ will give a non-zero contribution $s_i + \beta \cdot s_{i+1} = (1 + \beta) \cdot s_i$. However, all elements corresponding to a repetition are now multiples of $(1 + \beta)$ and we are able to use this to “identify” them and not use them in the comparison with the table. Specifically, we show that this check can be done correctly and efficiently using a “grand product argument” similar to the one used in [GWC19]’s permutation argument.

A technicality is that as mentioned this approach assumes all of t ’s values appear at least once in f ; which is one reason we in fact take s to be the sorted version of the *concatenation* of f and t . This ends up also helping to batch both checks into one product: The check that $f \subset s$, and the check that $s \subset t$.

Precise details of the scheme are given in Section 3.

2 Preliminaries

Polynomial Protocols We use the ranged polynomial protocol terminology from [GWC19] to describe our protocols. Here a prover sends polynomials of a certain degree bound d , and at the end of protocol, the verifier can ask whether certain identities hold between the polynomials sent during the protocol, the input polynomials, and the preprocessed polynomials, on a predefined set H . It is shown in [GWC19] how such a protocol can be compiled into one in the algebraic group model using the polynomial commitment scheme of [KZG10].

For such a protocol \mathcal{P} , let $\mathfrak{d}(\mathcal{P})$ be the maximum over s_1, \dots, s_t sent by an honest prover during protocol execution of the sum of the degrees of the s_i plus the maximal degree of the verifier identity when plugging in s_1, \dots, s_t into the verifier identity minus the size of H .

The prover complexity ends up being closely tied to $\mathfrak{d}(\mathcal{P})$ as this corresponds to the number of group operations to commit to all polynomials (including the quotient polynomial that comes up in translating a ranged polynomial protocol to a polynomial protocol) using [KZG10]. We will thus attempt to minimize $\mathfrak{d}(\mathcal{P})$ in our protocols.

We refer the reader to Section 4 of [GWC19] for more details.

3 The main scheme

Notation: Fix integers n, d and vectors $f \in \mathbb{F}^n, t \in \mathbb{F}^d$. We use the notation $f \subset t$ to mean $\{f_i\}_{i \in [n]} \subset \{t_i\}_{i \in [d]}$. Let $H = \{\mathbf{g}, \dots, \mathbf{g}^{n+1} = 1\}$ be a multiplicative subgroup of order $n+1$ in \mathbb{F} . Denote by κ be an element of a non-trivial coset of H in \mathbb{F}^* . For a polynomial $f \in \mathbb{F}[X]$ and $i \in [n+1]$ we sometimes denote $f_i := f(\mathbf{g}^i)$. For a vector $f \in \mathbb{F}^n$, we also denote by f the polynomial in $\mathbb{F}_{<n}[X]$ with $f(\mathbf{g}^i) = f_i$.

When $f \subset t$, we say that f is *sorted by t* when values appear in the same order in f as they do in t . Formally, for any $i < i' \in [n]$ such that $f_i \neq f_{i'}$, if $j, j' \in [d]$ are such that $t_j = f_i, t_{j'} = f_{i'}$ then $j < j'$.

Now, given $t \in \mathbb{F}^d, f \in \mathbb{F}^n, s \in \mathbb{F}^{n+d}$, define bi-variate polynomials F, G as

$$F(\beta, \gamma) := (1 + \beta)^n \cdot \prod_{i \in [n]} (\gamma + f_i) \prod_{i \in [d-1]} (\gamma(1 + \beta) + t_i + \beta t_{i+1})$$

$$G(\beta, \gamma) := \prod_{i \in [n+d-1]} (\gamma(1 + \beta) + s_i + \beta s_{i+1})$$

we have

Claim 3.1. $F \equiv G$ if and only if

1. $f \subset t$, and
2. s is (f, t) sorted by t .

Proof. We write F, G as elements of $\mathbb{F}(\beta)[\gamma]$ while taking out a $(1 + \beta)$ factor as follows.

$$F(\beta, \gamma) = (1 + \beta)^{n+d-1} \cdot \prod_{i \in [n]} (\gamma + f_i) \prod_{i \in [d-1]} (\gamma + (t_i + \beta t_{i+1}) / (1 + \beta))$$

$$G(\beta, \gamma) = (1 + \beta)^{n+d-1} \prod_{i \in [n+d-1]} (\gamma + (s_i + \beta s_{i+1}) / (1 + \beta))$$

Suppose first that $f \subset t$ and $s \in \mathbb{F}^{n+d}$ is (f, t) sorted by t .

Then for each $j \in [d-1]$, there is an index $i \in [n+d-1]$ such that $(t_j, t_{j+1}) = (s_i, s_{i+1})$. The corresponding factors in F, G are equal. That is,

$$(\gamma + (t_j + \beta t_{j+1}) / (1 + \beta)) \equiv (\gamma + (s_i + \beta s_{i+1}) / (1 + \beta))$$

Let $P' \subset [n+d-1]$ the set of these $d-1$ indices i , and let $P := [n+d-1] \setminus P'$. The n indices $i \in P$ are such that $s_i = s_{i+1}$, and $\{s_i\}_{i \in P}$ equals $\{f_i\}_{i \in [n]}$. For each $i \in P$, the corresponding factor of G , will be

$$\gamma + (s_i + \beta s_{i+1}) / (1 + \beta) = \gamma + s_i,$$

which, for the j such that $s_i = f_j$ will equal the factor $(\gamma + f_j)$ in F .

