A Brief History of Lookup Arguments

Tomer Solberg
tomer@ingonyama.com

1 Introduction

zk-SNARKs have evolved in recent years focusing mostly on making them more succinct and lowering prover and verifier time. However most SNARKs are still limited to arithmetic operations that can easily be converted to polynomials. Such operations are generally referred to as "SNARK-friendly", while the other "SNARK-unfriendly" operations are left mostly undealt with.

Lookup protocols were invented [BCG+18] to deal with some of those SNARK-unfriendly operations. In general, they offer a way to prove the following statement:

"Given a table $T = \{t_i\}_{i=0,\ldots,N-1}$ of distinct values ("rows"),
and a list of lookups $F = \{f_j\}_{j=0,\ldots,m-1}$ (possibly with multiplicities),
the lookups are all contained in the table $F \subseteq T$."

The table will generally be treated as public, while the lookups will be treated as a private witness. One can think of the table as containing all legal values of a certain variable, and the lookups as the values of this variable given from a certain program execution. The statement then means that the variable maintained legal status throughout the execution. We will assume $m < N$ (and mostly $m \ll N$) unless otherwise stated.

In this work we will consider various uses of proving these kind of statements, review the different protocols available and their evolution. We focus mostly on plookup [GW20] and cq [EFG22], while briefly reviewing the chain of ideas that led from one to the other. We aim to present the protocols and the ideas that they are based upon in a readable and accessible way.

2 Uses of Lookup Protocols

2.1 Range Checks

Say we wish to check that a number x is in the range $\{0, \ldots, N-1\}$ for $N = 2^n$. The arithmetic constraint way would be to define n numbers $b_0 \ldots, b_{n-1}$ and check that for each i, $b_i \in \{0, 1\}$ and that $\sum_i b_i 2^i = x$. This takes $n+1$ constraints, and to check m such numbers we would need $O(mn) = O(m \log N)$ constraints. This can be done in a single lookup constraint which would check for the entire set of lookups is in a table consisting of all numbers in the range. We will see the costs of such a constraint in the next sections. Notice that the constraint approach also gets cumbersome when N is not a power of 2, while the lookup approach does not.

2.2 Finite Domain Functions

Lookups can also be used to implement any (finite domain) function, by simply defining the table as the full list of input and output values of the function. This can be used to implement functions of any number of variables. One example is k-bit XOR, which is a function used widely in hash computations. Following the logic of §2.1, Using arithmetic constraints we would need $6k$ constraints to implement it. Alternatively, it can be naively implemented using a table T, whose rows are

$$t_i = (A, B, C), \tag{2.1}$$

where for each i, $A, B \in \{0, \dots, 2^k - 1\}$ is a distinct combination of two k-bit numbers, and $C = A \oplus B$. The full table will then have 2^{2k} rows, where each row takes $3k$ bits to store. For $k = 32$ which is the value commonly used in many hash functions, such a table is of course completely impractical. Even for $k = 16$ this table will take 24GB of storage, which is impractical for most applications.

2.3 Better XOR

A more efficient way to implement bit-wise XOR with lookup tables is by zero-interleaving [Hal]. Zero-interleaving is a function which takes the binary representation a_l of a number

$$A = \sum_{l=0}^{k-1} a_l 2^l, \tag{2.2}$$

and adds a '0' bit between any two original bits, so that we get

$$A' = \sum_{l=0}^{k-1} a_l 4^l. \tag{2.3}$$

After zero-interleaving both inputs $A, B \to A', B'$, we can simply add the two numbers to get C'', which has the bits of the required output C in the even bit positions. To convert back from C'' to C, break C'' into even and odd bits

$$C'' = \sum_{l=0}^{k-1} c_l^{even} 4^l + 2 \sum_{l=0}^{k-1} c_l^{odd} 4^l. \tag{2.4}$$

Then c_l^{even} is the bit representation of C, and as a byproduct we get c_l^{odd} which is the bit representation for $D = A \wedge B$ where $\wedge$ represents bit-wise AND. We can then prove both the bit-wise XOR and bit-wise AND results of A, B, by using zero-interleaving table 4 times: for $A, B, C, D \to A', B', C', D'$, along with the arithmetic constraint

$$A' + B' = C'' = C' + 2D'. \tag{2.5}$$

For $k = 32$, this implementation is still impractically large, with a single table taking 48GB of storage. However for $k = 16$ this implementation of the lookup table requires only 384kB. Lowering the number of bits can be done in a SNARK-friendly way by slicing, i.e. introducing an arithmetic gate which takes, say, 2 16-bit numbers and adds them together with constant weights $1, 2^{16}$ to get a 32-bit number.

