Marlin & Me

Karthik Inbasekar
karthik@ingonyama.com

1 Introduction

In this article, we dive deep into the heart of the Marlin protocol introduced in [1]. The goal of the document is to provide intuition in the various technical steps involved in the protocol with necessary explanation wherever possible. We will assume some familiarity with protocols such as Groth16 [2, 3]. We believe the article is most useful for readers who have gone through the Marlin paper [1], and if not we recommend a quick reading of the paper before going over this article.

In constructing a proof verification scheme, it is generally assumed that both the prover and the verifier must possess knowledge of the circuit or the compute statement. The compute statement is expressed as a Circuit Satisfiability condition (CSAT)

$$C(z) = 1 \tag{1.1}$$

such that $z = [x||w]$ satisfies it, where x is the public input and w is a private witness. In general, the proof system for the CSAT relation consists of an offline phase called setup which needs to be run in a trusted setting with the help of a secure elaborate MPC (MultiParty Computation) ceremony for a given circuit. The output of the MPC is a string of Elliptic curve group elements called as the SRS (Structured Reference String) which depends on the specific circuit in consideration. This is for example the case in [2]. The main limitation of this approach is that when the circuit changes, even by a small amount, the MPC ceremony needs to be rerun making it non-universal.

One of the key issues addressed by Marlin, is to address the complexity in the offline phase of the circuit dependent processing. Up to a predetermined circuit size bound, a universal and updateable SRS is generated via an MPC. Here updateable means that the SRS can be updated by any MPC participant and can be run at any time [4]. The security of the SRS is guaranteed as long as there is one honest participant in the MPC ceremony since the beginning of the first one. The circuit size bound appears because the universal SRS is a public data that takes the form

$$\Sigma := \begin{pmatrix} G & \beta G & \beta^2 G & \dots & \beta^D G \\ \gamma G & \gamma\beta G & \gamma\beta^2 G & \dots & \gamma\beta^D G \end{pmatrix} \tag{1.2}$$

where $\beta, \gamma \in \mathbb{F}_q$ and G is a group element in the Elliptic Curve group $\mathbb{G}_1$. Thus the public parameters can support any polynomial upto a maximum degree D, which in turn bounds the maximum size of the circuit supported by the SRS.

For a given circuit, in the offline setup phase, the universal SRS is used to generate a circuit specific SRS for any circuit satisfiability problem up that satisfies degree bound required in the MPC cermony. The most important contribution by Marlin is that there is

Equation	R1CS vector a_i	R1CS vector b_i	R1CS vector c_i
$x_1.x_1 = u$	[0 0 1 0 0 0]	[0 0 1 0 0 0]	0 0 0 0 1 0]
$u.x_2 = v$	[0 0 0 0 1 0]	[0 0 0 1 0 0]	0 0 0 0 0 1]
$c.(c + x_1 + v) = d$	[1 0 0 0 0 0]	[1 0 1 0 0 1]	[0 1 0 0 0 0]

Table 1: R1CS constraint system for (2.2)

no need to rerun the MPC ceremony everytime the circuit is changed or if someone updates the SRS. In this sense, Marlin achieves universal SRS for any circuit bounded by degree D.

In the rest of the article, we focus mainly on technical steps leading to the proof of the satisfiability of the Rank One Constraint System.

2 CSAT relation: R1CS

In Marlin [1], the CSAT relation is expressed as a rank 1 constraint system (R1CS). This is widely discussed in the literature, so we will keep our discussion brief and our goal in this section is to give an intuition for how R1CS representations lead to sparse matrices. The R1CS equation is expressed as

$$(A \cdot z) \circ (B \cdot z) = C \cdot z \tag{2.1}$$

where A, B, C are $n \times n$ matrices with elements in $\mathbb{F}_q$. In the above $A \cdot z$ refers to the usual dot product of the matrix A with the vector z, while $\circ$ refers to the Hadamard product. Let us take a simple example of the R1CS system (discussed in [5]). We modify the example slightly to also illustrate the large fan in support of the R1CS system.

Suppose we wish to prove that we know $(x_1, x_2) \in \mathbb{F}_p$ that satisfy

$$x_1^2 x_2 + x_1 + 1 = 22. \tag{2.2}$$

One can check that $x_1 = 3, x_2 = 2$ satisfies the above equation. We wish to represent (2.2) as an arithmetic circuit. Firstly, note that the total degree of the polynomial is 3 while the individual degrees are at most 2.

Consider the vector $z = [c, d, x_1, x_2, u, v]$ where $c = 1, d = 22$ are public inputs and u, v, w are intermediate values that depend on the solution x_1, x_2. We employ the arithmetization as shown in the first column of table (2.2), where the "." refers to ordinary multiplication. Each of the equations in the table (2.2) is a constraint that is executed sequentially. Furthermore, each constraint can be thought of as a "gate" and if one can find selection vectors a_i, b_i, c_i to activate the gates

$$(a_i \cdot z) \circ (b_i \cdot z) = (c_i \cdot z) \tag{2.3}$$

$\forall i = 1, 2, 3$, such that the three constraints can be realized, we would obtain a R1CS arithmetization of the problem. In terms of the addition and multiplication gates the arithmetization is as shown in fig.1. One can check that each line of selection vectors in table 1 produces the corresponding constraint when plugged into the R1CS vector equation

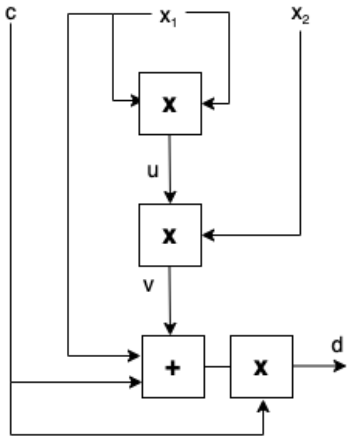

Figure 1: Circuit representation of the arithmetization in column 1 of table 1

(2.3). The last constraint $c.(c + x_1 + v) = d$ is realized using a larger fan in gate which is something R1CS is very efficient at. The fig.1 also makes it clear that any arithmetic circuit with gates can be transformed into an R1CS instance. We can represent all the constraints in table 1 together as a single Matrix equation

$$\left(\sum_{j=1}^{6} A_{ij} \cdot z_j\right) \circ \left(\sum_{j=1}^{6} B_{ij} \cdot z_j\right) = \left(\sum_{j=1}^{6} C_{ij} \cdot z_j\right) \quad \forall i = 1, 2, 3 \tag{2.4}$$

where

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \; B = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \; C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.5}$$

It is easy to check that the solution vector

$$z = [c, d, x_1, x_2, u, v] = [\underbrace{1 \quad 22}_{x} \quad \underbrace{3 \quad 2 \quad 9 \quad 18}_{w}]. \tag{2.6}$$

satisfies the R1CS equation.

The Matrices A, B, C encode the circuit information, in particular the gates and the wiring. The public inputs are the constants $x = [c, d]$, and the witness vector provided by the prover is $w = [x_1, x_2, u, v]$. The prover's claim is that the concatenation $z = [x||w]$ satisfies the R1CS equation, and the verifier is the entity that checks this claim. The number of rows/columns in the matrices (2.5) are proportional to the number of gates while the number of non zero entries in any row is less than or equal to the maximum fan-in of the circuit. This means for example that, for fan in two circuits, the R1CS matrices are mostly sparse.

