LAMP: Linear Verification of Matrix Multiplication via Proximity Testing

Kyeongtae Lee
Kookmin University
Seoul, Republic of Korea
sklee63kr@gmail.com

Byeongkyu Han
University of Illinois at Chicago
Chicago, IL, USA
bhan26@uic.edu

Jihye Kim
Kookmin University
Seoul, Republic of Korea
jihyek@kookmin.ac.kr

Hyunok Oh
Hanyang University
Seoul, Republic of Korea
hoh@hanyang.ac.kr

***Abstract*—Verifiable computation systems often need to prove large matrix multiplication statements, but a direct SNARK arithmetization of a $k \times k$ product requires $\mathcal{O}(k^3)$ constraints. Freivalds' randomized check reduces the algebraic computation to vector-matrix products, but proving those products inside a SNARK still costs $\mathcal{O}(k^2)$ constraints.**

We present LAMP, a matrix-multiplication checking protocol that combines Freivalds' randomized check with proximity testing over linear error-correcting codes. The prover commits to encoded matrices and intermediate vectors before the sampled query positions are derived. The CP-SNARK circuit then checks only the sampled codeword positions and commits to the values used inside the circuit, while Merkle openings and CP-Link proofs ensure consistency between the in-circuit witnesses and the externally committed values. We prove soundness for this committed-input setting under the soundness of the SNARK backend, the binding of the commitments, the correctness of the CP-Link checks, and the relative distance and unique-decoding radius of the Reed–Solomon code.

For a fixed number t of sampled positions, the main in-circuit SNARK relation has $\mathcal{O}(tk + E_{\mathsf{code}})$ constraints, with additional $\mathcal{O}(t \log n) + E_{\mathsf{link}}(k, t)$ backend work for Merkle openings and CP-Link checks. We implement LAMP in Go and compare it with a Freivalds-based SNARK circuit. In the matrix benchmark at $k = 2^{12}$, LAMP reduces the constraint count by $9.68\times$ and shortens proof generation time by $5.81\times$; verification remains approximately 0.13 s across the measured range.

***Index Terms*—Verifiable computation, SNARKs, Proximity testing, Linear codes, Freivalds' algorithm, Merkle trees, Commitment linking**

1. Introduction

Matrix multiplication is a core computational primitive underlying many large-scale computations, including numerical linear algebra, signal processing, computer graphics, and artificial intelligence. Consequently, many applications require verifiability for matrix products while keeping the underlying matrices private. Zero-knowledge proofs provide a cryptographic mechanism for this goal: a prover can convince a verifier that a statement is correct without revealing any secret information that witnesses the statement.

Applying general-purpose ZKP systems directly to large matrix multiplication, however, remains expensive. Succinct proof systems such as [1]–[4] prove relations after the computation has been represented as an arithmetic circuit or a similar algebraic constraint system. A standard arithmetization of a $k \times k$ matrix product $\mathbf{AB} = \mathbf{C}$ computes every output entry and therefore uses $\mathcal{O}(k^3)$ arithmetic constraints. This is already impractical for moderate k, and it is especially problematic for Transformer-based neural networks [5]–[7], whose projection and attention layers often contain matrix multiplications with dimension $k \geq 1{,}000$.

Quadratic circuit cost of Freivalds. Freivalds' classical randomized check [8] reduces matrix-product verification from a cubic computation to a quadratic one. Given matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and a random vector $\mathbf{r}$, the verifier checks

$$\mathbf{rAB} = \mathbf{rC}$$

instead of recomputing the full product. This reduces the verification cost from $\mathcal{O}(k^3)$ to $\mathcal{O}(k^2)$. The resulting $\mathcal{O}(k^2)$-size circuit is an improvement, but it is still expensive: for example, at $k = 2^{12}$, it already requires 50.5 million R1CS constraints and 411.28 s of proving time with Groth16 in our measured baseline. This motivates reducing the circuit cost of the Freivalds-in-SNARK approach.