2.4 Finite State Machines

Another use of lookup tables is to implement finite state machines. A state machine is composed of a set of states, and transitions between the states depending on an input. A table implementing the state machine can contain all legal combinations of (current state, input, next state). An execution trace of a state machine can be generally put in the form

$$(state(j), input(j), next\_state(j)), \tag{2.6}$$

which can be proven to be a legal execution of the state machine using a lookup protocol, together with proving the wiring constraint

$$next\_state(j) = state(j+1), \tag{2.7}$$

which is SNARK-friendly.

3 Plookup

Plookup [GW20] is one of the earliest lookup protocols, and it is in fact a simplification of what is regarded as the first lookup protocol. Plookup is based on the idea that given the vectors $t \in \mathbb{F}^N, f \in \mathbb{F}^m, s \in \mathbb{F}^{N+m}$ and the bivariate polynomials

$$F(\beta,\gamma) := (1+\beta)^m \prod_{j=1}^{m}(\gamma + f_j) \prod_{i=1}^{N-1}(\gamma(1+\beta) + t_i + \beta t_{i+1}) \tag{3.1}$$

$$G(\beta,\gamma) := \prod_{k=1}^{m+N-1}(\gamma(1+\beta) + s_k + \beta s_{k+1}). \tag{3.2}$$

Then the following holds

$$F \equiv G \Leftrightarrow \begin{cases} \{f_j\} \subseteq \{t_i\}, \text{ AND} \\ s = (f,t) \text{ sorted by } t, \end{cases} \tag{3.3}$$

where by $s = (f,t)$ sorted by t we mean that values in s appear in the same order they appear in t (notice that this is well defined since $\{f_j\} \subseteq \{t_i\}$).

The reason this is true is that if $\{f_j\} \subseteq \{t_i\}$ and $s = (f,t)$ sorted by t then for each $i = 1,\ldots,N-1$ there is a distinct $k \in \{1,\ldots,m+N-1\}$ such that

$$(\gamma(1+\beta) + t_i + \beta t_{i+1}) = (\gamma(1+\beta) + s_k + \beta s_{k+1}), \tag{3.4}$$

and for the other indices k $s_k = s_{k+1}$ so for each of those there is a $j \in \{1,\ldots,m\}$ such that $f_j = s_k$ and

$$(1+\beta)(\gamma + f_j) = (\gamma(1+\beta) + s_k + \beta s_{k+1}). \tag{3.5}$$

The other direction is proved using the fact that F, G are polynomials in $\mathbb{F}[\gamma]$ (with β treated as a coefficient) so they have a unique factorization. After identifying the factors as is done in (3.4),(3.5), one can use the fact that the factors are polynomials of the variable β to identify the elements of f, t with those of s.

3.1 Definitions

Plookup uses the following definitions:

- m is taken to be one smaller than N. If either m, N are too small to satisfy $N = m+1$ then the corresponding table is padded with repetitions of the last element until it is satisfied.
- $H = \{g, \ldots, g^N = 1\}$ is a multiplicative subgroup of order N in $\mathbb{F}$.
- For a vector $p \in \mathbb{F}^N$, define the polynomial $p(x) \in \mathbb{F}[X]_{<N}$ such that the vector elements are its evaluations $p_i = p(g^i)$.
- $L_i(x) \in \mathbb{F}[X]_{<N}$ is the i-th Lagrange polynomial on H, satisfying $L_i(g^j) = \delta_{ij}$ (the Kronecker delta).
- $s \in \mathbb{F}^{2N-1}$ is (f, t) sorted by t.

3.2 The Protocol

The resulting protocol is as follows:

1. The prover computes and commits two polynomials $h_1, h_2 \in \mathbb{F}[x]_{<N}$, such that for each $i = 1, \ldots, N$

$$h_1(g^i) = s_i \tag{3.6}$$

$$h_2(g^i) = s_{N+i-1}. \tag{3.7}$$

2. The verifier sends random challenges β, γ to the prover.

3. The prover computes and commits a polynomial $Z \in \mathbb{F}[x]_{<N}$ that aggregates $F(\beta,\gamma)/G(\beta,\gamma)$ as follows

$$Z(g) = 1 \tag{3.8}$$

$$Z(g^i) = \frac{(1+\beta)^{i-1} \prod_{l=1}^{i-1}(\gamma + f_l)(\gamma(1+\beta) + t_l + \beta t_{l+1})}{\prod_{l=1}^{i-1}(\gamma(1+\beta) + s_l + \beta s_{l+1})(\gamma(1+\beta) + s_{N+l} + \beta s_{N+l+1})}, \quad i = 2, \ldots, N-1 \tag{3.9}$$

