Ligerito: A Small and Concretely Fast Polynomial Commitment Scheme

Andrija Novakovic
anovakovic@baincapital.com

Guillermo Angeris
gangeris@baincapital.com

May 2025

Abstract

In this note we present Ligerito, a small and practically fast polynomial commitment and inner product scheme. For the case of univariate and multilinear polynomial evaluations, the scheme has a proof size of $\sim \log(N)^2 / \log\log(N)$ up to constants and for a large enough field, where N is the size of the input. Ligerito is also fast on consumer hardware: when run on an M1 MacBook Pro for a polynomial with 2^{24} coefficients over a 32-bit binary field, our Julia prover implementation has a proving time of 1.3 seconds and a proof size of 255 KiB. Ligerito is also relatively flexible: any linear code for which the rows of the generator matrix can be efficiently evaluated can be used. Such codes include Reed–Solomon codes, Reed–Muller codes, among others. This, in turn, allows for a high degree of flexibility on the choice of field and can likely give further efficiency gains in specific applications.

1 Introduction

Polynomial commitment schemes such as KZG [KZG10], Ligero [Ame+17], FRI [Ben+18], BaseFold [ZCF23], Blaze [Bre+24], and WHIR [Arn+24] have become workhorses for zero-knowledge proofs and other applications such as blockchain scaling [EMA25]. In many cases (with the notable exception of Ligero), state-of-the-art schemes require the use of specific error correcting codes, such as Reed–Solomon codes, or require fields with particular properties. In this note, we present Ligerito, a new polynomial commitment scheme and inner product argument, which works over essentially any code that has the following property: any row of the code's generator matrix can be efficiently evaluated. While this flexibility is nice, we show that, even when using basic Reed–Solomon codes over binary fields, Ligerito has small proofs and fast in practice on standard consumer hardware. This scheme also allows for the relatively 'rich' language of sumcheck to express more complicated constraints (such as arbitrary inner products with the coefficients of the provided polynomial) with nearly no additional overhead for the prover or the proof size. Finally, Ligerito is asymptotically small with a proof size of $\sim \log(N)^2 / \log\log(N)$, up to constants, for a large enough field, where N

is the size of the input. The proving time is, essentially, proportional to the encoding time of the codes used.

Structure of this note. The rest of this note is structured as follows. In section 2, we describe the notation and conventions we will use throughout the note. In sections 3 and 4, we describe the matrix-vector product protocol and partial sumcheck protocols, which we will use as black boxes in the construction of Ligerito. In section 5, we describe the matrix-vector product protocol with partial sumcheck, which is a 'simple' way of merging the matrix-vector product and sumcheck protocols, and serves as a stepping stone for the final protocol. In section 6, we describe the Ligerito protocol in its entirety and provide a simple proof of its error bounds, ending with a small set of practical experiments in section 7.

2 Notation and conventions

We use the notation and conventions of [EA23] for simplicity. While familiarity with the Ligero protocol [Ame+17] is not strictly necessary to read this note, we refer readers to [AER24] and [DG24] for proofs which we will rely on throughout.

2.1 Basic notation

Let $\mathbf{F}$ denote some finite field. We will sometimes make use of the fact that $\mathbf{F}' \subseteq \mathbf{F}$ is a subfield of $\mathbf{F}$. Throughout this note, we use uppercase letters such as X and G to refer to matrices, use lowercase letters to refer to column vectors, such that $v \in \mathbf{F}^n$ denotes an n-vector and $G \in \mathbf{F}^{m \times n}$ denotes an $m \times n$ matrix, each over the field $\mathbf{F}$. In general $G \in \mathbf{F}^{m \times n}$ will refer to some (linear) code with distance $d > 0$, where n is called the *message length* and m is called the *block length*. We write $\|v\|$ to denote the Hamming weight of a vector v (*i.e.*, the number of nonzero entries of v) and $\|X\|$ to denote the number of nonzero rows of a matrix X. Note that this notation is consistent if we interpret a vector v as an $n \times 1$ matrix, as we generally will in this note.

2.2 Matrix and polynomial notation

Since we constantly reshape matrices and vectors throughout the note, we will overload notation to simplify and clean up the exposition.

Matrices to vectors (and back). For a given matrix $\tilde{X} \in \mathbf{F}^{m \times n}$, we write $\mathbf{vec}(\tilde{X})$ to the mn-vector corresponding to stacking the columns of the matrix $\tilde{X}$ into a single column vector. Similarly, for a given vector $v \in \mathbf{F}^{mn}$, we write $\mathbf{Mat}(v)$ to denote matrix corresponding to reshaping the vector v into a column-major $m \times n$ matrix. In general, it should always be clear from context what the dimensions of this matrix are such that the expressions correctly parse. When there is any ambiguity, we will be explicit about the dimensions of the resulting matrix. We will also sometimes write $\mathbf{Mat}(\tilde{X})$ for the reshaping of a matrix $\tilde{X}$ to the appropriate dimensions. (This is equivalent to writing $\mathbf{Mat}(\mathbf{vec}(\tilde{X}))$, which is now well-defined, but this is rather cumbersome.)

Kronecker product. We will define the Kronecker product $\otimes$, defined for two matrices $A \in \mathbf{F}^{m \times n}$ and $B \in \mathbf{F}^{m' \times n'}$ as

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1n}B \\ A_{21}B & A_{22}B & \dots & A_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1}B & A_{m2}B & \dots & A_{mn}B \end{bmatrix}.$$

The resulting matrix $A \otimes B$ is therefore of size $mm' \times nn'$. To avoid overly complicated expressions we will write, for a given vector $r \in \mathbf{F}^k$, the 'barred' vector

$$\overline{r} = (1 - r_1, r_1) \otimes \dots \otimes (1 - r_k, r_k), \tag{1}$$

which is a (column) vector of size 2^k.

Polynomials and vectors. Let $v \in \mathbf{F}^{2^{k'}}$ be a general vector with elements in $\mathbf{F}$. It will be very convenient in what follows to overload notation and write, for $z \in \mathbf{F}^{k'}$,

$$v(z) = v^T \overline{z} = v^T((1 - z_1, z_1) \otimes \dots \otimes (1 - z_k, z_k)). \tag{2}$$

(Note that we have used the barred notation (1) for z.) In other words, when writing v as a function, evaluated at $z \in \mathbf{F}^{k'}$, we interpret the vector v as the coefficients of a multilinear polynomial in k variables and evaluate it at the point z. This lets us define the *partial evaluation* of v in the first $k' \leq k$ variables at a point $r \in \mathbf{F}^{k'}$, as

$$y = \mathbf{Mat}(v)\overline{r}. \tag{3}$$

Note that, from our definition (1), $\overline{r}$ is a vector of size $2^{k'}$. This means that we can infer the shape of $\mathbf{Mat}(v)$ to be a $2^{k-k'} \times 2^{k'}$ matrix corresponding to a column-major reshaping of the elements of v. This, in turn, means that y is a $2^{k-k'}$-vector, which is the partial evaluation of v (interpreted as a polynomial) when the first k variables are equal to r, with the rest free; *i.e.*,

$$y(z) = v(r, z),$$

for all $z \in \mathbf{F}^{k-k'}$. (As before, we are overloading notation as in (2) for both y and v.) Finally, for matrices $\tilde{X} \in \mathbf{F}^{2^k \times 2^{k'}}$ it will also be convenient to write

$$\tilde{X}(z) = \big(\mathbf{vec}(\tilde{X})\big)(z),$$

where $\mathbf{vec}(\tilde{X})$ corresponds to stacking the columns of the matrix $\tilde{X}$ into a size $2^{k+k'}$ vector.

3 Matrix-vector product protocol

At a high level, the main tool we will use is the Ligero protocol [Ame+17], viewed as a matrix-vector product protocol [AER24], with the logarithmic randomness of [DP24]. The Ligero protocol allows us to compute $\tilde{X}\overline{r}$, where r will be some randomness we specify next, for any

matrix $\tilde{X} \in \mathbf{F}^{n \times n'}$ with communication proportional to $n + n'$. To simplify the exposition, we will assume that $n' = 2^k$, though the results hold more broadly (by, *e.g.*, padding the matrix $\tilde{X}$). We will also assume there is a known code $G \in \mathbf{F}^{m \times n}$ with distance $d > 0$. We break the protocol down into two parts—the algorithm performed by the prover and the algorithm performed by the verifier.