For the other direction, assume $F \equiv G$ as polynomials in $\mathbb{F}[\beta, \gamma]$. Then $F \equiv G$ also as elements of $\mathbb{F}(\beta)[\gamma]$. Since $\mathbb{F}(\beta)[\gamma]$ is a unique factorization domain, we know that the linear factors of F, G , as written above must be equal. Thus, for each $i \in [d-1]$, G must have a factor equal to $(\gamma + (t_i + \beta t_{i+1}) / (1 + \beta))$. In other words, for some $j \in [n+d-1]$,

$$\gamma + (t_i + \beta t_{i+1}) / (1 + \beta) = \gamma + (s_j + \beta s_{j+1}) / (1 + \beta),$$

which implies $t_i + \beta t_{i+1} = s_j + \beta s_{j+1}$, and therefore $t_i = s_j, t_{i+1} = s_{j+1}$. Call $P' \subset [n+d-1]$ the set of these $d-1$ indices j . For any index $j \in [n+d-1] \setminus P'$, there must be a factor “coming from f ” in F that equals the corresponding factor in G . More precisely, for such j there exists $i \in [n]$ such that

$$\gamma + f_i = \gamma + (s_j + \beta s_{j+1}) / (1 + \beta),$$

or equivalently

$$f_i + \beta f_i = s_j + \beta s_{j+1}$$

which implies $f_i = s_j = s_{j+1}$.

Thus, we know that whenever consecutive values in s are different, they are exactly equal to two consecutive values in t , and all values of f are values of t . □

Claim 3.1 motivates the following protocol. It will be convenient to assume $d = n+1$. (If $d \leq n$ pad t with $n-d+1$ repetitions of the last element.)

Preprocessed polynomials: The polynomial $t \in \mathbb{F}_{<n}[X]$ describing the lookup values.

Inputs: $f \in \mathbb{F}_{<n}[X]$

Protocol:

1. Let $s \in \mathbb{F}^{2n+1}$ be the vector that is (f, t) sorted by t . Also denote by s the following polynomial in $\mathbb{F}_{<2n+2}[X]$ that represents s on $H \cup \kappa H$: $s(\mathbf{g}^i) = s_i$ for $i \in [n+1]$; and $s(\kappa \mathbf{g}^i) = s_{n+i}$ for each $i \in [n+1]$.
2. P_{poly} computes the polynomial s and sends it to \mathcal{I} .
3. V_{poly} chooses random $\beta, \gamma \in \mathbb{F}$ and sends them to P_{poly} .
4. P_{poly} computes a polynomial $Z \in \mathbb{F}_{<n+1}[X]$ that aggregates the value $F(\beta, \gamma)/G(\beta, \gamma)$ where F, G are as described above. Specifically, we let
 - (a) $Z(\mathbf{g}) = 1$,
 - (b) For $2 \leq i \leq n$

$$Z(\mathbf{g}^i) = \frac{(1 + \beta)^n \prod_{j < i} (\gamma + f_j) \cdot \prod_{1 \leq j < i} (\gamma(1 + \beta) + t_j + \beta t_{j+1})}{\prod_{1 \leq j < i} (\gamma(1 + \beta) + s_j + \beta s_{j+1}) (\gamma(1 + \beta) + s_{n+j} + \beta s_{j+1})},$$

and

- (c) $Z(\mathbf{g}^{n+1}) = 1$.
5. P_{poly} sends Z to \mathcal{I} .
6. V_{poly} checks that Z is indeed of the form described above, and that $Z(\mathbf{g}^{n+d}) = 1$. More precisely, V_{poly} checks the following identities on H .
 - (a) $L_1(Z - 1) \equiv 0$
 - (b)

$$(X - \mathbf{g}^{n+1})Z(X)(1 + \beta) \cdot (\gamma + f(X))(\gamma(1 + \beta) + t(X) + \beta t(X \cdot \mathbf{g}))$$

$$\equiv (X - \mathbf{g}^{n+1})Z(X \cdot \mathbf{g})(\gamma(1 + \beta) + s(X) + \beta s(X \cdot \mathbf{g}))(\gamma(1 + \beta) + s(\kappa \cdot X) + \beta s(\kappa \cdot X \cdot \mathbf{g})).$$
 - (c) $L_{n+1}(s(X) - s(\kappa X \mathbf{g})) \equiv 0$
 - (d) $L_{n+1}(Z(X) - 1) \equiv 0$

and outputs acc iff all checks hold.