$$Z(g^N) = 1. \tag{3.10}$$

4. The verifier checks the following identities for all $x \in H$

$$L_1(x)(Z(x) - 1) = 0 \tag{3.11}$$

$$L_N(x)(Z(x) - 1) = 0 \tag{3.12}$$

$$L_N(x)(h_1(x) - h_2(gx)) = 0 \tag{3.13}$$

$$\begin{aligned}&(x - g^N)Z(x)(1+\beta)(\gamma + f(x))(\gamma(1+\beta) + t(x) + \beta t(gx)) \\ &= (x - g^N)Z(gx)(\gamma(1+\beta) + h_1(x) + \beta h_1(gx))(\gamma(1+\beta) + h_2(x) + \beta h_2(gx)).\end{aligned} \tag{3.14}$$

Note that the polynomial $Z(x)$ is constructed such that each evaluation adds another multiplicative term to the products in both the numerator and the denominator of (3.9). This means that as the N-th term is added, equality of F, G implies that all terms cancel and we must get 1. The checks of (3.11,3.12) test that in fact $Z(g) = Z(g^N) = 1$, while the check of (3.14) test that indeed each evaluation adds the correct multiplicative term (the factor of $(x - g^N)$ multiplying both sides makes sure that this process does not include a relation between $Z(g^N)$ and $Z(g^{N+1}) = Z(g)$).

3.3 Costs

Note that this protocol does not rely on any particular commitment scheme. In general, the prover runtime in the protocol is $O(N\log N)$ field operations (to construct the polynomials from their evaluations), and $O(N)$ group operations (to construct the polynomial Z). To specify proof size and verifier runtime we assume that it uses KZG commitments, and then it can be shown that the proof size can be 5 group elements and 9 field elements, while the verifier runtime is 2 pairing functions.

3.4 Generalizations and Optimizations

Plookup protocol can be generalized to the case of several witness polynomials $f_1, \dots, f_w \in \mathbb{F}[x]_{<m}$ and several tables $t_1, \dots, t_w \in \mathbb{F}[x]_{<N}$. To do so, the verifier chooses a random number α, and then the polynomials are aggregated by

$$t = \sum_{\ell=1}^{w} \alpha^\ell t_\ell \tag{3.15}$$

$$f = \sum_{\ell=1}^{w} \alpha^\ell f_\ell, \tag{3.16}$$

and the protocol proceeds with f, t as before.

The protocol can be optimized if the table is simply a range of consecutive integers. In this case $t_{l+1} = t_l + 1$ and (3.9) can be simplified to

$$Z(g^i) = \frac{\prod_{l=1}^{i-1}(\gamma + f_l)(\gamma + t_l)}{\prod_{l=1}^{i-1}(\gamma + s_l)(\gamma + s_{N+l})}, \tag{3.17}$$

with the verifier checks adjusted accordingly.

Another generalization is the plonkup protocol [PFM$^+$22] which suggests an integration between plonk and plookup, allowing for efficient inclusion of lookup tables as generalized plonk gates.

4 cq Protocol

The main problem with plookup is how costly it is for the prover in the common case where $m << N$. In the few years since its publication, there has been a steady evolution in lookup protocols, each depending less strongly on N than the previous, with the main idea behind them is to move most of the table calculations to precomputation. As of the publishing of this work, the state-of-the-art protocol in this sense is cq [EFG22].

4.1 Logarithmic Derivative

The cq protocol is based on the following trick, known as logarithmic derivative: two polynomials $p(x) = \prod_{a\in A}(x+a)$ and $q(x) = \prod_{b\in B}(x+b)$ are equal if and only if the rational functions

$$\frac{p'(x)}{p(x)} = \sum_{a\in A}\frac{1}{x+a} \tag{4.1}$$

$$\frac{q'(x)}{q(x)} = \sum_{b\in B}\frac{1}{x+b} \tag{4.2}$$

are equal. The forward direction ($p(x) = q(x) \Rightarrow p'(x)/p(x) = q'(x)/q(x)$) is trivial. To prove the backward direction, notice that if $p'(x)/p(x) = q'(x)/q(x)$ then

$$\left(\frac{p(x)}{q(x)}\right)' = \frac{p'(x)q(x) - q'(x)p(x)}{p^2(x)} = 0. \tag{4.3}$$

This means that $p(x)/q(x) = c$ for some constant c. However since both $p(x), q(x)$ have a leading coefficient equal to 1, then $c = 1$ and $p(x) = q(x)$. We get that the set of lookups f is contained in the table t if and only if

$$\sum_{i=1}^{N}\frac{m_i}{x+t_i} = \sum_{j=1}^{m}\frac{1}{x+f_j}, \tag{4.4}$$

where m_i is the multiplicity of the value t_i in the lookup f_j (recall that each t_i is unique, but f_j's are allowed to repeat, and of course m_i's can be 0 for many t_i's). The idea of the protocol is to test this rational function identity on some random point $x = \beta$.