Prover algorithm. Here we describe the prover's algorithm for the matrix-vector product protocol. We assume that the prover algorithm has a matrix $\tilde{X} \in \mathbf{F}^{n \times n'}$. The algorithm is as follows:

1. Encode the columns of $\tilde{X}$ to get $X = G\tilde{X}$
2. Commit to the rows of X, via, say, a Merkle commitment
3. Receive some random vector $r \in \mathbf{F}^k$
4. Compute and send $y_r = \tilde{X}\overline{r}$

We will call the first two steps of the prover's algorithm the *commitment phase*.

Verifier algorithm. Here we describe the verifier's algorithm for the matrix-vector product protocol.

1. Receive the Merkle commitments to the rows of some opaque matrix X
2. Sample and send some random vector $r \in \mathbf{F}^k$, receiving some vector $y_r \in \mathbf{F}^n$
3. Uniformly sample $S \subseteq \{1, ..., m\}$, of some fixed size $|S|$
4. Verify that $X_S\overline{r} = G_S y_r$

Similar to the previous, we say that the first step is the *commitment phase* of the verifier's algorithm. If any step of this algorithm fails, the verifier rejects the proof, otherwise it accepts. The randomness used for $\overline{r}$ here will be *logarithmic* randomness, originally defined in [DP24] and improved in [AER24], though it can be uniform randomness, as in [Ame+17]. (In the special case that G is Reed–Solomon, the randomness can also be the powers of some uniformly random element [Ben+23].) For the remainder of this note, we will use logarithmic randomness, and mostly (but not always) assume that G is a Reed–Solomon code for which the results of [DG24] are the best-known and we present them below.

Guarantees. It is clear that, if the prover follows the above protocol, the verifier's algorithm will always succeed. (This property is often called *completeness*.) Additionally, from [AER24] and [DG24], we know that the verifier, if all steps pass, has the following guarantees:

1. There exists some unique matrix $\tilde{X}$ such that $\|X - G\tilde{X}\| < d/2$, where d is the distance of the Reed–Solomon code G

2. The vector y_r satisfies $y_r = \tilde{X}\overline{r}$; in other words, the received y_r is the partial evaluation of $\tilde{X}$, interpreted as a polynomial, where the first k variables are equal to r

All of these are true except with error probability no more than

$$\left(\frac{m-n-1}{2m}\right)^{|S|} + \frac{mk}{|\mathbf{F}|}, \tag{4}$$

in the special case that $G \in \mathbf{F}^{m \times n}$ is a Reed–Solomon code. In the case that the code G is a general linear code, the error probability is

$$\left(1 - \frac{d}{3m}\right)^{|S|} + \frac{dk}{3|\mathbf{F}|}, \tag{5}$$

where d is the distance of the code G.

Communication complexity. The communication complexity of this protocol is easy to tally up. The verifier received an n-vector (namely, y_r) and the $|S|$ requested rows of X. This totals to

$$(n + |S|n') \log_2(|\mathbf{F}|). \tag{6}$$

The prover also has to send the corresponding openings of each of the $|S|$ rows, which, if using a binary Merkle commitment, totals to

$$|S| \log_2(m). \tag{7}$$

(In a practical protocol, the concrete numbers can be made slightly better in expectation by using a Merkle tree multi-opening.) Letting $N = nn'$ be the total number of entries in $\tilde{X}$, then we may set $n = n' = \sqrt{N}$ to get a communication complexity of

$$|S| \log_2(|\mathbf{F}|)\sqrt{N}, \tag{8}$$

not including the binary Merkle openings, which are small relative to this number as N becomes large.

Discussion. The matrix-vector product protocol is relatively simple, and we will use it as a black box in what follows. Note that we only access y_r via the inner products with the $|S|$ rows of G, which is an observation we will use next.

4 Partial sumcheck protocol

The sumcheck protocol is a well-known interactive protocol for computing an inner product between two vectors. We will show a (only small) generalization here, which we use later in this note.

4.1 High level protocol

Given some known vector $w \in \mathbf{F}^{2^k}$, which we interpret as an k variable multilinear polynomial, and some claimed inner product $\alpha \in \mathbf{F}$, the sumcheck protocol is an interactive protocol for reducing a claims of the form

$$\sum_{z \in \{0,1\}^k} w(z)\tilde{X}(z) = \alpha \tag{9}$$

to a claim of the form

$$\sum_{z' \in \{0,1\}^{k'}} w(r, z')\tilde{X}(r, z') = \alpha'. \tag{10}$$

For some α', randomness r which we specify next, and $k' < k$. (In other words, we reduce the sum over k variables to a sum over k' variables and some partial evaluations of w and $\tilde{X}$.) Another way to view (9) and the protocol is that we begin with an inner product between $\tilde{X}$ and the vector w:

$$w^T \, \mathbf{vec}\big(\tilde{X}\big) = \alpha,$$

which is then reduced to a new inner product between the partial evaluations

$$(\mathbf{Mat}(w)\overline{r})^T \big(\tilde{X}\overline{r}\big) = \alpha',$$

where $\mathbf{Mat}(w)$ is the reshaping of w to the appropriate dimensions. We will make use of this particular observation later in the note.

4.2 Partial sumcheck protocol

As before, we will describe the prover and verifier algorithms separately. In this construction, we will reduce the original sum over n variables to a sum over n' variables.

Prover algorithm. The prover algorithm is as follows.

1. For $i = 1, ..., k - k'$
    1. Send the polynomial $s_i(t) = \sum_{z \in \{0,1\}^{k-i}} w(r_1, ..., r_{i-1}, t, z)\tilde{X}(r_1, ..., r_{i-1}, t, z)$
    2. Receive a random challenge $r_i \in \mathbf{F}$, to be used in the next round

Verifier algorithm. The verifier algorithm is as follows.

1. Set $s_0 = \alpha$ to be a constant polynomial and $r_0 = 0$ for convenience
2. For each of $i = 1, ..., k - k'$:
    1. Send some uniformly sampled $r_{i-1} \in \mathbf{F}$

2. Receive some degree 2 polynomial $s_i : \mathbf{F} \to \mathbf{F}$
3. Verify that $s_i(0) + s_i(1) = s_{i-1}(r_{i-1})$

If all verifications pass, the verifier accepts the proof. Otherwise, it rejects. Note that the verifier does not need to know neither w nor $\tilde{X}$ to run this algorithm.

Guarantees. The guarantees of the partial sumcheck is as follows. If the verifier knows that, for a new uniformly randomly sampled $r_{k-k'} \in \mathbf{F}$,

$$s_{k-k'}(r_{k-k'}) = \alpha' = \sum_{z' \in \{0,1\}^{k'}} w(r_1, ..., r_{k-k'}, z')\tilde{X}(r_1, ..., r_{k-k'}, z') \tag{11}$$

then, after running the verifier algorithm it is guaranteed that

$$\sum_{z \in \{0,1\}^k} w(z)\tilde{X}(z) = \alpha,$$

except with additional error probability no more than

$$\frac{2(k-k')}{|\mathbf{F}|}. \tag{12}$$

In particular, if $k' = 0$, this reduces to the standard sumcheck since the terminal condition (11) is simply to evaluate the product of w and $\tilde{X}$ at $(r_1, ..., r_k)$:

$$s_k(r_k) = w(r_1, ..., r_k)\tilde{X}(r_1, ..., r_k).$$

It is very easy (and relatively standard) to see that if the prover follows the prover algorithm, the verifier will always accept the proof. It is also easy to see the (partial) converse: if all steps of the verifier algorithm pass, including the final check (11), then, the verifier knows that the inner product $w^T \mathbf{vec}(\tilde{X}) = \alpha$, except with error probability no more than specified in (12). The proof is identical to the standard sumcheck protocol proof and we refer readers to [Tha22], chapter 4.

Discussion. It will be convenient to note that equation (11) can be written in matrix form as

$$s_{k-k'}(r_{k-k'}) = (\mathbf{Mat}(w)\overline{r})^T\big(\tilde{X}\overline{r}\big),$$

where, as before, $\overline{r} = (1 - r_1, r_1) \otimes ... \otimes (1 - r_{k-k'}, r_{k-k'})$.

