PLONKUP SCHEME WITH MULTIPLE QUERIES

ALEXANDR BULKIN AND TIM DOKCHITSER

Abstract. There is a line of 'lookup' protocols to show that all elements of a sequence $f \in \mathbb{F}^n$ are contained in a table $t \in \mathbb{F}^d$, for some field $\mathbb{F}$. Lookup has become an important primitive in Zero Knowledge Virtual Machines, and is used for range checks and other parts of the proofs of a correct program execution. In this note we give several variants of the protocol. We adapt it to the situation when there are multiple lookups with the same table (as is usually the case with range checks), and handle also 'bounded lookup' that caps the number of repetitions.

1. Introduction

A zero-knowledge proof allows a *prover* to convince a *verifier* that a statement is true without revealing any additional information [GMR85]. We are interested in 'general purpose Zero Knowledge Virtual Machines (ZKVMs)' that handle the statements of the form 'the following program terminates correctly after $\leq N$ steps'. In [B+18] (see also [Cer19]), Bootle et al. describe in detail how to build such a ZKVM, for programs in the TinyRAM assembly language [BS+13]. Essentially, execution trace of the program is stored in columns of size N, so that one column keeps the value of the program counter after every time step, one keeps the flag, one keeps register #1, etc. Then correct program execution is rephrased in terms of arithmetic constraints on the columns.

To handle range checks and memory, they introduced two primitives, called **lookup** and **bounded lookup**. For two columns $f = (f_i)$ ('query') and $t = (t_i)$ ('table'), they provide a proof that $f \subset t$ as a set, and, for the bounded version, a cap on the number of repetitions of one element in f; in fact, they only need the case when the cap is one. Gabizon et al. [G+20] have improved the lookup scheme ('plookup') and Pearson et al. [P+22] improved it even further ('plonkup'); see also [A+22]. All these schemes readily generalize to the 'multiple input single output' setting when one proves that $f(x_1, .., x_m) = y$ with a lookup in the table of valid tuples $(x_1, .., x_m, y)$. With this in mind, these primitives have proved extremely useful since, especially in implementing functions f that are 'unfriendly' for arithmetic operations (bitwise operations, hash functions such as SHA-256, etc.), by breaking them into a combination of quadratic constraints and manageable lookups.

In this note, we offer several modifications of the plonkup scheme. One handles efficiently multiple queries to the same table (§3). This is relevant in the ZKVM setting and applies, for instance, to multiple range checks in the same word range

Date: January 4, 2023.

(as in [B+18]). Another handles general 'bounded lookup' (§4) and a special case of 'injective lookup' (§5) in the same context.

2. Plonkup protocol

Consider vectors $f = (f_i)_{i=1}^n$ ('query') and $t = (t_i)_{i=1}^d$ ('table') with values in some field $\mathbb{F}$. The problem is to provide a proof that $f \subset t$ as sets. Let s be a vector of length $n + d$. Gabizon et al. [G+20] prove that the three conditions

(1) $f \subset t$ as a set (unordered, ignoring repetitions)
(2) $s = f \cup t$ as a multiset (unordered, with correct multiplicities)
(3) s is sorted by t (if $s_i = t_{i'}, s_j = t_{j'}$ with $s_i \neq s_j$, then $i < j \Leftrightarrow i' < j'$)

are satisfied if and only if the following polynomials in $\mathbb{F}[\beta, \gamma]$ are equal:

$$(1+\beta)^n \prod_{i=1}^{n}(\gamma + f_i) \prod_{i=1}^{d-1}(\gamma(1+\beta) + t_i + \beta t_{i+1}) = \prod_{i=1}^{n+d-1} (\gamma(1+\beta) + s_i + \beta s_{i+1}).$$

When $d = n + 1$, this takes the form $\prod_{i=1}^{n} \cdots \prod_{i=1}^{n} \cdots / \prod_{i=1}^{2n} \cdots = 1$. Splitting the last product into pairs, we get a condition of the form $\prod_{i=1}^{n} q_i = 1$, which can be verified by building a column with partial products $z_i = \prod_{j=1}^{i-1} q_j$, and imposing the conditions that $z_1 = z_{n+1} = 1$ and a recursive formula $z_{i+1} = q_i z_i$. In practice, β and γ are taken to be 'random' constants in $\mathbb{F}$ that come from the verifier, after f, t and s have been committed.