4.2 Reformulating Into Sumcheck

The way cq tests this identity is by defining polynomials $A(x), B(x)$ such that their evaluations are

$$A_i = \frac{m_i}{\beta + t_i}\ ,\ i = 1,\dots,N \tag{4.5}$$

$$B_j = \frac{1}{\beta + f_j}\ ,\ j = 1,\dots,m. \tag{4.6}$$

Here $A_i = A(g^i)$ where g is a generator of a multiplicative subgroup $V \subset F$ of order N ($V = \{g, g^2, \dots, g^N = 1\}$), and $B_j = B(\omega^j)$ where ω is a generator of a multiplicative subgroup $H \subset F$ of order m ($H = \{\omega, \omega^2, \dots, \omega^m = 1\}$). To satisfy the required relationship (4.4) on the random point β, A_i, B_j need to satisfy $\sum_i A_i = \sum_j B_j$. As these are the evaluations of polynomials that are defined over multiplicative groups, they obey

$$\sum_{i=1}^{N} A_i = N \cdot A(0) \tag{4.7}$$

$$\sum_{j=1}^{m} B_j = m \cdot B(0). \tag{4.8}$$

The prover then must prove

$$N \cdot A(0) = m \cdot B(0). \tag{4.9}$$

This is a univariate sumcheck problem.

4.3 Quotient Polynomials

We now look at the polynomials

$$p(x) = A(x)(T(x) + \beta) - m(x) \tag{4.10}$$

$$q(x) = B(x)(F(x) + \beta) - 1, \tag{4.11}$$

where $T(x), m(x), F(x)$ are the polynomials whose evaluations are t_i, m_i, f_j. We see that $p(x)$ must evaluate to zero on V, and similarly $q(x)$ must evaluate to zero on H. We get that we can define the quotient polynomials $Q_A(x), Q_B(x)$ such that

$$Q_A(x) = \frac{A(x)(T(x) + \beta) - m(x)}{Z_V(x)} \tag{4.12}$$

$$Q_B(x) = \frac{B(x)(F(x) + \beta) - 1}{Z_H(x)}, \tag{4.13}$$

where $Z_V(x), Z_H(x)$ are the vanishing polynomials on V, H. The prover now needs to prove two things:

- That they know the polynomials A, B, Q_A, Q_B, F, m
- That the polynomials obey the required relationships (4.9,4.12,4.13)

This is done using KZG commitments, which can then be checked using pairing. We will break these statements down in the following subsections.

4.4 Proving Knowledge of the Polynomial A and its Sum

Recall that to prove knowledge of a polynomial $\phi(x)$ using a KZG commitment to it $[\phi(x)]_1$, the prover evaluates it at a point z, defines a new polynomial

$$P_\phi = \frac{\phi(x) - \phi(z)}{x - z}, \tag{4.14}$$

and then sends a commitment to it $[P_\phi]_1$. The verifier can then test this using the pairing equation

$$e([\phi(x)]_1 - [\phi(z)]_1, [1]_2) = e([P_\phi]_1, [x - z]_2), \tag{4.15}$$

which we rewrite for convenience as

$$e([\phi(x)]_1 - [\phi(z)]_1 + z \cdot [P_\phi]_1, [1]_2) = e([P_\phi]_1, [x]_2), \tag{4.16}$$

so that the second argument of the pairing function is always either $[1]_2$ or $[x]_2$. To prove knowledge of the polynomial A we evaluate it at the point $x = 0$, which as we have seen gives its sum. The prover defines

$$P_A(x) = \frac{A(x) - A(0)}{x} \tag{4.17}$$

and commits $[P_A]_1$, which is then proved using

$$e([A(x)]_1 - [A(0)]_1, [1]_2) = e([P_A(x)]_1, [x]_2). \tag{4.18}$$

4.5 Low Degree Testing of $B(x)$

To prove knowledge of $B(x)$ we first prove that it is indeed of degree $< m$. Note that we do not need to prove the degree of $A(x)$ as it is the maximum degree allowed by the SRS. The way this is done in KZG is by defining

$$P_B(x) = \frac{B(x) - B(0)}{x}, \tag{4.19}$$

then committing $[P_B(x) \cdot x^{N-m+1}]_1$ and finally testing

$$e([P_B(x)]_1, [x^{N-m+1})]_2) = e([P_B(x) \cdot x^{N-m+1}]_1, [1]_2). \tag{4.20}$$