4.3 Observations

In this section, we make some observations about the partial sumcheck protocol which we use in the protocol that follows.

Inner products. As mentioned before, the sumcheck protocol can be viewed as a protocol for verifying that the inner product between two vectors, w and $\mathbf{vec}(\tilde{X})$ is, say, α; written out, this means that

$$w^T \mathbf{vec}(\tilde{X}) = \sum_{z \in \{0,1\}^k} w(z)\tilde{X}(z) = \alpha.$$

Batching evaluations. If we are given a list of q vectors $w_1, ..., w_q \in \mathbf{F}^{2^k}$, and claimed inner products $\alpha_1, ..., \alpha_q \in \mathbf{F}$, we can 'batch' the evaluations and proofs into a single partial sumcheck protocol instance. The observation is very simple: once $\alpha_1, ..., \alpha_q$ are received, the verifier can draw some uniform randomness $\beta_1, ..., \beta_q \in \mathbf{F}$ and simply run the partial sumcheck protocol over the new vector

$$\tilde{w} = \beta_1 w_1 + ... + \beta_q w_q, \tag{13}$$

and the new claimed inner product

$$\tilde{\alpha} = \beta_1 \alpha_1 + ... + \beta_q \alpha_q.$$

holding $\tilde{X}$ constant. This adds only a small probability of error, namely $1/|\mathbf{F}|$, to the error probability of the protocol, if we use uniform and independent randomness for each β_i. We will call this new vector w of (13), resulting from the batching process, the *batched vector* or *batched polynomial*.

Gluing sumchecks. One observation is that, given a partial sumcheck protocol instance, where the we are reducing a sum over k variables to a sum over k'' variables, it is possible to, at any round k' with $k'' < k' < k$, view the partial sumcheck protocol from k to k'' variables as two 'glued' instances of the sumcheck protocol: one from k to k' variables and one from k' to k'' variables. It is then possible, via the batching observation above, to batch two inner products: one sum over k variables and one sum over k' variables (with the rest of the variables fixed by the randomness of the first partial sumcheck).

Complete batched and glued sumcheck. For completeness, we describe the batched and glued sumcheck below. We assume that the prover has some matrix $\tilde{X} \in \mathbf{F}^{2^k \times 2^{k'}}$ (which itself could be a partial evaluation of some larger matrix). The prover algorithm is as follows:

1. Receive some vectors $w_1, ..., w_q \in \mathbf{F}^{2^{k+k'}}$
2. Send evaluations $\alpha_i = \mathbf{vec}(\tilde{X})^T w_j$ for $j = 1, ..., q$
3. Receive some batching randomness $\beta \in \mathbf{F}^q$
4. Construct a batched vector $\tilde{w} = \sum_{j=1}^{q} \beta_j w_j$
5. For $i = 1, ..., k - k'$

1. Send the polynomial $s_i(t) = \sum_{z \in \{0,1\}^{k-i}} \tilde{w}(r_1, ..., r_{i-1}, t, z)\tilde{X}(r_1, ..., r_{i-1}, t, z)$
2. Receive a random challenge $r_i \in \mathbf{F}$, to be used in the next round, if $i < k - k'$

For the verifier:

1. Send some vectors $w_1, ..., w_q \in \mathbf{F}^{2^{k+k'}}$
2. Receive (claimed) evaluations α_j for $j = 1, ..., q$
3. Send batching randomness $\beta \in \mathbf{F}^q$
4. Construct a batched vector $\tilde{w} = \sum_{j=1}^{q} \beta_j w_j \in \mathbf{F}^{2^{k+k'}}$
5. Set $s_0 = \sum_{j=1}^{q} \beta_j \alpha_j$ to be a constant polynomial and $r_0 = 0$ for convenience
6. For $i = 1, ..., k - k'$:
    1. Send some uniformly sampled $r_{i-1} \in \mathbf{F}$
    2. Receive some degree 2 polynomial $s_i : \mathbf{F} \to \mathbf{F}$
    3. Verify that $s_i(0) + s_i(1) = s_{i-1}(r_{i-1})$

If all verifier checks pass, the verifier accepts the proof. Additionally, if the verifier knows that, for a new uniformly randomly sampled $r_{k-k'} \in \mathbf{F}$, the following is true:

$$s_{k-k'}(r_{k-k'}) = \sum_{z' \in \{0,1\}^{k'}} \tilde{w}(r_1, ..., r_{k-k'}, z')\tilde{X}(r_1, ..., r_{k-k'}, z'), \tag{14}$$

then the verifier is guaranteed that

$$\sum_{z \in \{0,1\}^k} w_j(z)\tilde{X}(z) = \alpha_j,$$

with error probability no more than

$$\frac{2(k-k')}{|\mathbf{F}|} + \frac{1}{|\mathbf{F}|},$$

if the β are uniformly randomly chosen. It is important to note that the vectors w_j can be themselves partial evaluations of some larger vectors and that the verifier only needs access to $\tilde{X}$ via the terminal condition (14). Note also that (from the gluing observation) the final check (14) can itself be reduced via the sumcheck protocol to a sum over even fewer variables using the above protocol.

5 Matrix-vector product with partial sumcheck

We will now show a simple combination of the matrix-vector product protocol and the partial sumcheck protocol. This construction allows a prover to evaluate and prove that some inner product between $\mathbf{vec}(\tilde{X})$, where $\tilde{X}$ is some matrix known to the prover, and some public vector w, is equal to α. This protocol essentially uses the following observation: the vector y_r received by the verifier in the matrix-vector product protocol is, itself, a partial evaluation of $\tilde{X}$ at some point g_r. This is essentially the right hand side of the final check of the partial sumcheck protocol in (11).

Prover algorithm. Given some matrix $\tilde{X} \in \mathbf{F}^{2^k \times 2^{k'}}$, the prover algorithm is as follows.

1. Run the commitment phase (steps 1-2) of the matrix-vector product on $\tilde{X}$
2. Receive some vector $w \in \mathbf{F}^{2^{k+k'}}$
3. Run the partial sumcheck prover protocol on the vector w, the matrix $\tilde{X}$, and the claimed inner product α to reduce the sum over $(k + k')$ to k variables. In doing so, receive randomness $r_1, ..., r_{k'-1}$ and send final polynomial $s_{k'}$
4. Receive $r_{k'} \in \mathbf{F}$ and construct $\overline{r} = (1 - r_1, r_1) \otimes ... \otimes (1 - r_{k'}, r_{k'})$
5. Send $y_r = \tilde{X}\overline{r}$, like the matrix-vector product protocol

Verifier algorithm. We assume that the verifier has some vector $w \in \mathbf{F}^{2^{k+k'}}$ over which they would like to compute some inner product with the (unknown and opaque) matrix $\tilde{X}$. The verifier algorithm is as follows.

1. Receive the Merkle commitments to the rows of some opaque matrix X
2. Send the vector $w \in \mathbf{F}^{2^{k+k'}}$ and receive claimed evaluation α
3. Run the partial sumcheck verifier protocol with the claimed evaluation α, receiving some degree 2 polynomial $s_{k'}$ and some randomness $r_1, ..., r_{k'-1}$
4. Sample and send $r_{k'} \in \mathbf{F}$ and construct $\overline{r} = (1 - r_1, r_1) \otimes ... \otimes (1 - r_{k'}, r_{k'})$
5. Receive some vector $y_r \in \mathbf{F}^{2^k}$
6. Uniformly sample $S \subseteq \{1, ..., m\}$, of some fixed size $|S|$
7. Verify that $X_S\overline{r} = G_S y_r$, therefore conclude that $y_r = \tilde{X}\overline{r}$ from the matrix-vector product protocol guarantees, where $\tilde{X}$ is the unique matrix with $\|X - G\tilde{X}\| < d/2$
8. Verify the partial sumcheck's final check $s_{k'}(r_{k'}) = y_r^T \, \mathbf{Mat}(w)\overline{r}$

If all check pass then the verifier accepts the proof. Otherwise, it rejects. Step 8 can be easily interpreted as computing the right hand side of the final check (11):

$$y_r^T \, \mathbf{Mat}(w)\overline{r} = \sum_{z' \in \{0,1\}^k} y_r(z') w(r_1, ..., r_{k'}, z') = \sum_{z' \in \{0,1\}^k} \tilde{X}(r_1, ..., r_{k'}, z') w(r_1, ..., r_{k'}, z'),$$

where the second equality comes from the guarantee given in step 7 of the verifier algorithm. Note that the total error probability of the protocol can be bounded by the sum of the error probabilities of the matrix-vector product protocol and the partial sumcheck protocol:

$$\left(\frac{m-n-1}{2m}\right)^{|S|} + \frac{mk'}{|\mathbf{F}|} + \frac{2k'}{|\mathbf{F}|}, \tag{15}$$

in the case that the code G is a Reed–Solomon code, while in general, the error probability is

$$\left(1 - \frac{d}{3m}\right)^{|S|} + \frac{dk'}{3|\mathbf{F}|} + \frac{2k'}{|\mathbf{F}|}, \tag{16}$$

where d is the distance of the code G; note that the first two terms are exactly those of (5).