3 Proof statement

Let us first rewrite the R1CS in the following notation

$$(z_A)_i = \sum_{j=1}^{n} A_{ij} z_j \ , \ (z_B)_i = \sum_{j=1}^{n} B_{ij} z_j \ , \ (z_C)_i = \sum_{j=1}^{n} C_{ij} z_j \tag{3.1}$$

where z is an m dimensional vector and A, B, C are $m \times n$ dimensional matrices. Thus the R1CS equation is represented as (we suppress all matrix and vector indices unless otherwise indicated)

$$z_A \circ z_B = z_C \tag{3.2}$$

where $\circ$ represents the element wise multiplication. Proving an instance of the R1CS equation (3.2) is equivalent to proving the statements [6]

1. **Lincheck problem:** To prove the computation

$$z_{\mathcal{M}} := \mathcal{M}.z \tag{3.3}$$

$\forall \mathcal{M} \in \{A, B, C\} \in \mathbb{F}^{m \times n}$ where $z = [x||w]$, with $x \in \mathbb{F}^k$ and $w \in \mathbb{F}^{n-k}$ as defined before.

2. **Rowcheck problem:**

$$z_A \circ z_B = z_C \tag{3.4}$$

where $z_{\mathcal{M}} \in \mathbb{F}^m$.

In the following, we assume that $\mathbb{F}$ possesses a multiplicative subgroup H such that $|H| = n$. We also assume that the Matrices are zero-padded to make them square in $\mathbb{F}^{n \times n}$. Thus we can index any vector or collection of points up-to $|n|$ using the roots of unity set $\{h^0, h^1, \ldots h^{n-1}\} \in H$ as shown in fig.2. We denote $\hat{z}(X)$ to denote the unique $deg < n-1$

Figure 2: Indexing vectors with roots of unity. $z^{(i)}$ is the i'th element in the vector z.

Low Degree Extension (LDE) of the vector z. (see §A for definitions of an LDE). What this means is that

$$\begin{aligned} \hat{z}(X) &= \sum_{i=0}^{n-1} a_i X^i \\ \hat{z}(h^r) &= z^{(r)} \ \forall \ 0 \leq r \leq n-1 \end{aligned} \tag{3.5}$$

where $z^{(r)}$ is the r'th element in z. Normally given a bunch of points the LDE is easily constructed using methods such as Lagrange interpolation (see §A) or using Number Theoretic Transforms (NTT).

Thus verifying the R1CS instance requires that the verifier check that the following polynomial identity holds for the rowcheck problem: (3.4)

$$f(X) = \hat{z}_A(X).\hat{z}_B(X) - \hat{z}_C(X) = 0 \ \forall X \in H \tag{3.6}$$

First for proving/verifying (3.6), we recollect that the vanishing polynomial in a domain H is defined as the monic

$$V_H(X) = X^H - 1 = \prod_{k=0}^{n-1}(X - x_k) \ \forall x_k \in H \tag{3.7}$$

that vanishes only on all points within H, and is non-vanishing outside. Thus any equation $f(X) = 0, X \in H$ can be written as

$$f(X) = h_0(X)V_H(X) \tag{3.8}$$

If $f(X)$ is of degree D, then it follows that $h_0(X)$ can be at most of degree $D - |H|$. Thus in order to verify the equation $f(X) = 0$. The prover is required to compute and commit the polynomials $f(X), h_0(X)$. (The polynomial commitment scheme used in Marlin is KZG [7]) The verifier can query evaluations of the commitments at some random value $\beta \in \mathbb{F}$, i.e $f(\beta), h_0(\beta)$ and evaluate $V_H(\beta)$. The queries occur in constant time ,while the evaluation complexity of $V_H(X)$ is $\mathcal{O}(\log |H|)$ field operations. Thus in the row check problem, the task of the prover is to compute and commit $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X), h_0(X)$. The verifier checks the equation

$$\hat{z}_A(X).\hat{z}_B(X) - \hat{z}_C(X) = h_0(X)V_H(X) \tag{3.9}$$

Since the LHS is at most of degree $2n$, $h_0(X)$ is at most of degree n.

For the Lincheck problem (3.3) we denote $\hat{z}_{\mathcal{M}}(X)$ as the LDE of the vectors $z_{\mathcal{M}}$ and $f_{\mathcal{M}}(X, Y)$ a bivariate polynomial representation of $\mathcal{M}_{n\times n}$ with degree n each in (X, Y) respectively .

$$\hat{z}_{\mathcal{M}}(X) = \sum_{Y \in H} f_{\mathcal{M}}(X, Y)\hat{z}(Y) \ \forall X \in H \ , \forall \mathcal{M} \in \{A, B, C\} \tag{3.10}$$

In Marlin the sumcheck problem is batched, i.e linear combinations of (3.10) are made

$$\sum_{\mathcal{M}} \eta_{\mathcal{M}} \hat{z}_{\mathcal{M}}(X) = \sum_{\mathcal{M}} \eta_{\mathcal{M}} \sum_{Y \in H} f_{\mathcal{M}}(X, Y)\hat{z}(Y) \tag{3.11}$$

where $\eta_{\mathcal{M}} \in \mathbb{F}$ are challenges sent by verifier. We dont need to consider this optimization at this point for the purposes of this article, but we will mention its effect in the end.

The lincheck problem takes bulk of the proving/verifying burdern in Marlin. But it also is the source of innovations and clever tricks in the protocol. The key steps are highlighted below

- In the first step, we convert the Lincheck to a sumcheck problem §4. Let us rewrite (3.10) as the sumcheck by (naively) taking the sum

$$\sum_{Y\in H} F(Y) = \sum_{Y\in H}\left(\frac{\hat{z}_{\mathcal{M}}(X)}{|H|} - f_{\mathcal{M}}(X,Y)\hat{z}(Y)\right) = 0 \tag{3.12}$$

  Thus the goal of the lincheck problem can be reduced to prove that $\sum_{Y\in H} F(Y) = 0$, where $F(Y)$ is the term in the brackets.

- In the second step we define a univariant sumcheck protocol that will be used to establish any $\sum_{Y\in H} F(Y) = \sigma$ for $\sigma \in \mathbb{F}$ §5

- In the third step, we define a succinct representation in §6, that leads to a representation of $f_{\mathcal{M}}(X,Y)$ using polynomials whose degrees are bounded by the number of non-zero entries $|K|$ in the R1CS Matrix. This allows efficient verification of $\mathcal{O}(|K|)$, and is referred to as holography.