Notice that if $B(x)$ had terms of degree $\geq m$, these would have wrap around and appear as low degree terms in the commitment $[P_B(x) \cdot x^{N-m+1}]_1$, as the SRS does not allow committing of terms of degree $\geq N$.

4.6 Proving the Relation (4.12) with Cached Quotients

This relation can be checked using

$$e([A(x)]_1, [T(x)]_2) = e([Q_A(x)]_1, [Z_V(x)]_2) \cdot e([m(x)]_1 - \beta[A(x)]_1, [1]_2). \tag{4.21}$$

Here, $[T]_2, [Z_V]_2, [1]_2, [x]_2$ are all independent on the lookups and can then be precomputed. The prover needs to compute $[A(x)]_1, [Q_A(x)]_1, [m(x)]_1, A(0), \left[\frac{A(x)-A(0)}{x}\right]_1$. Computing nearly all of these can be done in $O(m)$ since they only need to be computed for the lookups (if t_i is not in the lookups then $m_i, A_i = 0$). The problem is computing $[Q_A(x)]_1$ as this is $O(N)$. The solution is to precompute the *cached quotients*

$$Q_i(x) = \frac{L_i(x)(T(x) - t_i)}{Z_V(x)} = \frac{T(x) - t_i}{k^i(x - g^i)}, \tag{4.22}$$

where $L_i(x)$ are the Lagrange polynomials and

$$L_i(x) = \frac{Z_V(x)}{k^i(x - g^i)} \tag{4.23}$$

$$k^i = Z_V'(g^i) = (x^N - 1)'|_{x=g^i} = N \cdot g_i^{N-1} = \frac{N}{g^i}. \tag{4.24}$$

$Q_i(x)$ are computed in $O(N\text{log}N)$ as an NTT, and with them $[Q_A]_1$ can be computed in $O(m)$ by

$$[Q_A(x)]_1 = \sum_{A_i \neq 0} A_i \cdot [Q_i(x)]_1, \tag{4.25}$$

where A_i are the evaluations of the polynomials $A(x)$ as defined in (4.5).

4.7 Proving the Relations (4.9,4.13)

The last relations are proved by introducing another random point $x = \gamma$. First, we prove knowledge of $F(\gamma)$ and $P_B(\gamma)$ by committing the polynomials

$$P_F(x) = \frac{F(x) - F(\gamma)}{x - \gamma} \tag{4.26}$$

$$P_{B,\gamma}(x) = \frac{P_B(x) - P_B(\gamma)}{x - \gamma}. \tag{4.27}$$

This is verified using

$$e([F(x)]_1 - [F(\gamma)]_1 + \gamma[P_F(x)]_1, [1]_2) = e([P_F(x)]_1, [x]_2) \tag{4.28}$$
$$e([P_B(x)]_1 - [P_B(\gamma)]_1 + \gamma[P_{B,\gamma}(x)]_1, [1]_2) = e([P_{B,\gamma}(x)]_1, [x]_2). \tag{4.29}$$

This proves that indeed $B(x), F(x)$ evaluate to $B(\gamma), F(\gamma)$, so we can use these evaluations to prove the desired relations. We first define

$$Q_{b,\gamma} = \frac{(B(\gamma) - B(0) + A(0) \cdot N/m)(F(\gamma) + \beta) - 1}{Z_H(\gamma)} \tag{4.30}$$
$$P_{Q_B}(x) = \frac{Q_B(x) - Q_{b,\gamma}}{x - \gamma}, \tag{4.31}$$

and then send their commitments $[Q_{b,\gamma}]_1, [P_{Q_B}(x)]_1$, so that they can be verified by using the pairing equation

$$e[Q_B(x)]_1 - [Q_{b,\gamma}]_1 + \gamma[P_{Q_B}(x)]_1, [1]_2) = e([P_{Q_B}(x)]_1, [x]_2). \tag{4.32}$$

Note that this construction proves both of the following statements

$$Q_B(\gamma) = \frac{(B(\gamma))(F(\gamma) + \beta) - 1}{Z_H(\gamma)} \tag{4.33}$$
$$B(0) \cdot m = A(0) \cdot N, \tag{4.34}$$

thus finalizing our proofs.