Guarantees and discussion. The guarantees of the above protocol are essentially the same as those of the matrix-vector product protocol—*i.e.*, there exists a unique matrix $\tilde{X}$ closest to the committed matrix X and that of step 7—with the additional conclusion that, indeed

$$w^T \, \mathbf{vec}\big(\tilde{X}\big) = \alpha;$$

in other words, that the inner product of this unique $\tilde{X}$ with w is, indeed, α.

As before, we can also use the observation above to batch multiple $\{w_i\}$ vectors and so on. We note that this style of protocol is not new and has been used, in slightly different forms, in other contexts such as [Fs24] and [DP25].

6 Ligerito

In this section, we present Ligerito, the main protocol, which shows how to, roughly speaking, recurse the protocol presented in the previous section.

6.1 High level idea

The high level idea for this protocol is very simple. In the general matrix-vector product protocol and its extension with the partial sumcheck, the verifier receives all of y_r in order to verify that $G_S y_r$ matches the opened rows of X; *i.e.*, that $X_S \overline{r} = G_S y_r$, as in step 7 of the verifier protocol. The main thing we observe is that the verifier only accesses the received partial evaluation y_r via the $|S|$ inner products with the rows of G and the final inner product of step 8, $y_r^T \, \mathbf{Mat}(w)\overline{r}$. Note, of course, that the prover does not need to send y_r: it can, instead, commit to $\mathbf{Mat}(y_r)$ as a new matrix, and then, again, use the previous section's protocol to prove the desired inner products with the rows of G. We can continue to recurse

this, say ℓ, times, where, in the last step, we simply run the matrix-vector product with the partial sumcheck of the previous section.

6.2 Protocol algorithms

For now, it will be useful to specify a parameter $\ell \geq 2$ which will determine the number of rounds we will run; we will then later show an optimal choice of ℓ. The protocol requires, as in the previous protocol, a sequence of dimensions $k_1, ..., k_\ell$ and $k'_1, ..., k'_\ell$ with $k_{i+1} + k'_{i+1} = k_i$, and a number of codes $G_i \in \mathbf{F}^{m_i \times 2^{k_i}}$ for $i = 1, ..., \ell - 1$, each with distance $d_i > 0$. The only requirements for these code matrices will be that it is efficient to evaluate a (generally random) row of each G_i. This is true for a large number of code matrices such as Reed–Solomon codes, Reed–Muller codes, among many others.

Prover algorithm. The prover has some matrix $\tilde{X}_1 \in \mathbf{F}^{2^{k_1} \times 2^{k'_1}}$ along with the previously mentioned public data. After committing to $\tilde{X}_1$, the prover will receive some $w \in \mathbf{F}^{2^{k_1 + k'_1}}$ and will show that $w^T \, \mathbf{vec}(\tilde{X}_1) = \alpha$.

1. For $i = 1, ..., \ell - 1$
    1. Run the commitment phase (steps 1-2) of the matrix-vector product on $\tilde{X}_i$
    2. If $i = 1$ receive $w \in \mathbf{F}^{k_1 + k'_1}$ and set $\tilde{w}_0 = w$
    2. If $i \geq 2$, receive the sampled indices $S_{i-1} \subseteq \{1, ..., m_{i-1}\}$
    3. If $i \geq 2$, compute the expected symbols $v_{i-1} = (G_{i-1} \, \mathbf{vec}(\tilde{X}_i))_{S_{i-1}}$, this is $|S_{i-1}|$ inner products with $\tilde{X}_i$
    4. Run the batched and glued sumcheck
        1. If $i \geq 2$, send the $|S|$ expected symbols v_{i-1}
        2. Batch the inner product evaluations of the v_{i-1} with $\tilde{w}_{i-1}$, getting a new batched vector, $\tilde{w}_i$
        3. Run the partial sumcheck protocol to reduce $\tilde{w}_i^T \, \mathbf{vec}(\tilde{X}_i)$ by k_i variables, receiving randomness $r_i \in \mathbf{F}^{k_i}$) and sending final partial sum s_i
    5. Set $\tilde{X}_{i+1} = \mathbf{Mat}(\tilde{X}_i \overline{r}_i) \in \mathbf{F}^{2^{k_{i+1}} \times 2^{k'_{i+1}}}$ to be committed in the next iteration
2. Send $y_\ell = \mathbf{vec}(\tilde{X}_\ell)$

Verifier algorithm. The verifier algorithm is as follows.

1. For $i = 1, ..., \ell - 1$
    1. Receive a Merkle commitment to the rows of some opaque matrix X_i
    2. If $i = 1$, send w, set $\tilde{w}_0 = w$ for convenience, receive evaluation α, set $s_0 = \alpha$

3. If $i \geq 2$:
    1. Uniformly sample and send $S_{i-1} \subseteq \{1, ..., m_{i-1}\}$ of fixed size $|S_{i-1}|$
    2. Receive purported vector of symbols $v_{i-1} \in \mathbf{F}^{|S_{i-1}|}$
    3. Verify that $(X_{i-1}\overline{r}_{i-1})_{S_{i-1}} = v_{i-1}$
4. Run the batched partial sumcheck verifier of section 4:
    1. Batch the following checks into a new batched vector $\tilde{w}_i$:
        - The running sum $s_{i-1}((r_{i-1})_{k_{i-1}})$ should be equal to the inner product with $\tilde{w}_{i-1}$
        - If $i \geq 2$, then v_{i-1} should be equal to the inner products with each row of $(G_{i-1})_{S_{i-1}}$
    2. Run the partial sumcheck protocol over the batched vector $\tilde{w}_i$, sending randomness $r_i \in \mathbf{F}^{k'_i}$ and receiving the final degree-2 polynomial s_i

2. Receive vector y_ℓ
3. Uniformly sample $S_{\ell-1} \subseteq \{1, ..., m_{\ell-1}\}$ of fixed size $|S_{\ell-1}|$
4. Verify that $(X_{\ell-1}\overline{r}_{\ell-1})_{S_{\ell-1}} = (G_{\ell-1})_{S_{\ell-1}} y_\ell$
5. Verify that $s_{\ell-1}((r_{\ell-1})_{k_{\ell-1}}) = \tilde{w}_\ell^T y_\ell$

Discussion. Note that when $\ell = 2$, this reduces exactly to both the prover and verifier algorithms for the matrix-vector product protocol with the rtial sumcheck protocol of section 5. Remco Bloemen and Giacomo Fenzi have commented that this protocol is structurally similar to the WHIR protocol of [Arn+24], though we note that Ligerito uses general linear codes, the logarithmic randomness of [DP24], and, as far as we can tell, results in concretely different (and smaller) numbers in the unique decoding regime. A natural open question is whether there is a simple generalization of both protocols that can recast them in a common framework. Another important observation is that the verifier needs to compute $\tilde{w}_\ell$ (which corresponds to a number of batched and partially-evaluated vectors from the previous rounds, some of which may be very large) in order to complete the final check. While this is sublinear in the size of the original matrix $\tilde{X}_1$ and fast in practice, some definition of succinctness require that the verifier does at most logarithmic work in the size of the original matrix. We show how to achieve this in the case where the G_i are tensorizable codes (such as Reed–Solomon or Reed–Muller codes) in section 6.6.