- The representation $f_{\mathcal{M}}(X,Y)$ is a rational function and in the final step we define a univariate sumcheck specialized to rational functions §7

4 Lincheck to sumcheck

Formally let us define the lincheck problem (3.3) as

$$\hat{z}_{\mathcal{M}}(X) \stackrel{?}{=} \sum_{Y\in H} f_{\mathcal{M}}(X,Y)\hat{z}(Y) \tag{4.1}$$

where $X \in H$ and $f_{\mathcal{M}}(X,Y)$ is a bivariate polynomial representation of a $m \times n$ matrix. To begin the protocol, he prover and verifier agree on a polynomial $r(Z,X)$ with $deg < n$ in X and Z, such that $r(Z,X)|_{X\in H}$ form a linearly independent set. [1] The verifier samples a random element $\alpha \in \mathbb{F}$ and sends it to the prover. The prover computes the polynomial

$$F(X) = r(\alpha,X)\hat{z}_{\mathcal{M}}(X) - r(\alpha,X)\sum_{Y\in H} f_{\mathcal{M}}(X,Y)\hat{z}(Y) \tag{4.2}$$

Now, the naive summation we did earlier is more precisely done as follows. Sum both the LHS and RHS of the above equation to get

$$\sum_{X\in H} F(X) = \sum_{X\in H} r(\alpha,X)\hat{z}_{\mathcal{M}}(X) - \sum_{X\in H}\sum_{Y\in H} r(\alpha,X) f_{\mathcal{M}}(X,Y)\hat{z}(Y) \tag{4.3}$$

$$= \sum_{X\in H} r(\alpha,X)\hat{z}_{\mathcal{M}}(X) - \sum_{X\in H}\sum_{Y\in H} r(\alpha,Y) f_{\mathcal{M}}(Y,X)\hat{z}(X) \tag{4.4}$$

[1] A simple basis are the derivatives of the vanishing polynomial which will be defined in (6.6), which is also used in Marlin. The current discussion is valid for any $r(Z,X)|_{X\in H}$ that form a linearly independent set.

where the second step is allowed since these are finite dimensional sums/Matrices and swapping variables in $f_{\mathcal{M}}(X, Y)$ is equivalent to the polynomial representation of the transposed R1CS matrices. Thus, we can rewrite

$$F(X) := r(\alpha, X)\hat{z}_{\mathcal{M}}(X) - r_{\mathcal{M}}(\alpha, X)\hat{z}(X) \tag{4.5}$$

where

$$r_{\mathcal{M}}(Z, X) = \sum_{Y \in H} r(Z, Y) f_{\mathcal{M}}(Y, X) \tag{4.6}$$

and taking the sum over H on both sides of equation (4.5), we reproduce (4.4). Thus the task of proving (4.1) is equivalent to proving the equivalent sumcheck problem

$$\sum_{X \in H} F(X) \stackrel{?}{=} 0 \tag{4.7}$$

where $F(X)$ is given by (4.5). Thus we have reduced the lincheck problem (4.1) to the sumcheck problem. In the next section, we describe the univariate sumcheck protocol to prove the above equation.

5 Univariate sum-check protocol

The classical sumcheck protocol [8] is an Interactive protocol for the claim

$$\sum_{x_1} \sum_{x_2} \dots \sum_{x_m} F(x_1, x_2, \dots x_m) = 0 \tag{5.1}$$

where $x_i \in \mathbb{F}$. It proceeds over m rounds, with a step wise reduction of a sumcheck problem with r variables to a smaller sumcheck problem with $r - 1$ variables. In the univariate sumcheck [9] there is only one variable, and the statement to be proven is

$$\sum_{Y \in H} F(Y) \stackrel{?}{=} 0 \tag{5.2}$$

We begin by considering an auxiliary problem

$$\sum_{Y \in H} F(Y) \stackrel{?}{=} \sigma \tag{5.3}$$

for $\sigma \in \mathbb{F}$. In our discussion H is a multiplicative subgroup in $\mathbb{F}$ and is spanned by the set $\{1, h, h^2, \dots, h^{n-1}\}$, expanding the sum we get

$$\sum_{Y \in H} F(Y) = F(1) + F(h) + \dots F(h^{n-1}) \tag{5.4}$$

Expanding each term in the sum using $F(Y) = \sum_{i=0}^{d} a_i Y^i$ we get

$$\begin{aligned} \sum_{Y \in H} F(Y) = {} & (a_0 + a_1 + a_2 + \dots + a_d) \\ & + (a_0 + a_1 h + a_2 h^2 + \dots + a_d h^d) \\ & + (a_0 + a_1 h^2 + a_2 h^4 + \dots + a_d h^{2d}) \\ & \quad \vdots \\ & + (a_0 + a_1 h^{n-1} + a_2 h^{2(n-1)} + \dots + a_d h^{(n-1)d}) \end{aligned} \tag{5.5}$$

Grouping the terms in terms of the coefficients a_i

$$\sum_{Y\in H} F(Y) = a_0|H| + a_1\sum_{i=0}^{n-1} h^i + a_2\sum_{i=0}^{n-1} h^{2i} + \ldots + a_d\sum_{i=0}^{n-1} h^{id} \tag{5.6}$$

Since $h^n = 1$ (modulo prime field usually), all the sums vanish due to the property

$$\sum_{i=0}^{n-1} h^{i\cdot d} = \frac{1-h^{n\cdot d}}{1-h^d} = \begin{cases} |H| \text{ if } d \equiv 0 \mod n \\ 0 \text{ if } d = \{1, \ldots n-1\} \end{cases} \tag{5.7}$$

except for the constant term in the polynomial $F(Y)$ and thus we have the relation

$$\sum_{X\in H} F(X) = \begin{cases} \sum_{k=0}^{d-1} \mod |H| \ a_k \cdot |H| & ; d \geq |H| \\ |H| \cdot a_0 = |H| \cdot F(0) & ;\ d < |H| \end{cases} \tag{5.8}$$

which means that to satisfy (5.3) we need to have the relations

$$\sum_{k=0 \mod n}^{d} a_k = \frac{\sigma}{|H|} \ ;\ d \geq |H| \tag{5.9}$$

$$a_0 = \frac{\sigma}{|H|} \ ;\ d < |H| \tag{5.10}$$

or the prover needs to "know" this coefficient, which is the essential verification issue in the sumcheck problem.

In the general case (arbitrary d), the summand in the LHS of (5.8) does not vanish (for eg in (3.12)). Thus if we divide it by the vanishing polynomial it must have a finite remainder.

$$F(Y) = h(Y) \cdot V_H(Y) + R(Y) \tag{5.11}$$

While we could use this equation for the sumcheck, by requiring the prover to send the polynomials $h(Y)$ with $deg(h) = d-n$ and $R(Y)$ with $deg(R) < n-1$, it allows a malicious prover to manipulate the coefficient $F(0)$, since it hides in R (see Appendix §C). Taking the sum on both sides over H, and observing that the $V_H(X)$ vanishes in H, we find that the constant term $F(0)$ comes from the Remainder polynomial

$$\sum_{Y\in H} R(Y) = \sigma \tag{5.12}$$

which means that $R(Y)$ must have the structure

$$R(Y) = Yg(Y) + \frac{\sigma}{|H|} \tag{5.13}$$

where $deg(g) < |H| - 1$. Substituting the above equation in (5.12) and using (5.7) we see that the relation (5.12) is satisfied. Thus the most general expression for an $F(Y)$ that can satisfy (5.3) is given by the expression

$$F(Y) = h(Y).V_H(Y) + Yg(Y) + \frac{\sigma}{|H|} \tag{5.14}$$

Taking the sum $\sum_{Y \in H}$ on both sides, we see that it indeed reproduces the equation (5.3). Thus in order to prove that (5.3) sums to a certain value, it is sufficient that the prover knows polynomials $h(Y), g(Y)$ such that (5.14) is satisfied at any random $\alpha \in \mathbb{F}$. This is the main technical result of the univariate sumcheck protocol.