4.8 Proof Batching

The three last proofs have a very similar structure. The cq protocol batches them into a single proof by introducing a new random variable η and defining

$$c(x) = P_B(x) + \eta F(x) + \eta^2 Q_B(x) \tag{4.35}$$
$$v = P_B(\gamma) + \eta F(\gamma) + \eta^2 Q_{b,\gamma} \tag{4.36}$$
$$P_\gamma(x) = P_{B,\gamma}(x) + \eta P_F(x) + \eta^2 P_{Q_B}(x). \tag{4.37}$$

Then, the checks (4.29,4.28,4.32) aggregate to a single check

$$e([c(x)]_1 - [v]_1 + \gamma[P_\gamma(x)]_1, [1]_2) = e([P_\gamma(x)]_1, [x]_2). \tag{4.38}$$

4.9 The Full Protocol

4.9.1 Setup Phase

Both the prover and verifier receive as input t_i for $i = 1, \ldots, N$. The following steps are then performed by a trusted party

1. Choose a random $x \in \mathbb{F}$ and output $\{[x^i]_1\}_{i=0}^{N-1}$ and $\{[x^i]_2\}_{i=1}^{N}$. The number x must then be deleted.

2. Compute and output $[Z_V(x)]_2$.

3. Compute $T(x) = \sum_i t_i L_i(x)$. Compute and output $[T(x)]_2$.

4. For each $i = 1, \ldots, N$ compute and output

   – $[Q_i(x)]_1$ according to (4.22),

   – $[L_i(x)]_1$,

   – $\left[\frac{L_i(x) - L_i(0)}{x}\right]_1$.

The outputs of this phase constitute the SRS, which are sent to both the prover and the verifier.

4.9.2 Proving Phase

The prover receives as input the private witness values f_j for $j = 1, \ldots, m$, to which the verifier receives as input the commitment $[F(x)]_1$ (the input must be received from a trusted source for the prover and the verifier to agree on the problem that needs to be proven). The following protocol then takes place

Prover	Communication	Verifier
Compute $m_i, m(x)$	$[m(x)]_1 \longrightarrow$	
	$\longleftarrow \beta$	Choose random $\beta \in \mathbb{F}$
Compute $A(x), Q_A(x), B(x), Q_B(x)$	$[A(x)]_1, [Q_A(x)]_1, [Q_B(x)]_1, [P_A(x)]_1,$ $[P_B(x)]_1, [P_B(x) \cdot x^{N-m+1}]_1 \longrightarrow$	
	$\longleftarrow \gamma, \eta$	Verify (4.20,4.21) Choose random $\gamma, \eta \in \mathbb{F}$
Compute $P_B(\gamma), F(\gamma), A(0), P_\gamma(x)$	$P_B(\gamma), F(\gamma), A(0),$ $[P_\gamma(x)]_1 \longrightarrow$	
		Compute $[c(x)]_1, Q_{b,\gamma}, [v]_1$ Verify(4.18,4.38)

Table 1: Proving phase of the cq protocol

Note that the verifier computes $Q_{b,\gamma}$ which is defined in (4.30). To do so she supposedly needs $B(\gamma), B(0)$ which are not sent by the prover. However, $P_B(\gamma) = (B(\gamma) - B(0))/\gamma$ is sent by the prover, so the verifier computes $Q_{b,\gamma}$ using

$$Q_{b,\gamma} = \frac{(P_B(\gamma) \cdot \gamma + A(0) \cdot N/m)(F(\gamma) + \beta) - 1}{Z_H(\gamma)}. \tag{4.39}$$

4.10 Further Batching and Aggregation

In general, one would use the Fiat-Shamir transformation to render this protocol non-interactive. In the case, the proof would contain

$$\pi_{cq} = \{[m]_1, [A]_1, [Q_A]_1, [Q_B]_1, [P_A]_1, [P_B]_1, [P_B x^{N-m+1}]_1, [P_\gamma]_1, P_B(\gamma), F(\gamma), A(0)\}, \tag{4.40}$$

where the first eight members are group elements, and the last three are field elements. The pairing equations are

$$e([P_B(x)]_1, [x^{N-m+1)}]_2) = e([P_B(x) \cdot x^{N-m+1}]_1, [1]_2) \tag{4.41}$$

$$e([A(x)]_1, [T(x)]_2) = e([Q_A(x)]_1, [Z_V(x)]_2) \cdot e([m(x)]_1 - \beta[A(x)]_1, [1]_2) \tag{4.42}$$

$$e([A(x)]_1 - [A(0)]_1, [1]_2) = e([P_A(x)]_1, [x]_2) \tag{4.43}$$

$$e([c(x)]_1 - [v]_1 + \gamma[P_\gamma(x)]_1, [1]_2) = e([P_\gamma(x)]_1, [x]_2). \tag{4.44}$$