6.3 Guarantees

The guarantees of this protocol are as follows. If the prover follows the prover algorithm, the verifier will always accept the proof. (This is easy to see from the verifier algorithm.) On the other hand, if the verifier follows the verifier algorithm and accepts, then the verifier is guaranteed that (1) there exists some unique matrix $\tilde{X}_i$ such that

$$\|X_i - G_i\tilde{X}_i\| < \frac{d_i}{2},$$

(*i.e.*, the prover 'knows' a unique matrix $\tilde{X}_i$ which is close to the committed matrix X_i) for $i = 1, ..., \ell - 1$, and (2) that the inner product of the unique matrix $\tilde{X}_1$ with the vector w is equal to the claimed evaluation α:

$$w^T \, \mathbf{vec}\big(\tilde{X}_1\big) = \alpha.$$

Finally, the verifier guarantees that

$$y_\ell = \tilde{X}_{\ell-1}\overline{r}_{\ell-1} = \mathbf{Mat}\big(\tilde{X}_{\ell-2}\big)(\overline{r}_{\ell-2} \otimes \overline{r}_{\ell-1}) = ... = \mathbf{Mat}\big(\tilde{X}_1\big)(\overline{r}_1 \otimes ... \otimes \overline{r}_{\ell-1}).$$

Or, in other words, that y_ℓ is the partial evaluation of $\tilde{X}_1$ over randomness $(\overline{r}_1 \otimes ... \otimes \overline{r}_{\ell-1})$. Both statements are true except with error probability no more than the sum of the error probabilities of the matrix-vector product protocol and the partial sumcheck protocol:

$$\begin{aligned}\sum_{i=1}^{\ell-1}\left(\frac{2k_{i-1}}{|\mathbf{F}|} + \frac{|S_i|+1}{|\mathbf{F}|} + \left(\frac{m_i - 2^{k_i} - 1}{2m_i}\right)^{|S_i|} + \frac{m_i k_i}{|\mathbf{F}|}\right) \\ + \frac{2k_{\ell-1}}{|\mathbf{F}|} + \left(\frac{m_\ell - 2^{k_\ell} - 1}{2m_\ell}\right)^{|S_\ell|} + \frac{m_\ell k_\ell}{|\mathbf{F}|},\end{aligned} \tag{17}$$

in the case that the codes G_i are Reed–Solomon codes. Here the first two terms in the sum correspond to the error from the sumcheck and the second part comes from the matrix-vector product protocol. The tail term is simply the term in the sum without the error incurred by batching the evaluations. In the case where the codes G_i are general linear codes, the error probability is

$$\begin{aligned}\sum_{i=1}^{\ell-1}\left(\frac{d_i k_{i-1}}{3|\mathbf{F}|} + \frac{|S_i|+1}{|\mathbf{F}|} + \left(\frac{d_i}{3m_i}\right)^{|S_i|} + \frac{d_i k_i}{3|\mathbf{F}|}\right) \\ + \frac{2k_{\ell-1}}{|\mathbf{F}|} + \left(1 - \frac{d_\ell}{3m_\ell}\right)^{|S_\ell|} + \frac{d_\ell k_\ell}{3|\mathbf{F}|},\end{aligned} \tag{18}$$

using (16) for the tail term and (5) for the error probabilities of the individual per-round checks.

Proof. The proof essentially follows from the guarantees of the matrix-vector product protocol and the partial sumcheck protocol. We will prove this in the case that the G_i are all Reed–Solomon code matrices. (The general case is essentially identical, except with the error bound of (5) instead.) We proceed by induction on $\ell \geq 2$. The base case, $\ell = 2$, is simply the matrix-vector product with the partial sumcheck protocol of section 5, where the guarantees follow from the guarantees provided in that section. To reiterate those guarantees, the verifier knows that there exists a unique matrix $\tilde{X}_1$ whose encoding is closest to the committed matrix X_1, that $\tilde{X}_1 \overline{r} = \mathbf{vec}(\tilde{X}_2) = y_2$, and that the inner product $w^T \, \mathbf{vec}(\tilde{X}_1) = \alpha$. The error probability of this protocol is exactly that of (15), where $n = 2^{k_1}$ and $k' = k'_1$:

$$\frac{2k'_1}{|\mathbf{F}|} + \left(\frac{m_1 - 2^{k_1} - 1}{2m_1} \right)^{|S|} + \frac{m_1 k'_1}{|\mathbf{F}|}.$$

Now, assume that the guarantees hold for $\ell - 1$: that is, for the matrix X_2 (and, indeed, all 'later' matrices $X_3, ..., X_\ell$) we have that there exists a unique matrix $\tilde{X}_2$ with

$$\|X_2 - G_2 \tilde{X}_2\| < \frac{d_2}{2},$$

that $\tilde{X}_2 \overline{r}_2 = \mathbf{vec}(\tilde{X}_3)$, and that the inner product $\tilde{w}_1^T \, \mathbf{vec}(\tilde{X}_2) = \tilde{\alpha}_1$ holds, where $\tilde{w}_1$ is the batched vector of the previous round and $\tilde{\alpha}_1$ is the expected batched result. From the guarantees of the base case, we know that

$$(G_1)_{S_1} \, \mathbf{vec}\big(\tilde{X}_2\big) = v_1,$$

except with additional error probability no more than

$$\frac{1}{|\mathbf{F}|},$$

from the batching of the evaluations in the sumcheck protocol. The verifier checks, in step 3.3 that

$$(X_1)_{S_1} \overline{r}_1 = v_1,$$

hence

$$(X_1)_{S_1} \overline{r}_1 = (G_1)_{S_1} \, \mathbf{vec}\big(\tilde{X}_2\big),$$

and, from the matrix-vector product protocol guarantees of 3, we know that this implies

$$\|X_1 - G_1 \tilde{X}_1\| < \frac{d_1}{2},$$

and that $\mathbf{vec}(\tilde{X}_1)$ satisfies

$$\mathbf{vec}\big(\tilde{X}_2\big) = \tilde{X}_1 \overline{r}_1,$$

except with (additional) error probability no more than

$$\left(\frac{m_1 - 2^{k_1} - 1}{2m_1}\right)^{|S_1|} + \frac{m_1 k_1}{|\mathbf{F}|}.$$

In other words, $\tilde{X}_2$ is known to be the partial evaluation of the unique matrix $\tilde{X}_1$ at $\overline{r}_1$. This, in turn, means that, from the guarantees of the partial sumcheck protocol, we know that

$$w^T\, \mathbf{vec}\big(\tilde{X}_1\big) = \tilde{w}_0^T\, \mathbf{vec}\big(\tilde{X}_1\big) = \alpha,$$

except with additional error probability no more than

$$\frac{2k_1'}{|\mathbf{F}|},$$

Tallying up the error gives a total error probability of

$$\frac{2k_1'}{|\mathbf{F}|} + \frac{|S_1|+1}{|\mathbf{F}|} + \left(\frac{m_1 - 2^{k_1} - 1}{2m_1}\right)^{|S_1|} + \frac{m_1 k_1}{|\mathbf{F}|} + p_{\text{rest}},$$

where p_{rest} is the total error probability incurred by the inductive case (rounds 2 through ℓ).

6.4 Communication complexity

The communication complexity is also easy to total up and we will assume that the verifier randomness is derived noninteractively (via, *e.g.*, Fiat–Shamir) and hence not sent by the verifier nor counted in the total communication. At round $i < \ell$, the prover sends $|S_{i-1}| + 1$ field elements for the expected symbols v_{i-1} and inner product with $\tilde{w}_i$. It also sends $3k_i$ field elements, 3 elements for each degree-2 polynomial in the sumcheck protocol in step 4 of the prover algorithm. Finally, it sends $|S_{i-1}|$ openings of the rows of X_{i-1}, which total to $|S_{i-1}|2^{k'_{i-1}}$ field elements and $|S_{i-1}|\log_2(m_{i-1})$ hashes for the Merkle openings of the corresponding rows. At round $i = \ell$, the prover sends $\tilde{X}_\ell$ in its entirety, which totals $2^{k_\ell + k'_\ell} = 2^{k_{\ell-1}}$ field elements, along with the $S_{\ell-1}$ row openings and corresponding Merkle proofs. The total communication complexity is therefore

$$\begin{aligned}\sum_{i=2}^{\ell-1} \Big(3k'_{i-1}\log_2(|\mathbf{F}|) + |S_{i-1}|\,\left(\left(2^{k'_{i-1}}+1\right)\log_2(|\mathbf{F}|) + C\log_2(m_{i-1})\right)\Big) \\ + \Big(|S_{\ell-1}|2^{k'_{\ell-1}} + 2^{k_{\ell-1}}\Big)\log_2(|\mathbf{F}|) + C\;|S_{\ell-1}|\log_2(m_{\ell-1}),\end{aligned} \tag{19}$$

where C is the size in bits of the hash function used (in our case, $C = 256$, as we use SHA256).