So far, we have been very quiet about the functional form of $F(Y)$ that involves a bivariate representation of the R1CS matrix as given in (3.12). In the following section, we address how to succinctly represent these matrices.

6 Succinct representation of R1CS matrices

We recollect that the main goal of the lincheck problem is to prove (3.12). In §4, we showed the equivalence to the sumcheck problem

$$F(X) := r(\alpha, X)\hat{z}_{\mathcal{M}}(X) - r_{\mathcal{M}}(\alpha, X)\hat{z}(X) \tag{6.1}$$

where $\alpha \in \mathbb{F}$ is a random challenge from the verifier. Since the prover has already committed to $\hat{z}(X), \hat{z}_{\mathcal{M}}(X)$ following the rowcheck problem. The verifier can query the evaluations $\hat{z}_{\mathcal{M}}(\alpha)$. In the last step of the sumcheck problem, the verifier verifies that (6.1) holds as a polynomial identity, i.e he/she evaluates it at a random point $\beta \in \mathbb{F}$.

$$F(\beta) \stackrel{?}{=} r(\alpha, \beta)\hat{z}_{\mathcal{M}}(\beta) - r_{\mathcal{M}}(\alpha, \beta)\hat{z}(\beta) \tag{6.2}$$

Following earlier arguments $\hat{z}(\beta)$ can also be obtained in a single query and takes constant time. The function $r(\alpha, \beta)$ is written in terms of the vanishing polynomial (3.7) and is also easily evaluated in logarithmic time. However, the elephant in the room is the evaluation of $r_{\mathcal{M}}(\alpha, \beta)$ (4.6), which includes the evaluation of polynomial representation $f_{\mathcal{M}}(\alpha, \beta)$ of the R1CS matrices. Naively, the evaluation of $f_{\mathcal{M}}(\alpha, \beta)$ will take $\Omega(n+m)$ field operations (In general, the R1CS matrices are of dimension $m \times n$, though we have assumed square matrices in this discussion) in the best case scenario.

An important feature of the R1CS matrices are that for fan in 2 circuits, the matrices are mostly sparse. Thus it makes sense to attempt an encoding of the R1CS matrices using a sparse representation that depends on the number of non-zero entries in the matrices. In Marlin, the matrix encoding is done by a trusted party known as the "indexer" that commits to the sparse encoding of the matrices $\mathcal{M}$, and the verifier has oracle access to the encoding. i.e, the verifier can ask the prover to reveal an evaluation from the commitment at challenge points, say $f_{\mathcal{M}}(\alpha, \beta)$ and verify the evaluation by querying the indexer.

The goal is to achieve pre-processing time of the indexer, prover time that is of the order of linear time in the number of non-zero entries in the matrices. In the following paragraphs we discuss the method [6] used in [1] to succinctly represent the sparse R1CS matrices in terms of the number of non-zero entries. We motivate the above discussion with a simple example as shown in fig. 3. The example matrix has a dimension of $64 = 8 \times 8$ and the domain of the problem required to index the rows and columns of the matrix is $|H| = 8$. Thus, we have indexed the rows and columns of the matrix with the roots of unity in the domain H labelled by the generator set $\{h^0, h^1, \ldots, h^7\}$. Note that the total number

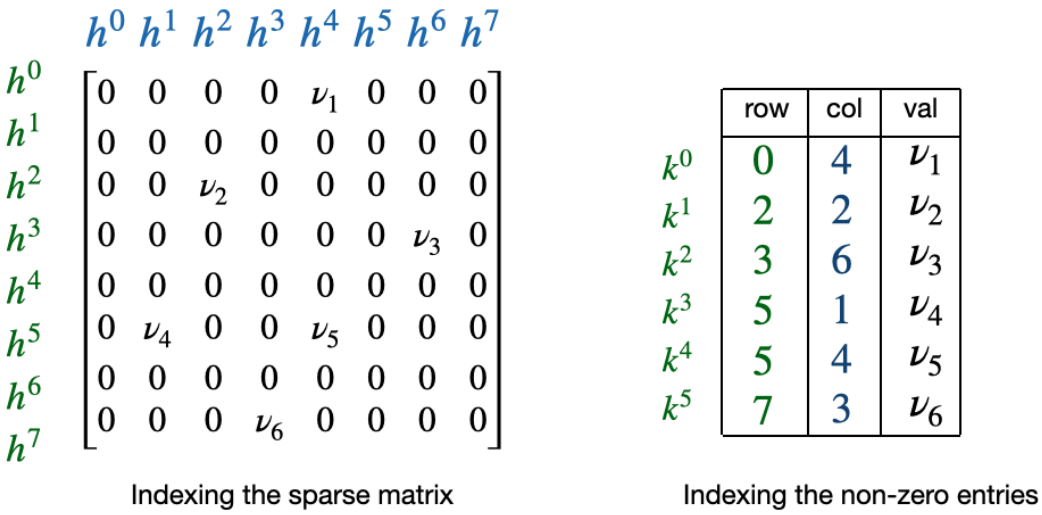

Figure 3: Succinct representation of R1CS matrices. On the left we indexed the R1CS matrix in the domain H, while on the right, we index only the non zero entries in the domain K where $|K| < |H|$.

of non-zero entries is just $|K| = 6 < |H|$. Thus the actual information to be represented is much less than the total number of entries in the matrix. Hence it makes sense to think about a representation of the matrix purely in terms of polynomials of degree $\leq |K|$. Thus in fig.3, the relevant data in the matrices are the triple : (row index, column index, actual value).

We assume that there exists a multiplicative subgroup $K \subset H$ in $\mathbb{F}$ and index the entries of the RHS of the table using the roots of unity in K. Let us represent the triple that represents the non-zero entries in the R1CS matrices as the evaluations $(row_{\mathcal{M}}(k), col_{\mathcal{M}}(k), val_{\mathcal{M}}(k)$: $\forall k \in K$ and $\mathcal{M} \in \{A, B, C\}$. Let $(r_{\mathcal{M}}(X), c_{\mathcal{M}}(X), v_{\mathcal{M}}(X))$ be the LDE of the vectors (see Appendix §A for definitions of LDE and Lagrange polynomials) in K defined as

$$\begin{aligned} r_{\mathcal{M}}(X) &= \sum_{Z \in K} row_{\mathcal{M}}(k) L_{K,Z}(X) \\ c_{\mathcal{M}}(X) &= \sum_{Z \in K} col_{\mathcal{M}}(k) L_{K,Z}(X) \\ v_{\mathcal{M}}(X) &= \sum_{Z \in K} val_{\mathcal{M}}(k) L_{K,Z}(X) \end{aligned} \tag{6.3}$$

where L_K are Lagrange polynomials (A.2), defined over the domain K. The LDE's $r(X), c(X), v(X)$ map $K \to \mathbb{F}^{<|K|}$ and each of the polynomials is of degree $deg < |K|$. In other words this is equivalent to an inverse FFT of $|K|$ point evaluations, that would give a unique polynomial of $deg < |K|$.