The last two can be easily batched by introducing a new random number $\mu \in \mathbb{F}$ to get

$$e([c(x)]_1 - [v]_1 + \gamma[P_\gamma(x)]_1 + \mu([A(x)]_1 - [A(0)]_1), [1]_2) = e([P_\gamma(x)]_1 + \mu[P_A(x)]_1, [x]_2), \tag{4.45}$$

which can by batched with the first pairing equation by introducing $\rho \in \mathbb{F}$ to get

$$\begin{aligned} e([P_\gamma(x)]_1 + \mu[P_A(x)]_1, [x]_2) \cdot e(\rho[P_B(x)]_1, [x^{N-m+1)}]_2) = \\ = e([c(x)]_1 - [v]_1 + \gamma[P_\gamma(x)]_1 + \mu([A(x)]_1 - [A(0)]_1) + \rho[P_B(x) \cdot x^{N-m+1}]_1, [1]_2). \end{aligned} \tag{4.46}$$

This can now be batched with the remaining equation by introducing $\sigma \in \mathbb{F}$ to get a single pairing equation. Define

$$L_a = [P_\gamma(x)]_1 + \mu[P_A(x)]_1 \tag{4.47}$$

$$L_b = \rho[P_B(x)]_1 \tag{4.48}$$

$$L_c = \sigma[A(x)]_1 \tag{4.49}$$

$$L_d = -\sigma[Q_A(x)]_1 \tag{4.50}$$

$$\begin{aligned} R = [c(x)]_1 - [v]_1 + \gamma[P_\gamma(x)]_1 + \mu([A(x)]_1 - [A(0)]_1) \\ + \rho[P_B(x) \cdot x^{N-m+1}]_1 + \sigma([m(x)]_1 - \beta[A(x)]_1). \end{aligned} \tag{4.51}$$

Then our pairing equation will be

$$e(L_a, [x]_2) \cdot e(L_b, [x^{N-m+1)}]_2) \cdot e(L_c, [T(x)]_2) \cdot e(L_d, [Z_V(x)]_2) = e(R, [1]_2). \tag{4.52}$$

Now, since the second argument of each pairing function depends only on the setup, different proofs with the same setup can be aggregated into a single proof by introducing a new random parameter $\chi \in \mathbb{F}$

$$\begin{aligned} e(\sum_k \chi^k L_{a,k}, [x]_2) \cdot e(\sum_k \chi^k L_{b,k}, [x^{N-m+1}]_2) \cdot \\ e(\sum_k \chi^k L_{c,k}, [T(x)]_2) \cdot e(\sum_k \chi^k L_{d,k}, [Z_V(x)]_2) = e(\sum_k \chi^k R_k, [1]_2). \end{aligned} \tag{4.53}$$

4.11 Costs

Unlike plookup, this protocol relies heavily on the KZG commitment scheme, and we analyze its cost in this scheme. The protocol shifts the $O(N\log N)$ complexity of plookup into the setup phase, so that these calculations only occur once per table. The proving phase calculations all depend on m, which we assume to be much smaller then N (if this is not the case, there is no reason to use cq as it will be outperformed by plookup). The prover work is then $O(m\log m)$, while proof size and verifier work are constant.

5 Evolution of lookup protocols

In this section we give a brief summary of the improvements that led from plookup to cq. We review the main protocols that came between them and their costs. Note that all of these protocols rely explicitly on KZG commitments.

5.1 Caulk

The main idea behind Caulk [ZBK+22], is to precompute an encoding of the table (in $O(N \log N)$) such that it is searchable in $O(\log N)$. Lookups can then be similarly encoded, so that it takes $O(m \log N)$ to look them up, and another $O(m^2)$ to prove the encoding. In more details, the table is precomputed as a vanishing polynomial

$$Z_T(x) = \prod_{i=1}^{N}(x - t_i), \tag{5.1}$$

and the lookups are similarly computed

$$f(x) = \prod_{j=1}^{m}(x - f_j). \tag{5.2}$$

The prover then sends commitments to Z_T, f, and the idea is to prove that $f = Z_S$ for some $S \subseteq T$, which can be done by proving that $Z_{T\setminus S}(x) = Z_T(x)/Z_S(x)$ is a polynomial. The way this is done in caulk is by also precomputing for each $i = 1, \ldots, N$ the polynomials $g_i(x) = Z_{T\setminus t_i}(x)$. In fact, these are basically the same cached quotients that give cq its name. Then, if $S \subseteq T$ then there exists a decomposition