Asymptotics. Let λ be the number of bits of security and $\mathbf{F}$ be a field with size $|\mathbf{F}| \gg 2^\lambda$. Also set $\log = \log_2$ for notational convenience. The asymptotic communication complexity is, ignoring small terms

$$\left(\sum_{i=1}^{\ell} |S_{i-1}| 2^{k'_{i-1}} \log(|\mathbf{F}|)\right) + 2^{k_{\ell-1}} \log(|\mathbf{F}|) + C\ell \log\left(\frac{N}{\rho}\right),$$

where $N = k_1 + k'_1$ is the total number of entries in $\tilde{X}_1$ and we use the (very coarse) bound $m_i \leq N/\rho$, assuming that all G_i have the same rate $\rho = 2^{k'_i}/m_i$. This, in turn implies that

$$|S_i| = \left\lceil -\frac{\lambda + \log(\ell)}{\log((1+\rho)/2)} \right\rceil$$

suffices to ensure that the error probability is smaller than $2^{-\lambda}$ for any ℓ. (This follows from (17) and the fact that $|\mathbf{F}| \gg 2^\lambda$, so all terms proportional to $1/|\mathbf{F}|$ may be ignored.) Finally, set $\ell = 2\log(N)/\log(\log(N))$ and $k'_i = \log(N)/\ell$ for each i, which results in, asymptotically, a communication complexity no larger than

$$\frac{2\log(N)}{\log(\log(N))}\big((\lambda + \log(N)^{1/2})t\log_2(|\mathbf{F}|) + C\log(N)\big) \quad \sim \quad 2C\frac{\log(N)^2}{\log(\log(N))},$$

as required, where $t = -1/\log((1+\rho)/2)$ for convenience. (It is not hard to show that this choice of k'_i and ℓ is optimal up to constants using the convex problem presented in appendix A as a lower bound for the total communication complexity, for given ℓ.) We note that, even at this optimal size of the k_i and ℓ, the dominant cost of communication comes from the Merkle proofs of the opening of the rows of the matrix X, rather than from the field elements in the provided rows. This likely points to a potential source for cost savings.

6.5 Extensions

In this subsection, we will discuss some simple extensions to the protocol that may be useful in practice.

Subfields for first round. A useful extension of the protocol is to allow the prover to commit to a matrix $\tilde{X}_1$ which lies in a subfield $\mathbf{F}' \subseteq \mathbf{F}$ of the main field. This makes the openings over the rows of X_1 smaller, which, in turn, decreases the communication complexity of the protocol for the first round. We use this flexibility in what follows.

Batching proofs. Another simple extension is that, if the prover would like to batch multiple Ligerito proofs, it simply runs the first round of the protocol for each proof in the batch and then can combine the two running instances of the protocol for all future rounds. We suspect that, while the proving time scales linearly in the size of the batch, the total communication will be substantially reduced, since the result of appendix A roughly states that each round will require (approximately) the same communication as any other.

6.6 Succinct computation for tensorizable codes

As noted before, the verifier needs to compute $\tilde{w}_\ell$ in order to complete the final check. The main problem is that, to compute $\tilde{w}_\ell$, the verifier needs to partially evaluate a polynomial which depends on $\tilde{w}_0$ (if provided) and $\tilde{w}_1$ which are large polynomials of size $2^{k_1+k'_1}$ (*i.e.*, the size of $\tilde{X}_1$) and 2^{k_1}, respectively. Unless $\tilde{w}_0$ is sparse, or itself tensorizable, Ligerito will not be succinct as it needs to at least read $\tilde{w}_0$ in order to complete the final check. On the other hand, if $\tilde{w}_0$ is, say, sparse, then the verifier can be succinct if it can succinctly partially evaluate $\tilde{w}_1$ over the first $k_1 - k_{\ell-1}$ variables (in order to compute $\tilde{w}_\ell$) and similarly for $\tilde{w}_i$ partially evaluated over the first $k_i - k_{\ell-1}$ variables. We will show that this is possible in the case that the G_i are tensorizable codes, which is a definition that some practical codes, such as Reed–Solomon codes and Reed–Muller codes, satisfy.

Tensorizable codes. A code $G \in \mathbf{F}^{m \times 2^k}$ (we will drop the indices for the remainder of the section) is said to be *tensorizable* if each row g_i^T of G can be written in the form

$$g_i = v_{i1} \otimes ... \otimes v_{ik},$$

where $v_{ij} \in \mathbf{F}^2$ for $j = 1, ..., k$ and $i = 1, ..., m$. (The dimensions can also vary for the v_{ij} but we choose length 2 for convenience, leaving the general case as a simple extension.) We will say a vector is *tensorizable* if it can be written in this form. (In other words, a code matrix is tensorizable if each row of the matrix is tensorizable.) One important example of a tensorizable code are the Binary Reed–Solomon codes, which can be written as

$$g_i = (1, B_1(t_i)) \otimes ... \otimes (1, B_k(t_i)),$$

where the $\{t_i\} \subseteq \mathbf{F}$ are evaluation points and the B_j are the subspace polynomials of [Lin+16].

Partial evaluation of tensorizable vectors. One easy observation is that, given two tensorizable vectors v and u of the same size 2^k, written

$$v = v_1 \otimes ... \otimes v_k,$$

and similarly for w, we can compute the inner product of $v^T w$ with $3k$ multiplications and k additions, by noting that

$$v^T u = (v_1 \otimes ... \otimes v_k)^T (u_1 \otimes ... \otimes u_k) = (v_1^T u_1)(v_2^T u_2)...(v_k^T u_k),$$

which follows from the mixed-product property of the Kronecker product. This is much faster than the naive $\sim 2^k$ operations required to compute the inner product of the two vectors. Note that the partial evaluation of v in the first $k' < k$ variables, when v is interpreted as a multilinear polynomial over k variables, can also be easily written as

$$\mathbf{Mat}(v)(r_1, ..., r_{k'}) = (\overline{r} \otimes I_2 \otimes ... \otimes I_2)^T v,$$

where I_2 is the 2×2 identity matrix. Since $\overline{r}$ itself is a tensorizable vector, this means that the right hand side can be computed quickly by noting that

$$(\overline{r} \otimes I_2 \otimes ... \otimes I_2)^T v = \Big((1 - r_1, r_1)^T v_1\Big)...\Big((1 - r_{k'}, r_{k'})^T v_{k'}\Big)(v_{k'+1} \otimes ... \otimes v_k),$$

which is just k' inner products, along with $k - k'$ Kronecker products. In our case, k', which is the number of remaining free variables at the ℓth round is very small (from the communication complexity section, $k - k' \sim \log(N)/\ell$), so evaluating this last tensor product is still succinct for the verifier, as required.

7 Numerics

In this section, we present some numerical results for the protocol. Our implementation is written in Julia, a high level language for scientific computing, and uses the `CryptoUtilities.jl` package for fast cryptographic operations in binary fields. This package includes a fast Reed–Solomon encoding library based on the algorithms described in [Lin+16]. All code for this work is available at

`https://github.com/bcc-research/Ligerito.jl`

In our numerical experiment, we choose the binary field $|\mathbf{F}| = 2^{128}$ as the main field and $\mathbf{F}' \subseteq \mathbf{F}$ with $|\mathbf{F}'| = 2^{32}$ as the subfield in the first round. As in the asymptotic section, we assume that all of the matrices G_i are Reed–Solomon code matrices, each with rate $\rho = 1/4$. (We choose this for simplicity, though the rates could differ for each matrix i here, with some potential gains.) We assume $\lambda = 100$ bits of security, making $|S_i| = 148$ (if $\ell \leq 8$, which is the case here). We choose ℓ by performing a grid search over the possible values of $\ell = 1, ..., 8$ and choose the $\{k_i\}$ and $\{k_i'\}$ by using a rounding scheme on the solution to a convex program, given ℓ, which we outline in appendix A. We run the prover on an M1 MacBook Pro with 64 GB of RAM and report our times (and proof sizes) below. Since the MacBook Pro has 8 performance cores (and 2 efficiency cores), we run the prover on 8 threads.