In the first step we attempt to express the R1CS matrices as a bivariate polynomial using the Lagrange representation. The key idea is to compose the Lagrange polynomials in terms of the degree $< |K|$ polynomials (6.3). We begin with the representation of a

bivariate polynomial using the Lagrange polynomials in H composed in terms of (6.3).

$$f_{\mathcal{M}}(X,Y) = \sum_{Z \in K} L_{H,r_{\mathcal{M}}(Z)}(X) L_{H,c_{\mathcal{M}}(Z)}(Y).v_{\mathcal{M}}(Z) \tag{6.4}$$

Note that this is not the usual way to compose a decomposition, since the Lagrange polynomial itself is of degree $|H|$, and it is composed in terms of polynomials $r_{\mathcal{M}}(Z), c_{\mathcal{M}}(Z)$ with each term in the sum of degree $deg \sim |H||K|$ in the variables X, Y independently. Our goal is to get the verifier to have linear evaluation time in $|K|$, which means every term in the summand of (6.4) should have a degree $\leq |K|$

The main trick that is used in Marlin to generate the succinct representation is to use the sparse matrix representation discussed in [6]. A key step involves, rewriting the Lagrange polynomial in terms of derivatives of the vanishing polynomial (see (3.7)), which we discuss below. The formal derivative of the vanishing polynomial [10] (independent of H) is defined as

$$V'_H(X,Y) = \begin{cases} \frac{V_H(X) - V_H(Y)}{X - Y} & X \neq Y \\ |H| X^{|H|-1} & X = Y \end{cases} \tag{6.5}$$

Since $V_H(X)\big|_H = 0$, the derivative $V'_H(X,Y)\big|_H$ vanishes on all square $H \times H$ except the diagonal where the non vanishing entries are of degree $|H| - 1$. Substituting $Y = x_i \in H$ in the first equation of (6.5), we see that the derivative of the vanishing polynomial is proportional to the Lagrange polynomial upto an overall constant

$$\begin{aligned} V'_H(X, x_i) = \frac{V_H(X)}{X - x_i} &= \prod_{k=0, k \neq i}^{n-1} (X - x_k) \\ &= L_{H,x_i}(X) \prod_{k=0, k \neq i}^{n-1} (x_i - x_k) \\ &= L_{H,x_i}(X).V'_H(x_i, x_i) \end{aligned} \tag{6.6}$$

where in the last line of (6.6) we have used for $x_i \in H$ (where the evaluation is always done after the formal application of the derivatives)

$$\begin{aligned} \frac{\partial}{\partial X}(V_H(X))\Big|_{X=x_i} &= \frac{\partial}{\partial X} \prod_{k=0}^{n-1} (X - x_k)\Big|_{X=x_i} \\ |H| X^{|H|-1}\Big|_{X=x_i} &= \sum_{r=0}^{n-1} \prod_{k=0, k \neq r}^{n-1} (X - x_k)\Big|_{X=x_i} \\ V'_H(x_i, x_i) &= \prod_{k=0, k \neq i}^{n-1} (x_i - x_k) \end{aligned} \tag{6.7}$$

Thus we can write the Lagrange polynomial in terms of the vanishing polynomial as

$$L_{H,x_i}(X) = \frac{V'_H(X, x_i)}{V'_H(x_i, x_i)} = \frac{V_H(X)}{X - x_i} \frac{1}{V'_H(x_i, x_i)} \tag{6.8}$$

Substituting the relation between the Lagrange polynomial and the vanishing polynomial (6.8) into the naive bivariate representation (6.4)

$$f_{\mathcal{M}}(X,Y) = \sum_{Z\in K} \frac{V_H(X)}{X - r_{\mathcal{M}}(Z)} \cdot \frac{V_H(Y)}{Y - c_{\mathcal{M}}(Z)} \tilde{v}_{\mathcal{M}}(Z) \tag{6.9}$$

where we have defined a normalized value element

$$\tilde{v}_{\mathcal{M}}(Z) = \frac{v_{\mathcal{M}}(Z)}{V'_H(r_{\mathcal{M}}(Z), r_{\mathcal{M}}(Z)) V'_H(c_{\mathcal{M}}(Z), c_{\mathcal{M}}(Z))} \tag{6.10}$$

What have we accomplished in (6.9) ? First note that the RHS is expressed entirely in terms of polynomials $V_H(X), (r_{\mathcal{M}}(Z), c_{\mathcal{M}}(Z), v_{\mathcal{M}}(Z))$. From the verifiers point of view, the vanishing polynomials are sparse and are evaluated in $\mathcal{O}(\log |H|)$ field operations. The key innovation in marlin is that the representations $(r_{\mathcal{M}}(Z), c_{\mathcal{M}}(Z), v_{\mathcal{M}}(Z))$ are computed and committed by the Indexer algorithm in the pre-processing phase. The verifier is given oracle access to these commitments and thus can do $\mathcal{O}(|K|)$ queries, to get the data needed to evaluate the RHS of (6.9) since the polynomials are at most $deg < |K|$. The verifier chooses random field elements $\alpha, \beta \in \mathbb{F}$ and evaluates $f_{\mathcal{M}}(\alpha, \beta)$.

From the prover's point of view, this change adds another layer of computation wherein the prover has to first commit to the summand in (6.9) namely

$$T_{\mathcal{M}}(Z) = \frac{V_H(\alpha) V_H(\beta) \tilde{v}_{\mathcal{M}}(Z)}{(\alpha - r_{\mathcal{M}}(Z))(\beta - c_{\mathcal{M}}(Z))} \tag{6.11}$$

Then the prover and verifier engage in a sub-protocol which basically checks that

$$f_{\mathcal{M}}(\alpha, \beta) \stackrel{?}{=} \sum_{Z\in K} T_{\mathcal{M}}(Z) \tag{6.12}$$

which looks much like the uni-variate sumcheck problem that we discussed earlier in §5, except that $S(Z) \sim p(Z)/q(Z)$ is a rational function and one needs to generalize the univariate sumcheck for rational functions, which we discuss in the following section §7.

7 Rational Sum-check protocol

The rational sumcheck is a generalization of the univariate sumcheck (5.3) where the statement to be proven is

$$\sum_{Z\in K} \frac{p(Z)}{q(Z)} \stackrel{?}{=} \sigma \tag{7.1}$$

for $\sigma \in \mathbb{F}$ and $q(Z) \neq 0 \ \forall \ Z \in K$. The prover first constructs a polynomial $f(Z)$ with $deg(f) < |K|$ that agrees with the LHS of (7.1). i.e

$$p(Z) - q(Z) \cdot f(Z) = h(Z) v_K(Z) \tag{7.2}$$

which one can verify by taking sum over the domain K on both sides of the equation. Then to prove the sum (7.1) it is sufficient to require

$$f(Z) = Z \cdot g(Z) + \frac{\sigma}{|K|} \tag{7.3}$$

which again one can check by taking the sum over K on both sides and using the property (5.7). The two equations (7.2) and (7.3) can be combined in a single step to obtain

$$p(Z) - q(Z) \cdot \left(Z \cdot g(Z) + \frac{\sigma}{|K|} \right) \stackrel{?}{=} h(Z) v_K(Z) \tag{7.4}$$

If $deg(p) = d_1, deg(q) = d_2$ then it follows from (7.2) that $deg(h) = d_3 \leq max(d_1, |K| - 1 + d_2)$. Since $deg(f) < |K|$, it follows from (7.3) that $deg(g) \leq |K| - 2$. Thus for the rational sumcheck, it is sufficient for the prover to send the polynomials $g(Z)$ and $h(Z)$ satisfying the degree bounds

$$deg(g) \leq |K| - 2 \ , \ deg(h) \leq d_3 \tag{7.5}$$

8 Adding zero knowledge to the sumcheck

In order to make the sumcheck protocol Zero Knowledge, at the beginning of the protocol, the prover samples a random polynomial called the Masking polynomial $S(X) \leftarrow \mathbb{F}^{<2|H|+b-1}$, where b is the query bound (is set to $b = 1$) for the verifier. The prover commits the polynomial $S(X)$ and also sends the $\sum_{X \in K} S(X) = \sigma_1$ to the verifier.