$$Z_{T\setminus S}(x) = \sum_{j=1}^{m} c_j g_{i(j)}(x) \tag{5.3}$$

for some numbers $c_j \in \mathbb{F}$. The prover then needs to compute the commitment to this polynomial which the verifier can check using pairing:

$$e([f], [Z_{T\setminus S}]) = e([Z_T], [1]). \tag{5.4}$$

5.2 Caulk+

A main disadvantage of Caulk is that it treated the table as a generic set, rather than encoding it as some group. Caulk+ [PK22] improves upon that by precomputing the table so it can be treated as a multiplicative group, with the original values being the powers of the group generator. This eliminated N dependence from the complexity of the computation, so the prover runtime was just $O(m^2)$.

5.3 Flookup

Flookup [GK22] was able to reduce the prover runtime to $O(m \log^2 m)$ at the cost of increasing the preprocessing cost to $O(N \log^2 N)$, and also losing the nice property that the commitment is homomorphic (thus not allowing collection of several lookups and tables into a single run of the protocol as was shown for lookup).

Protocol	Preprocessing	Prover work	Verifier work	proof size	Hom.	Agg.
Plookup	-	$O(N\log N)\mathbb{F} + O(N)G_1$	$2P$	$9\mathbb{F}, 5G_1$	Yes	No
Caulk	$O(N\log N)$	$O(m\log N + m^2)\mathbb{F} + O(m)G_1$	$4P$	$4\mathbb{F}, 14G_1, 1G_2$	Yes	No
Caulk+	$O(N\log N)$	$O(m^2)\mathbb{F} + O(m)G_1$	$3P$	$2\mathbb{F}, 7G_1, 1G_2$	Yes	No
Flookup	$O(N\log^2 N)$	$O(m\log^2 m)\mathbb{F} + O(m)G_1$	$3P$	$4\mathbb{F}, 7G_1, 1G_2$	No	No
Baloo	$O(N\log N)$	$O(m\log^2 m)\mathbb{F} + O(m)G_1$	$5P$	$4\mathbb{F}, 12G_1, 1G_2$	Yes	No
cq	$O(N\log N)$	$O(m\log m)\mathbb{F} + O(m)G_1$	$5P$	$3\mathbb{F}, 8G_1$	Yes	Yes

Table 2: Comparing the different lookup protocols

5.4 Baloo

Baloo [ZGK+22] is an optimized version of Caulk+, which replaces the lookups with a linear variant. It then uses a univariate sumcheck to prove its committed polynomials. It keeps the $O(N\log N)$ preprocessing cost of Caulk+, while reducing the prover cost to $O(m\log^2 m)$ - the same as flookup. While its actual cost is slightly higher than flookup due to larger constants, it retains the homomorphic nature of its commitments.

5.5 Summary

The overall comparison of the different lookup protocols is summarized in table 2.

References

[BCG+18] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. Nearly linear-time zero-knowledge proofs for correct program execution. Cryptology ePrint Archive, Paper 2018/380, 2018. `https://eprint.iacr.org/2018/380`.

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Paper 2022/1763, 2022. `https://eprint.iacr.org/2022/1763`.

[GK22] Ariel Gabizon and Dmitry Khovratovich. flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. Cryptology ePrint Archive, Paper 2022/1447, 2022. `https://eprint.iacr.org/2022/1447`.

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

[Hal] Halo2. Halo2 documentation. `https://zcash.github.io/halo2/design/gadgets/sha256/table16.html`.

[PFM+22] Luke Pearson, Joshua Fitzgerald, Héctor Masip, Marta Bellés-Muñoz, and Jose Luis Muñoz-Tapia. Plonkup: Reconciling plonk with plookup. Cryptology ePrint Archive, Paper 2022/086, 2022. `https://eprint.iacr.org/2022/086`.

[PK22] Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Paper 2022/957, 2022. `https://eprint.iacr.org/2022/957`.

[ZBK$^+$22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. Cryptology ePrint Archive, Paper 2022/621, 2022. https://eprint.iacr.org/2022/621.

[ZGK$^+$22] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments. Cryptology ePrint Archive, Paper 2022/1565, 2022. https://eprint.iacr.org/2022/1565.