Coefficient count	Polynomial size	Proof size	Proving time
2^{20}	4 MiB	145 KiB	.08 s
2^{24}	64 MiB	255 KiB	1.3 s
2^{28}	1 GiB	360 KiB	21 s
2^{30}	4 GiB	420 KiB	80 s

Table 1: Proof size and time for various multilinear polynomials over $|\mathbf{F}'| = 2^{32}$ on 8 threads of an M1 MacBook Pro (64 GB) using Reed–Solomon codes.

Note that the proof sizes can differ from run to run since the Merkle batched opening size is not deterministic and depends on the exact rows opened. We also note that the scheme is

relatively memory efficient, with the prover using (asymptotically) a little over twice the size of the largest encoded matrix X_1 in the protocol.

Coefficient count	Encoded polynomial size	Prover allocated memory
2^{20}	16 MiB	49 MiB
2^{24}	256 MiB	630 MiB
2^{28}	4 GiB	9.8 GiB
2^{30}	16 GiB	31 GiB

Table 2: Total allocated memory for the prover, relative to the size of the largest matrix X_1 in the protocol, with rate $\rho = 1/4$.

We note that this is the total allocated memory for the prover during proving time, not the highest amount of memory used at any one time, which is likely much closer to the polynomial encoding size. This suggests that modern phones (which, almost universally, have at least 1 GiB of RAM) should be able to run the protocol for polynomials of size 2^{24} or so, even if the expected proving time is on the order of a few seconds. Appendix B also shows the theoretical proof size when using an RAA code instead of a Reed–Solomon code as the first matrix G_1. Due to the choice of field, in spite of the fact that RAA codes have much worse distance than Reed–Solomon codes, the proof sizes are only slightly larger than those of the Reed–Solomon codes.

8 Conclusion and future work

In this note, we presented a new polynomial commitment scheme and inner product protocol, Ligerito. We showed that it is a concretely small and fast protocol that is practical for even relatively large circuits, while also being asymptotically efficient. Given that large, practical circuits are currently on the order of around 2^{28} or so constraints, this sets basic groundwork for the ability to do 'real-time proving' of blockchains—*i.e.*, the ability to prove that the state transition function of a blockchain is correct within the allowed blocktime—or the ability to prove important, but smaller circuits (degree 2^{20} or so) on mobile devices such as phones.

Future work. There are a number of important and interesting open questions left by this note. For example, in Ligerito, the verifier performs a nontrivial amount of work (roughly on the order of around $N^{1-1/\ell}$ operations, with $\ell \leq 4$) since it has to construct the rows of the matrices G_i, which are large in the first iterations. One easy way of reducing this work is to further 'delegate' the row computation and inner product checks to the prover via, say a GKR-style approach used in the first round of [Bre+24], as opposed to the simple sumcheck construction used here, which is then 'glued' to the running sumcheck instance. We similarly suspect that a protocol such as the matrix multiplication protocol of [Tha13] could be used to further reduce the computational overhead of the prover for structured matrices such as the generator matrices of RAA codes.

9 Acknowledgements

We would like to thank Philip Jovanovic, Giacomo Fenzi, Remco Bloemen, Alex Evans, Kobi Gurkan, and Ron Rothblum for discussions, comments, and suggestions. We would specifically like to thank Nicolas Mohnblatt for carefully reading this paper and providing helpful comments and edits, many of which we have incorporated here.

Bibliography

[KZG10] A. Kate, G. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology–ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, 2010, pp. 177–194.

[Ame+17] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, Dallas Texas USA: ACM, Oct. 2017, pp. 2087–2104. doi: 10.1145/3133956.3134104.

[Ben+18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast Reed-Solomon Interactive Oracle Proofs of Proximity," pp. 1–17, 2018, doi: 10.4230/LIPICS.ICALP.2018.14.

[ZCF23] H. Zeilberger, B. Chen, and B. Fisch, "BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes." [Online]. Available: https://eprint.iacr.org/2023/1705

[Bre+24] M. Brehm, B. Chen, B. Fisch, N. Resch, R. D. Rothblum, and H. Zeilberger, "Blaze: Fast SNARKs from Interleaved RAA Codes." [Online]. Available: https://eprint.iacr.org/2024/1609

[Arn+24] G. Arnon, A. Chiesa, G. Fenzi, and E. Yogev, "WHIR: Reed–Solomon Proximity Testing with Super-Fast Verification." [Online]. Available: https://eprint.iacr.org/2024/1586

[EMA25] A. Evans, N. Mohnblatt, and G. Angeris, "ZODA: Zero-Overhead Data Availability." [Online]. Available: https://eprint.iacr.org/2025/034

[EA23] A. Evans and G. Angeris, "Succinct Proofs and Linear Algebra." [Online]. Available: https://eprint.iacr.org/2023/1478

[AER24] G. Angeris, A. Evans, and G. Roh, "A Note on Ligero and Logarithmic Randomness." [Online]. Available: https://eprint.iacr.org/2024/1399

[DG24] B. E. Diamond and A. Gruen, "Proximity Gaps in Interleaved Codes." [Online]. Available: https://eprint.iacr.org/2024/1351

[DP24] B. E. Diamond and J. Posen, "Proximity Testing with Logarithmic Randomness," *IACR Commun. Cryptol.*, vol. 1, no. 1, p. 2, 2024, doi: 10.62056/AKSDKP10.

[Ben+23] E. Ben-Sasson, D. Carmon, Y. Ishai, S. Kopparty, and S. Saraf, "Proximity Gaps for Reed–Solomon Codes," *Journal of the ACM*, p. 3614423, Aug. 2023, doi: 10.1145/3614423.

[Tha22] J. Thaler, "Proofs, Arguments, and Zero-Knowledge," *Foundations and Trends® in Privacy and Security*, vol. 4, no. 2–4, pp. 117–660, 2022, doi: 10.1561/3300000030.

[Fs24] M. Frigo and abhi shelat, "Anonymous credentials from ECDSA." [Online]. Available: https://eprint.iacr.org/2024/2010

[DP25] B. E. Diamond and J. Posen, "Succinct Arguments over Towers of Binary Fields," in *Advances in Cryptology – EUROCRYPT 2025*, S. Fehr and P.-A. Fouque, Eds., Cham: Springer Nature Switzerland, 2025, pp. 93–122.

[Lin+16] S.-J. Lin, T. Y. Al-Naffouri, Y. S. Han, and W.-H. Chung, "Novel Polynomial Basis With Fast Fourier Transform and Its Application to Reed–Solomon Erasure Codes," *IEEE Transactions on Information Theory*, vol. 62, no. 11, pp. 6284–6299, 2016, doi: 10.1109/TIT.2016.2608892.

[Tha13] J. Thaler, "Time-Optimal Interactive Proofs for Circuit Evaluation," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 71–89.