Normally, following the prover message, the verifier would respond with a random challenge c and thus one can combine in the original sumcheck equation (5.3)

$$\sum_{X \in H} (c \cdot S(X) + F(X)) \stackrel{?}{=} c \cdot \sigma_1 + \sigma \tag{8.1}$$

to restate it as a zero knowledge sumcheck. Why is this zero knowledge? Note that (5.14) is a unique decomposition of $F(X)$ and the verifier gets basically everything about $F(X)$ from the polynomials $g(X), h(X)$ sent by the prover. But whereas, in (8.1) this information is masked by the random polynomial S, i,e the verifier gets the decomposition g, h of the polynomial $\tilde{F}(X) = c.S(X) + F(X)$ and cannot reconstruct $F(X)$, thus this is zero knowledge.

In the following scenario: for a Malicious prover sends $\sum_{X \in K} S(X) = \sigma_1$. Let $\sum_{X \in H} F(X) = \sigma'$, and the malicious prover generates $\sum_{X \in H} S'(X) = \sigma_1'$, the check will still pass if $c = \frac{\sigma - \sigma'}{\sigma_1' - \sigma_1}$, however the probability of this happening is negligible, since c is generated after the prover commits to $S(X)$. Nevertheless, forcing the prover to committ to $S(X)$ before c is generated is important to prevent malleability.

The precise way in which ZK is added in Marlin is

- In the beginning of the protocol, the prover commits $\hat{z}(X), \hat{z}_{\mathcal{M}}(X), S(X)$ and also sends the value $\sum_{X \in H} S(X) = \sigma_1$.
- The verifier generates the challenge α based on the above commitments and sends it to the prover. Or in the non-interactive case, the Fiat-Shamir transform takes into account all the commitments to generate the challenge α.
- The masking is applied in the first step of converting the lincheck to the sumcheck in (4.2) i.e

$$F(X) = S(X) + r(\alpha, X) \left(\hat{z}_{\mathcal{M}}(X) - \sum_{Y \in H} f_{\mathcal{M}}(X, Y) \hat{z}(Y) \right) \tag{8.2}$$

Note that the prover cannot proceed with the sumcheck for the bracketted term in (8.2) without knowing α, (also in (4.2))which in turn necessitates committing to $S(X)$ in the first step. The challenge α is generated by a oracle (hash function) that takes into account the commitment of $S(X)$. In other words, the multiplication by the $r(\alpha, X)$ is equivalent to the multiplication of c in (8.1).

9 Putting it all together

In this section, we put together all the various steps discussed above in an overview form.

- In the preprocessing step, the indexer algorithm computes the polynomials (6.3) and commits to $r_{\mathcal{M}}(X), c_{\mathcal{M}}(X), v_{\mathcal{M}}(X)$, that represent the location and values of the non-zero entries in the R1CS matrix. The verifier has oracle access to the polynomials i.e the verifier can query the evaluations of any of these polynomials in constant time.

- In the beginning of the protocol, In Round 1

  - The prover computes and commits to the polynomial representations of $\hat{z}(X), \hat{z}_{\mathcal{M}}(X)$ that appear in the LHS and RHS in the Lincheck and Sumcheck problems as discussed in §4
  - The prover computes and commits to the masking polynomial $S(X)$ as discussed in §8 and sends $\sum_{X\in H} S(X) = \sigma_1$ to the verifier.
  - The prover computes the quotient polynomial $h_0(X)$ in the row check problem

$$\hat{z}_A(X).\hat{z}_B(X) - \hat{z}_C(X) \stackrel{?}{=} h_0(X)V_H(X) \tag{9.1}$$

    and sends $h_0(X)$ to the verifier. This part completes the necessary checks for the rowcheck problem (3.4).
  - The verifer responds with challenge values $\alpha, \eta_{\mathcal{M}} \in \mathbb{F}$.

- As discussed in §3, in Marlin the sumcheck problem is batched, i.e linear combinations of (3.10) are made

$$\sum_{\mathcal{M}} \eta_{\mathcal{M}} \hat{z}_{\mathcal{M}}(X) = \sum_{\mathcal{M}} \eta_{\mathcal{M}} \sum_{Y\in H} f_{\mathcal{M}}(X, Y)\hat{z}(Y) \tag{9.2}$$

- In Round 2, we move on to the lincheck problem (3.3). As we discussed in §4 and §8 we add the masking polynomial and include the batching and convert the lincheck to the sumcheck problem

$$\sum_{X\in H} \left(S(X) + r(\alpha, X) \sum_{\mathcal{M}} \eta_{\mathcal{M}} \hat{z}_{\mathcal{M}}(X) - \left(\sum_{\mathcal{M}} \eta_{\mathcal{M}} r_{\mathcal{M}}(\alpha, X) \right) \hat{z}(X) \right) \stackrel{?}{=} \sigma_1 \tag{9.3}$$

where we apply the result of the univariate sumcheck protocol (5.14) to the above equation, i.e to show

$$S(X) + r(\alpha, X)\sum_{\mathcal{M}} \eta_{\mathcal{M}} \hat{z}_{\mathcal{M}}(X) - \left(\sum_{\mathcal{M}} \eta_{\mathcal{M}} r_{\mathcal{M}}(\alpha, X)\right) \hat{z}(X) \stackrel{?}{=} h_1(X)V_H(X) + Xg_1(X) + \frac{\sigma_1}{|H|} \tag{9.4}$$

  - The prover computes and sends the polynomials $g_1(X)$ and $h_1(X)$
  - Verifier responds with challenge value β_1 which proceeds further into evaluating the above equation.