In $\prod_{i=1}^{2n} \cdots$ above, the ith and $(i + n)$th term are paired together in [G+20], and [P+22] observe that it is more efficient to pair ith and $(i+1)$st, saving one boundary condition. The 'cost' of the resulting lookup ('plonkup') is

- 4 columns (in addition to f, t): z, q, and two halves h_1, h_2 of s,
- 2 quadratic constraints: $z_{i+1} = q_i z_i$ and the expression for q_i,
- boundary conditions: $z_1 = z_{n+1} = 1$.

3. Multiple Queries Lookup

Our first observation is that for multiple queries $f^{(1)},...,f^{(k)}$ with *the same* table t, we can use essentially the same scheme to decrease the cost of $4k$ extra columns to $2k + 2$. The algorithm is as follows:

Algorithm 1 (Multiple Queries Lookup)**.** *Let* $(f_i^{(1)})_{i=1}^n,...,(f_i^{(k)})_{i=1}^n$ *and* $t = (t_i)_{i=1}^{n+1}$ *be vectors in a field* $\mathbb{F}$*. The algorithm provides a proof that all* $f_i^{(j)} \in t$*.*

Prover:

(1) *Set* $s = t \cup f^{(1)} \cup \ldots \cup f^{(k)}$ *as a multiset, sorted by* t*.*
(2) *Set* $h^{(j)} = (s_j, s_{j+k+1}, s_{j+2(k+1)}, ...,)$ *for* $j = 1, ..., k+1$*.*
Thus, $|s| = kn + n + 1$*,* $|h^{(1)}| = n + 1$ *and* $|h^{(2)}| = ... = |h^{(k+1)}| = n$*.*
(3) *Commit* $h^{(1)}, ..., h^{(k+1)}$*.*

(4) *Get random constants* $\beta, \gamma \in \mathbb{F}$ *from the verifier*[1].
(5) *Compute vectors* z, $q^{(1)}, ..., q^{(k)}$ *recursively, as follows.*

```
z_1 ← 1
for i ← 1 to n do
    v ← γ(1+β)+h_i^(k+1)+βh_{i+1}^(1)
    for j ← k to 1 by −1 do
        v *:= (γ(1+β)+h_i^(j)+βh_i^(j+1))/(γ+f_i^(j))
        q_i^(j) := v
    end for
    z_{i+1} := z_i(γ(1+β)+t_i+βt_{i+1})(1+β)^k/v
end for
```

(6) *Commit* z, $q^{(1)}, ..., q^{(k)}$.

Verifier: *Check that* $z_1 = z_{n+1} = 1$*, and that the following* $k+1$ *quadratic constraints hold. For all* $i = 1, ..., n$*,*

(1) $z_{i+1}q_i^{(1)} = z_i(1+\beta)^k(\gamma(1+\beta)+t_i+\beta t_{i+1})$,
(2) $(\gamma(1+\beta)+h_i^{(j)}+\beta h_i^{(j+1)})q_i^{(j+1)} = (\gamma+f_i^{(j)})q_i^{(j)} \quad \text{for} \quad j = 1 \ldots k-1$,
(3) $(\gamma(1+\beta)+h_i^{(k)}+\beta h_i^{(k+1)})(\gamma(1+\beta)+h_i^{(k+1)}+\beta h_{i+1}^{(1)}) = (\gamma+f_i^{(k)})q_i^{(k)}$.