Lemma 3.2. *Suppose that $\{f(\mathbf{g}^i)\}_{i \in [n]}$ is not contained in $\{t(\mathbf{g}^i)\}_{i \in [d]}$. Then for any strategy of \mathcal{A} playing the role of P_{poly} in the above protocol, the probability that V_{poly} accepts is $\text{negl}(\lambda)$.*

Proof. Using Claim 3.1 we know that when f 's range is not contained in t 's for any choice of s sent by P_{poly} the polynomials $F(\beta, \gamma), G(\beta, \gamma)$ are different. From the SZ lemma e.w.p $\text{negl}(\lambda)$ V_{poly} chooses β, γ such that $F(\beta, \gamma) \neq G(\beta, \gamma)$. In this case we have $Z(\mathbf{g}^{n+d}) \neq 1$ which means V_{poly} rejects. \square

4 Generalizing to vector lookups and multiple tables

Suppose we have several witness polynomials $f_1, \dots, f_w \in \mathbb{F}_{<n}[X]$, and a table of values $t^* \in (\mathbb{F}^w)^d$. We wish to check that for each $j \in [n]$ $(f_1(\mathbf{g}^j), \dots, f_w(\mathbf{g}^j)) \in t^*$. We can use randomization to efficiently reduce to the case of Section 3.

For each $i \in [w]$ we will include in the set of preprocessed polynomials $t_i \in \mathbb{F}_{<d}[X]$ with $t_i(\mathbf{g}^j) = t_{i,j}$ for each $j \in [d]$.

The verifier will choose random $\alpha \in \mathbb{F}$.

Then we will define $t := \sum_{i \in [w]} \alpha^i t_i$, $f := \sum_{i \in [w]} \alpha^i f_i$.

Assume that for some $j \in [n]$, $(f_1(\mathbf{g}^j), \dots, f_w(\mathbf{g}^j)) \notin t^*$. Then e.w.p $d \cdot w/|\mathbb{F}|$, $f(\mathbf{g}^j) \notin t$. Thus, after the selection of α , we can run the protocol of the previous section on f, t .

As alluded to in the introduction, a natural use case for this vector lookup primitive is a key-value setting, where we have a function f with $w - 1$ inputs, and wish to verify a vector is of the form $(x_1, \dots, x_{w-1}, f(x_1, \dots, x_{w-1}))$ for some input (x_1, \dots, x_{w-1}) .

4.1 Multiple tables

Suppose further that we have in fact have multiple tables t_1^*, \dots, t_ℓ^* and for each $i \in [n]$ wish to check that for some predefined $j = j(i) \in [\ell]$ $(f_1(\mathbf{g}^i), \dots, f_w(\mathbf{g}^i)) \in t_j^*$. We can reduce to the previous setting as follows. We create a preprocessed table containing t_1^*, \dots, t_ℓ^* as sub-tables, by adding a column specifying the table index.

That is, suppose for simplicity that for each $j \in [\ell]$, $t_j^* \in (\mathbb{F}^w)^{d/\ell}$. We construct $t^* \in (\mathbb{F}^{w+1})^d$ consisting for each $j \in [\ell], i \in [d/\ell]$ of the element $(j, (t_j^*)_i)$. We preprocess a polynomial $q \in \mathbb{F}_{<n}[X]$ such that $q_i = j(i)$, where again $j(i)$ is the relevant subtable to be checked for the i 'th row. And now apply the method above to check that for each $i \in [n]$ $(q(\mathbf{g}^i), f_1(\mathbf{g}^i), \dots, f_w(\mathbf{g}^i)) \in t^*$

5 An optimized solution for continuous ranges

Suppose we wish to check that $f_i \in \{0, \dots, d - 1\}$ for some integer $d < n$. We could use our above protocol while setting $t_i = i - 1$ for $i \in [d]$. By using e.g. $d = n$, the above protocol allows us to check $f \in \{0, \dots, n - 1\}$ using one auxiliary polynomial of degree $2n$ (the polynomial s representing the sorted witness) and another of degree $n + 1$ (the product aggregating polynomial Z). We show a variant of the protocol specifically for continuous ranges with the same prover efficiency allowing for the larger range $\{0, \dots, c \cdot n - 1\}$.

Let us measure the cost of this in terms of auxiliary polynomials given. \mathbf{P} would need to send the sorted polynomial $s \in \mathbb{F}_{<n+d}[X]$. And the preprocessing would require computing $t \in \mathbb{F}_{<d}[X]$. This would start being prohibitively expensive e.g. for $d = 2^{32}$.

For such large d , one solution would be to split each value of f into two values in the range $\{0, \dots, \sqrt{d}\}$. And then check the values are in this smaller range. This will double the size of f to $2n$, and require $s \in \mathbb{F}_{<2n+\sqrt{d}}[X]$.

We propose an alternative solution where we remain with $f \in \mathbb{F}_{<n}[X]$ and require two auxiliary polynomials $s, s' \in \mathbb{F}_{<n}[X]$

References

- [BCG⁺] J. Bootle, A. Cerulli, J. Groth, S. K. Jakobsen, and M. Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 595–626.
- [GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptology ePrint Archive*, 2019:953, 2019.
- [KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. pages 177–194, 2010.