- In our earlier discussion on succinct representation §6, we moved directly to the rational sumcheck for the evaluation of $r_{\mathcal{M}}(\alpha, \beta)$. However, here (9.4) we have batched by adding a linear combination. Thus in Round 3 we have another sumcheck for the batching. i.e in (4.6)

$$\sum_{X \in H} \sum_{\mathcal{M}} \eta_{\mathcal{M}} r(\alpha, X) f_{\mathcal{M}}(X, \beta_1) \stackrel{?}{=} \sigma_2 \tag{9.5}$$

  Again, applying the univariate sumcheck result (5.14) for the above equation, the prover needs show

$$\eta_{\mathcal{M}} r(\alpha, X) f_{\mathcal{M}}(X, \beta_1) \stackrel{?}{=} h_2(X)V_H(X) + Xg_2(X) + \frac{\sigma_2}{|H|} \tag{9.6}$$

  - The prover computes and sends the polynomials $g_2(X)$ and $h_2(X)$ and the sumvalue σ_2
  - The verifier sends the challenge value β_2

- Finally in Round 4 we address the rational sumcheck for the evaluation of $f_{\mathcal{M}}(\beta_2, \beta_1)$. i.e the prover needs to establish the batched version of (6.12)

$$\sum_{X \in H} \sum_{\mathcal{M}} \eta_{\mathcal{M}} \frac{V_H(\beta_2)V_H(\beta_1)\tilde{v}_{\mathcal{M}}(X)}{(\beta_2 - r_{\mathcal{M}}(X))(\beta_1 - c_{\mathcal{M}}(X))} \stackrel{?}{=} \sigma_3 \tag{9.7}$$

  We apply the result of the rational sumcheck (7.4) and thus the verifier needs to show

$$p(X) - q(X) \cdot \left(X \cdot g_3(X) + \frac{\sigma_3}{|K|}\right) \stackrel{?}{=} h_3(X)v_K(X) \tag{9.8}$$

  - The prover computes the polynomials $p(X), q(X), h_3(X), g_3(X)$ and sends $h_3(X), g_3(X)$ to the verifier along with the sumvalue σ_3. The verifier can compute the numerator and denominator in $\mathcal{O}(|K|)$ time by querying the indexer for evaluations of $r_{\mathcal{M}}(X), c_{\mathcal{M}}(X), \tilde{v}_{\mathcal{M}}(X)$.

This concludes our discussion of the Marlin protocol. We hope to add more details in future versions of the article.

Acknowledgments

We would like to thank the Geometry Study club, especially Kobi Gurkan, Andrija Novakovic and Nicholas Mohnblatt, and Suyash Bagad for several enjoyable discussions. We would like to thank Omer Shlomovits, and Oren Margalit for reviewing the article.

A Low Degree Extension

A useful definition is the notion of a low degree extension [8] (LDE). Given a vector $a = (a_1, a_2, \ldots a_n) \in \mathbb{F}^n$, and a functional map $f : H \to \mathbb{F}$ where $|H| = n$ is the domain size of the vector a. The LDE is a unique polynomial $P_a(X) \in \mathbb{F}^{<|H|}$ that passes through all the points in a and agrees with $f \in H$. In other words, $P_a(X)$ is defined such that evaluations in the domain H give the vector elements

$$P_a(x_i) = a_{i+1} \ \forall i \in \{0, 1, \ldots n-1\} \tag{A.1}$$

where x_i is a generator of a root of unity in the domain H. The definition generalizes for a multi variate case $H_1 \times H_2 \times \ldots H_m \to \mathbb{F}$. i.e For a multivariate polynomial $P(X_1, X_2, \ldots X_m)$ such that $deg(P(X_1, X_2, \ldots X_m))$ is $\leq |H_i| \ \forall \ X_i \ \in H_i$ respectively.

Such a polynomial has a unique representation in terms of the Lagrange basis

$$L_{H,x_i}(X) = \frac{\prod_{k=0,k\neq i}^{n-1}(X - x_k)}{\prod_{k=0,k\neq i}^{n-1}(x_i - x_k)} \tag{A.2}$$

that satisfy the relations

$$L_{H,x_i}(X = x_j) = \delta_{ji} = \begin{cases} 1 \ , \ j = i \ \forall \ i, j = 0, 1, \ldots n-1 \\ 0 \ , \ \text{otherwise} \end{cases} \tag{A.3}$$

Note that each basis polynomial (A.2) is at most of degree $n-1$. Thus, a LDE of any given set of n points is defined in terms of the Lagrange polynomial (A.2)

$$P_a(X) = \sum_{j=0}^{n-1} a_{j+1} L_{H,x_{j+1}}(X) \tag{A.4}$$

which is a polynomial of at most $deg \leq n-1$. Since any two polynomials of $deg \leq n-1$ can agree at at most $n-1$ points, whereas (A.1) specifies it at n points. Thus two different $n-1$ degree polynomials cannot specify (A.1) which establishes that (A.4) is unique.

B The multivariate Sum check protocol

One of the most important tools used in Marlin is the sumcheck protocol [8]. The Sumcheck is a statement about sums of evaluations of a polynomial in a finite set. Consider a m variate polynomial $g(x_1, x_2, \ldots x_m)$, with degree $deg_{x_i}(g)$ in each variable x_i. The sumcheck protocol is a method of proving the statement that the evaluation of the sum

$$C = \sum_{x_1} \sum_{x_2} \ldots \sum_{x_m} g(x_1, x_2, \ldots x_m) \ \forall x_i \in \{0, 1\} \text{ where } i \in \{1, \ldots, m\} \tag{B.1}$$

holds true. The evaluation domain of g is over a boolean hypercube of dimension 2^m. In general, the sum check protocol can compute the sum for any $x_i \in \mathbb{F}$, where $\mathbb{F}$ is a finite field. In many protocols (like in Marlin), the verifier has an oracle access to the polynomial g and can evaluate efficiently at any random value $r \in \mathbb{F}$. In general to check the sum, the verifier has to compute 2^m evaluations. In the current discussion we assume that there is an interaction between the prover and the verifier, thus the verifier can ask the prover for evaluations at random points in the field and verify the statements till he/she is convinced.

Given the above, the protocol proceeds as follows. Before the beginning of the protocol the prover and verifier agree on the LHS C.

- In the first round
  - The prover evaluates the multivariate polynomial in the last $m-1$ points and sends a univariate polynomial $g_1(X)$ with the claim

$$g_1(X) := \sum_{x_i \neq x_1, x_i \in \{0,1\}} g(X, x_2, \ldots x_m)$$

    to the verifier. In all prover communication, we assume that the prover sends the polynomial in the coefficient form.
  - The verifier first checks that
    - $deg_{x_1}(g) \stackrel{?}{=} deg(g_1)$, if this check fails, the verifier exits and rejects the proof. This is obvious by definition that g is of degree $deg_{x_i}(g)$ for each x_i,
    - $C \stackrel{?}{=} \sum_{X \in \{0,1\}} g_1(X)$. This check also follows from the definition (B.1) that upon evaluation of all variables on the RHS the value must be equal to C.
  - If the check passes, the verifier picks a random $r_1 \in \mathbb{F}$ and evaluates $g_1(r_1)$, then the verifier sends r_1 to the prover.
- For rounds $k = 2, ..m-1$, in the k'th round, the verifier checks the relation between g_k and g_{k-1}.
  - The Prover receives a random field element r_{k-1} and evaluates

$$g_k(X) := \sum_{x_i \neq x_k, x_i \in \{0,1\}} g(r_1, r_2, \ldots, r_{k-1}, X, x_{k+1} \ldots, x_m)$$

    and sends $g_k(X)$ to the verifier.
  - The verifier first checks that
    - $deg_{x_k}(g) \stackrel{?}{=} deg(g_k)$
    - $g_{k-1}(r_{k-1}) \stackrel{?}{=} \sum_{X \in \{0,1\}} g_k(X)$.
  - If the check passes, the verifier picks a random $r_k \in \mathbb{F}$ and evaluates $g_k(r_k)$, then the verifier sends r_k to the prover.
- In the final round m,

– The Prover receives a random field element r_{m-1} and evaluates

$$g_m(X) := g(r_1, r_2 \ldots, r_{m-1}, X)$$

and sends $g_m(X)$ to the verifier. The prover's work is done.