A Computing (approximately) optimal dimensions

We begin with the total communication complexity of the protocol, given in (19), ignoring the terms corresponding to the Merkle openings and the (comparatively small) terms proportional to k_i':

$$\log_2(|\mathbf{F}|)\left(\sum_{i=2}^{\ell-1}\left(|S_{i-1}|\ 2^{k'_{i-1}}\right) + |S_{\ell-1}|2^{k'_{\ell-1}} + 2^{k_{\ell-1}}\right).$$

Setting $n_i = 2^{k'_i}$ for $i = 1, ..., \ell-1$ and $n_\ell = 2^{k_{\ell-1}} = 2^{k'_\ell + k_\ell}$, and noting that $k_{i+1} + k'_{i+1} = k_i$ means that we can write the following optimization problem over the dimensions $n \in \boldsymbol{N}$:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{\ell} a_i n_i \\ \text{subject to} \quad & \prod_{i=1}^{n} n_i = N, \end{aligned}$$

and there is an implicit constraint that n_i must be a power of two. Here, $N = 2^{k_1 + k'_1}$ is the total number of coefficients in the polynomial and the coefficients a_i come from the communication complexity of the protocol:

$$a_i = \log_2(|\mathbf{F}|)\ |S_i|, \quad \text{for } i = 1, ..., \ell-1,$$

with $a_\ell = \log_2(|\mathbf{F}|)$. Unfortunately the above problem is hard to solve (we suspect this problem is NP-hard in general) since the dimensions n are constrained to be integral powers of two. If we relax this constraint and allow the n_i to take on values in the reals, the problem is easily turned into a convex problem (we will use this in an extension later). It is also very simple to solve and has solution:

$$n_i^\star = \frac{1}{a_i}\left(N\prod_{j=1}^{\ell} a_j\right)^{1/\ell}, \quad \text{for } i = 1, ..., \ell, \tag{20}$$

and optimal value

$$p^\star = \ell\left(N\prod_{i=1}^{\ell} a_i\right)^{1/\ell},$$

when $a_i > 0$ for all i and $N > 0$ (as is the case here). Note that, in general, the optimal solutions for the relaxed problem, $n_i^\star$, will not be integral, much less a power of two. We will later show a simple rounding scheme which will take a solution to the relaxed problem and round it to a feasible point that appears to work very well in practice.

Optimality. We can see this is the optimal value (and therefore the optimal choice of n_i) via the variable substitution

$$n_i = \frac{1}{a_i}\left(N\prod_{j=1}^{\ell} a_j\right)^{1/\ell} x_i, \tag{21}$$

which leads to the new problem over the variables $x \in \mathbf{R}^\ell$:

$$\begin{array}{ll} \text{minimize} & \left(N\prod_{j=1}^{\ell} a_j\right)\sum_{i=1}^{\ell} x_i \\ \text{subject to} & \prod_{i=1}^{n} x_i = 1. \end{array}$$

Note that the factor $N\prod_{j=1}^{\ell} a_j$ in the objective is a constant and will not change the optimal point $x^\star$. From the arithmetic-mean geometric-mean (AM-GM) inequality, for any feasible choice of x,

$$\frac{1}{\ell}\sum_{i=1}^{\ell} x_i \geq \left(\prod_{i=1}^{\ell} x_i\right)^{1/\ell} = 1,$$

where the second equality follows from the constraint $\prod_{i=1}^{\ell} x_i = 1$. Note that this inequality is tight at $x_i = 1$ for all $i = 1, ..., \ell$. Multiplying both sides by $\ell(N\prod_{j=1}^{\ell} a_j)^{1/\ell}$ gives a lower

bound for the objective value that is achieved when $x_i = 1$, or equivalently, when the n_i satisfy (20) via the substitution (21).

Rounding scheme. The rounding scheme is similarly very simple. First compute the optimal dimensions $n^\star$ for the relaxed problem (20) and then run the following procedure:

1. Round each $n_i^\star$ to the nearest power of two
2. If the product of the rounded $n^\star$ is not equal to N, then:
    1. If the product of the $n^\star$ is smaller than N, find any smallest $n_i^\star$ and double it
    2. If the product of the $n^\star$ is larger than N, find any largest $n_i^\star$ and halve it

This procedure is easily seen to terminate and, in practice, we find that the resulting dimensions are very good. Indeed, in our numerical experiments, the rounding procedure often terminates after no more than one iteration.

Extension. From the asymptotics of the protocol, the dominating term is, funnily enough, the Merkle proof size, which we ignored above. It is possible to give an extension of the optimization problem to include optimizing over the Merkle proof sizes. This new problem is similarly easily mapped to the following problem:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{\ell}(a_i n_i + b_i \log(n_i)) \\ \text{subject to} & \displaystyle\prod_{i=1}^{n} n_i = N, \end{array} \tag{22}$$

where the $b_i \geq 0$ are derived from the Merkle proof sizes and the a_i are the same as above. (Here we assume that log is the natural logarithm, for convenience.) This problem can be turned into an equivalent convex problem, so long as the parameters n are relaxed to be real-valued:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{\ell}(a_i n_i + b_i \log(n_i)) \\ \text{subject to} & \displaystyle\left(\prod_{i=1}^{n} n_i\right)^{1/\ell} \geq N^{1/\ell}. \end{array}$$

While this problem can be solved by a general convex solver over ℓ variables, we note that, using the variable substitution $n_i = \exp(x_i)$ and taking the dual yields a smooth, concave problem over one variable $\lambda \in \mathbf{R}$,

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{\ell} (\lambda - b_i)\left(1 - \log\left(\frac{\lambda - b_i}{a_i}\right)\right) + \lambda \log(N) \\ \text{subject to} \quad & \lambda \geq b_i, \quad \text{for each } i = 1, ..., \ell. \end{aligned}$$

The optimal $n^\star$ for the new problem (22) can be written in terms of the optimal $\lambda^\star$ for this dual problem,

$$n_i^\star = \frac{\lambda^\star - b_i}{a_i}, \quad \text{for } i = 1, ..., \ell.$$

This means that the problem can be solved by a simple bisection algorithm over the dual variable λ which, in turn, gives the optimal solution for the relaxed problem that can be rounded using the procedure outlined above. An informative exercise is to verify that the solution to this dual problem when $b = 0$ indeed matches the solution to the original problem (20) and can be written in closed form. (The general solution to the dual can be written as a complicated expression using Lambert W function family, but this is rather silly in practice compared to simply running bisection.) Finally, we note that, while asymptotically the Merkle proof sizes dominate, we suspect this happens only for very large values of N which are (for now) uncommon in practice. This suggests that the simpler solution and rounding scheme of (20) likely suffices for most practical scenarios.

B Repeat-Accumulate-Accumulate to Reed–Solomon codes

As suggested by Ron Rothblum, and similar in spirit to the Blaze protocol [Bre+24], another possibility is to use a repeat-accumulate-accumulate (RAA) code for the first code G_1 and Reed–Solomon codes for the remaining codes. In particular, G_1 will have the form

$$G_1 = AP_1AP_2(I \otimes \mathbf{1}_t),$$

where $t \geq 4$, $\mathbf{1}_t$ is the all-ones vector of size t, P_1 and P_2 are some permutation matrices, and A is the 'cumulative sum' matrix:

$$A = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ 1 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}.$$

Note that evaluating $G_1 v$ can clearly be done in linear time in the size of v (as it is simply some repetitions, permutations, and accumulations, each of which are linear time) and, most importantly, requires no multiplications, which are expensive to perform on consumer hardware. Depending on the application and rate, the relative distance of G_1 is no smaller than .19 or .29 for rate-1/4 or rate-1/8 codes, respectively [Bre+24]. This is somewhat smaller

than the relative distance of the previously-used Reed–Solomon codes with rate $\rho = 1/4$. This distance hit, which leads to more queries performed in the first round, is partially offset by the fact that smaller fields can be used for the first round, making each query smaller in size.

Parameters. Set the main field to the binary field $|\mathbf{F}| = 2^{128}$ and let $\mathbf{F}' \subseteq \mathbf{F}$ be the subfield of size $|\mathbf{F}'| = 2^8$. The set up is very similar to the previous for the remaining matrices. In partiular, the matrices $G_2, ..., G_{\ell-1}$ are rate-1/4 Reed–Solomon codes, while G_1 is a rate-1/4 repeat-accumulate-accumulate code with relative distance $\delta = .19$ described above. This means that $|S_1| = 1060$ suffices, while $|S_i| = 148$ suffices for $i \geq 2$, when the number of bits of security required is $\lambda = 100$.

Coefficient count	Polynomial size	Proof size
2^{20}	1 MiB	165 KiB
2^{24}	16 MiB	260 KiB
2^{28}	256 MiB	375 KiB
2^{30}	1 GiB	460 KiB

Table 3: Proof size for various multilinear polynomials over $|\mathbf{F}'| = 2^8$ using RAA and Reed–Solomon codes.

Notes. Unfortunately, so far as we could find, generating the rows of the RAA code G_1 was the dominating cost for both the prover and verifier, as we did not use the GKR-like construction in the first round of [Bre+24]. We suspect that this construction could be similarly applied, with very little added proof size, to the Ligerito protocol.