4. Bounded lookup

We now turn to 'bounded lookup', a version of the above scheme that puts a cap m on how many times an entry in t can occur in the $f^{(j)}$ in total. In practice, the $f^{(j)}$ and t can have different sizes ($\leq n$), and we pad them with a 'dummy' value $\mu \in t$. Observe that in the scheme of §3, assuming the table t has no repetitions except for μ and has $> m$ distinct values,

$$\begin{aligned} \text{every } t_i \neq \mu \text{ occurs} \leq m \text{ times in } \textstyle\bigcup_j f^{(j)} \quad &\iff \quad \text{all multiplicities in } s \text{ are} \leq m+1 \\ &\iff \quad s_{i-m-1} \neq s_i \text{ for all } i. \end{aligned}$$

We let all indices in t, f, h, and I cycle modulo the column size $N = n+1$. For $i = 1$ for example, t_{i-1} refers to t_{n+1}, t_{i-2} to t_n etc. (In practice, all the column entries are elements of some finite cyclic group of order N.) From Algorithm 1(2) we see that $h_i^{(j)} = s_p$ with $p = j+(k+1)(i-1)$, and it follows easily that $s_{p-m-1} = h_{i_m}^{(j_m)}$ where $1 \leq j_m \leq k+1$ is the unique integer congruent to $j-m-1 \bmod k+1$, and $i_m = i + (j - j_m - m - 1)/(k+1)$.

With the above observation, the 'lookup' algorithm upgrades to 'bounded lookup' as follows, at a cost of one additional column and one constraint per query.

Algorithm 2 (Multiple Queries Bounded Lookup)**.** *Let* $(f_i^{(1)})_{i=1}^n, \ldots, (f_i^{(k)})_{i=1}^n$ *and* $t = (t_i)_{i=1}^{n+1}$ *be vectors in a field* $\mathbb{F}$*,* $\mu \in t$ *and* $m \geq 1$*. The algorithm provides a proof that* $f_i^{(j)} \in t$*, and all* $t_i \neq \mu$ *occur* $\leq m$ *times as an entry.*
Prover:

[1]in practice, via the Fiat-Shamir heuristic, through the hash of $(f, h^{(1)}, ..., h^{(k+1)})$ from (3)

(1) *Apply Algorithm 1 to show that all* $f_i^{(j)} \in t$. *Let* $h^{(j)}$ *be as in the algorithm, except we view them as all having length* $n+1$ *with* $h_{n+1}^{(j)} = \mu$ *for* $j > 1$.
(2) *Construct and commit* $k+1$ *vectors* $I^{(j)} = (I_i^{(j)})_{i=1}^{n+1}$, *with* $I_i^{(j)} = (h_i^{(j)} - h_{i_m}^{(j_m)})^{-1}$ *when* $h_i^{(j)} \neq h_{i_m}^{(j_m)}$ *and* $I_i^{(j)} = 0$ *otherwise.*

Verifier: *In addition to the constraints in Algorithm 1, check that*

$$\big((h_i^{(j)} - h_{i_m}^{(j_m)})I_i^{(j)} - 1\big)\big(h_i^{(j)} - \mu\big) = 0, \qquad 1 \leq i \leq n+1,\ 1 \leq j \leq k+1.$$

5. Injective lookup

Finally, we discuss the important case of bounded lookup with $m = 1$. This is the case used for memory accesses in [B+18, Cer19]. The algorithm then shows that every value $t_i \neq \mu$ occurs at most once in $\bigcup_j f^{(j)}$. However, in this case the above scheme can be improved from $4k+2$ auxiliary columns to $2k$. View $f^{(j)}$ as having length $n+1$, padding them with μ if necessary. What we want to show is that there are pairwise disjoint sets of indices $X^{(j)} \subset \{1, ..., n+1\}$ such that

$$f^{(j)} = \text{permutation of } \big\{t_i \text{ if } i \in X^{(j)} \text{ and } \mu \text{ if } i \notin X^{(j)}\big\}_{i=1}^{n+1}.$$

Proving that two vectors v and w are permutations of one another is easy with one extra vector $z = (z_i)$, $z_i = \prod_{j=1}^{i} \frac{\beta - v_j}{\beta - w_j}$ that satisfies an obvious constraint from the recursion relating z_i and z_{i-1}, and boundary conditions $(z_0 =) z_{n+1} = 1$. We get:

Algorithm 3 (Multiple Queries Injective Lookup)**.** *Let* $(f_i^{(1)})_{i=1}^{n+1}, \ldots, (f_i^{(k)})_{i=1}^{n+1}$ *and* $t = (t_i)_{i=1}^{n+1}$ *be vectors in a field* $\mathbb{F}$, *and* $\mu \in t$. *The algorithm provides a proof that* $f_i^{(j)} \in t$, *and every* $t_i \neq \mu$ *occurs at most once as an entry.*

Prover:

(1) *Let* $X^{(j)} \subset \{1, ..., n+1\}$ *be pairwise disjoint sets of indices for* $j = 1, ..., k$ *so that*

$$f^{(j)} = \textit{permutation of } t^{(j)},\ t^{(j)} = \{t_i \textit{ if } i \in X^{(j)} \textit{ and } \mu \textit{ if } i \notin X^{(j)}\}_{i=1}^{n+1}.$$

(2) *Commit* $f^{(j)}$, t *and* $t^{(j)}$ *for* $j = 1, \ldots, k$.
(3) *Get random constant* $\beta \in \mathbb{F}$ *from the verifier.*
(4) *Define vectors* $z^{(j)} = \big(\prod_{l=1}^{i} \frac{\beta - f_l^{(j)}}{\beta - t_l^{(j)}}\big)_{i=1}^{n+1}$ *for* $j = 1, ..., k$, *and commit them.*

Verifier: *Verify that*

(1) $z_i^{(j)}(\beta - t_i^{(j)}) = z_{i-1}^{(j)}(\beta - f_i^{(j)})$ *for* $i = 1, .., n+1$ *and* $z_{n+1}^{(j)} = 1$, *for* $j = 1, ..., k$.
(2) $(t_i^{(j)} - t_i)(t_i^{(j)} - \mu) = 0$ *for* $i = 1, .., n+1$, *and* $j = 1, ..., k$.
(3) $\big(t_i + (k-1)\mu - \sum_{j=1}^{k} t_i^{(j)}\big)\big(k\mu - \sum_{j=1}^{k} t_i^{(j)}\big) = 0$ *for* $i = 1, .., n+1$.

The first condition shows that $f^{(j)}$ and $t^{(j)}$ are permutations of one another. The second shows that $t^{(j)}$ either agrees with t or equals μ at every place. The last one guarantees that at every place $t^{(j)}$ can agree with t for at most one j, which is equivalent to the $X^{(j)}$ being disjoint.

Finally, we refer the reader to [A+22] for the discussion of lookup in the context when the $f^{(j)}$ and t have very different size, and optimizations for the prover in that case.

References

[A+22] H. M. Ardevol, J. B. Melé, D. Lubarov, J. L. Muñoz-Tapia, RapidUp: Multi-Domain Permutation Protocol for Lookup Tables, Cryptology ePrint Archive, Report 2022/1050, 2022, https://eprint.iacr.org/2022/1050.

[BS+13] E. Ben-Sasson, A. Chiesa, E. Tromer, M. Virza, Succinct non-interactive arguments for a von neumann architecture, Cryptology ePrint Archive, Report 2013/879, 2013, https://eprint.iacr.org/2013/879.

[B+18] J. Bootle, A. Cerulli, J. Groth, S. Jakobsen, M. Maller, Arya: Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution, Cryptology ePrint Archive, Report 2018/380, 2018, https://eprint.iacr.org/2018/380.

[Cer19] Efficient Zero-Knowledge Proofs and their Applications, PhD thesis, UCL, 2019.

[G+20] A. Gabizon, Z. J. Williamson, plookup: A simplified polynomial protocol for lookup tables, Cryptology ePrint Archive, Report 2020/315, 2020, https://eprint.iacr.org/2020/315.

[GMR85] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof-systems (extended abstract), in: Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA, pp 291–304, 1985.

[P+22] L. Pearson, J. Fitzgerald, H. Masip, M. Bellés-Muñoz, J. L. Muñoz-Tapia, PlonKup: Reconciling PlonK with plookup, Cryptology ePrint Archive, Report 2022/086, 2022, https://eprint.iacr.org/2022/086.

Adapt Framework solutions
Email address: `alex@adaptframework.solutions`
Email address: `timdok@gmail.com`