– The verifier checks in sequence

* $deg_{x_m}(g) \stackrel{?}{=} deg(g_m)$
* $g_{m-1}(r_{m-1}) \stackrel{?}{=} \sum_{X\in\{0,1\}} g_m(X)$.
* The verifier queries the evaluations of $g(x_1, x_2, \ldots x_m)$ on the set $(r_1, r_2, \ldots r_m)$ and executes the final check

$$g_m(r_m) \stackrel{?}{=} g_(r_1, r_2, \ldots r_m)$$

.

- If the prover sends all the prescribed polynomials $g_i(X)$, and all the checks in red pass, then the verifier accepts the claim with a probability of 1 ensuring that the protocol is complete. The protocol is summarized in table 2.

Round	Prover $\mathcal{P}$	Communication	Verifier $\mathcal{V}$
1	$g_1(X) := \sum_{x_i\neq x_1} g(X, x_2, \ldots x_m)$	$g_1(X) \longrightarrow$ $\longleftarrow r_1 \in \mathbb{F}$	$deg_{x_1}(g) \stackrel{?}{=} deg(g_1(X))$ $C \stackrel{?}{=} \sum_{X\in\{0,1\}} g_1(X)$ compute $g_1(r_1)$
2	$g_2(X) := \sum_{x_i\neq x_2} g(r_1, X, \ldots x_m)$	$g_2(X) \longrightarrow$ $\longleftarrow r_2 \in \mathbb{F}$	$deg_{x_2}(g) \stackrel{?}{=} deg(g_2(X))$ $g_1(r_1) \stackrel{?}{=} \sum_{X\in\{0,1\}} g_2(X)$ compute $g_2(r_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m-1$	$g_{m-1}(X) := \sum_{x_i\neq x_{m-1}} g(r_1, r_2, \ldots X, x_m)$	$g_{m-1}(X) \longrightarrow$ $\longleftarrow r_{m-1} \in \mathbb{F}$	$deg_{x_{m-1}}(g) \stackrel{?}{=} deg(g_{m-1}(X))$ $g_{m-2}(r_{m-2}) \stackrel{?}{=} \sum_{X\in\{0,1\}} g_{m-1}(X)$ compute $g_{m-1}(r_{m-1})$
m	$g_m(X) := \sum_{x_i\neq x_m} g(r_1, r_2, \ldots, r_{m-1}, X)$	$g_m(X) \longrightarrow$	$deg_{x_m}(g) \stackrel{?}{=} deg(g_m(X))$ $g_{m-1}(r_{m-1}) \stackrel{?}{=} \sum_{X\in\{0,1\}} g_m(X)$ $g_m(r_m) \stackrel{?}{=} g(r_1, r_2, \ldots r_m)$

Table 2: Sumcheck protocol summary

C Soundness in univariate sumcheck

In this section we motivate a soundness issue in how one implements the univariate sumcheck. It is subtle and can easily escape notice. Let us begin with

$$F(Y) = h(Y) \cdot V_H(Y) + R(Y) \tag{C.1}$$

and we want to verify that $\sum_{Y\in H} F(Y) = \sigma$, $R(Y)$ is the remainder after dividing $F(Y)$ with the vanishing polynomial and that $R(0) = \sigma/|H|$. Suppose the protocol proceeds as follows

- The prover computes $h(Y)$ and $g(Y) = R(Y) - R(0)$ and sends them to the verifier.
- The verifier generates a random challenge $\alpha \in \mathbb{F}$ and sends it to the prover.
- The prover evaluates $F(\alpha), g(\alpha), h(\alpha)$ and sends them to the verfier.
- The verifier checks

$$F(\alpha) \stackrel{?}{=} h(\alpha) \cdot V_H(\alpha) + g(\alpha) + \frac{\sigma}{|H|} \tag{C.2}$$

Suppose there exists a Malicious prover whose sum evaluates to $\sum_{Y\in H} F(Y) = \sigma'$. Since in the check (C.4), g is used as it is sent by the prover, He/she can trick the verifier into passing (C.4) by manipulating the constant term function g. For instance, if the Malicious prover constructs

$$\tilde{g}(Y) = \frac{\sigma' - \sigma}{|H|} + g(Y) \tag{C.3}$$

evaluates in the random value α sent by the verifier Then the verifier will be checking

$$\begin{aligned} F(\alpha) &\stackrel{?}{=} h(\alpha) \cdot V_H(\alpha) + \tilde{g}(\alpha) + \frac{\sigma}{|H|} \\ &\stackrel{?}{=} h(\alpha) \cdot V_H(\alpha) + g(\alpha) + \frac{\sigma'}{|H|} \end{aligned} \tag{C.4}$$

where we substituted (C.3) in the second line. The check will pass because the $\sum_{Y\in H} F(Y) = \sigma'$ in the malicious prover computation.

Thus in the univariate sumcheck, the soundness issue is resolved by multiplying the $g(Y)$ sent by the prover with Y as in (5.14). This prevents any maniputlation of the constant term in the RHS of (5.14), and the check will truly pass if and only if $\sum_{Y\in H} F(Y) = \sigma$.

References

[1] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs." Cryptology ePrint Archive, Report 2019/1047, 2019. https://ia.cr/2019/1047.

[2] J. Groth, "On the size of pairing-based non-interactive arguments." Cryptology ePrint Archive, Paper 2016/260, 2016. https://eprint.iacr.org/2016/260.

[3] A. Novakovic and K. Gurkan, "Groth16 malleability." https://geometry.xyz/notebook/groth16-malleability.

[4] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers, "Updatable and universal common reference strings with applications to zk-snarks." Cryptology ePrint Archive, Paper 2018/280, 2018. https://eprint.iacr.org/2018/280.

[5] T. labs. https://tlu.tarilabs.com/cryptography/rank-1.

[6] E. Ben-Sasson, A. Chiesa, L. Goldberg, T. Gur, M. Riabzev, and N. Spooner, "Linear-size constant-query iops for delegating computation." Cryptology ePrint Archive, Paper 2019/1230, 2019. https://eprint.iacr.org/2019/1230.

[7] A. Kate, G. M. Zaverucha, and I. Goldberg, *Constant-size commitments to polynomials and their applications*, in *Advances in Cryptology - ASIACRYPT 2010* (M. Abe, ed.), (Berlin, Heidelberg), pp. 177–194, Springer Berlin Heidelberg, 2010.

[8] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, *J. ACM* **39** (oct, 1992) 859–868.

[9] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward, "Aurora: Transparent succinct arguments for r1cs." Cryptology ePrint Archive, Report 2018/828, 2018. https://ia.cr/2018/828.

[10] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and transparent recursive proofs from holography." Cryptology ePrint Archive, Report 2019/1076, 2019. https://ia.cr/2019/1076.