Our insight: from computation to checking. In the Freivalds-in-SNARK baseline, the quadratic circuit cost arises from computing the vector–matrix products inside the circuit. LAMP reduces this in-circuit cost by checking sampled relations over encoded data, while retaining native prover work on the matrices. The prover performs the matrix arithmetic outside the SNARK and commits to the relevant encoded data and intermediate values. The verifier then checks only a small number of lightweight algebraic relations that are sufficient to detect inconsistent claims. The technical core of this approach is proximity testing over linear error-correcting codes. Let $\mathcal{C}$ be a linear code with

Jihye Kim and Hyunok Oh are co-corresponding authors.

relative distance δ. We encode each matrix row-wise and consider the Freivalds intermediate vectors

$$\mathbf{x} = \mathbf{r}\mathbf{A}, \qquad \mathbf{y} = \mathbf{x}\mathbf{B}, \qquad \mathbf{z} = \mathbf{r}\mathbf{C}.$$

The linearity of $\mathcal{C}$ lets us check these vector–matrix products in the encoded domain, column by column. For one encoded column, a single inner product, or *fold*, checks one coordinate of the encoded product. If the prover cheats in one of the corresponding relations, the two relevant codewords disagree on at least a δ-fraction of positions. A random query therefore detects the inconsistency with probability at least δ.

The key point is that the SNARK circuit never takes the full matrices as witnesses. Instead, it checks fold equations on t sampled encoded columns, where t is independent of the matrix dimension k. It also checks that the committed encoded views of the intermediate vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are valid codewords for those vectors. To bind the circuit witnesses to the data fixed before the query positions are known, we use a commit-and-prove SNARK for the sampled block witnesses. The prover also supplies Merkle membership proofs for the queried leaves under rt_{ABC} and rt_{XYZ}, together with CP-Link proofs showing that the SNARK-side sampled blocks and the opened Pedersen leaves contain the same data. These checks ensure, under the backend assumptions, that the values checked inside the circuit are consistent with commitments fixed before the verifier's sampling challenges.

As a result, the main SNARK relation has $\mathcal{O}(tk+E_{\mathsf{code}})$ constraints, where E_{code} is the cost of checking that the intermediate vectors are valid encodings under the error-correcting code. For the fixed-rate Reed–Solomon setting used in our implementation, this remains linear in the matrix dimension for fixed t. The sampled opening and linking layer adds $\mathcal{O}(t \log n) + E_{\mathsf{link}}(k, t)$ backend work. In our implementation, verification consists of SNARK verification, CP-Link verification, and $\mathcal{O}(t \log n)$ hash operations for Merkle openings; for the QALink instantiation, CP-Link verification includes $\mathcal{O}(t)$ pairing terms batched into a multi-pairing check.

Contributions. We present LAMP, a protocol for efficiently proving matrix-multiplication statements. Our contributions are as follows.

- **A code-based matrix-product checking protocol.** LAMP combines Freivalds' randomized reduction with proximity testing over linear error-correcting codes. Instead of computing vector–matrix products inside the SNARK circuit, the protocol checks sampled fold equations over encoded commitments and verifies code consistency for the intermediate vectors.
  For fixed t, the main circuit cost is $\mathcal{O}(tk + E_{\mathsf{code}})$, which is linear in k for the fixed-rate Reed–Solomon setting used in our implementation.
- **Soundness analysis.** We prove soundness in the committed-input setting by leveraging the relative distance and unique-decoding radius of the Reed–Solomon code. The analysis also accounts for Merkle openings and CP-Link proofs, which bind the values checked inside the SNARK circuit to the data committed before the sampling challenges are derived.
- **Implementation and evaluation.** We implement LAMP in Go. At $k = 4096$, it uses $9.68\times$ fewer constraints and proves $5.81\times$ faster than the measured Freivalds SNARK baseline; it proves faster than DualMatrix at every shared dimension, by $3.62\times$ at $k = 8192$. Against an independent BN254 zkMatrix implementation, LAMP proves $4.00\times$ faster at $k = 2048$, while zkMatrix verifies faster and produces smaller proofs throughout the measured range.
  Shared-layout batching keeps LAMP's Merkle proof size and verification nearly constant, but zkMatrix proves faster for the measured batches of one to ten independent 128×128 products. On GPT-2, LAMP proves faster than the Freivalds SNARK baseline and DualMatrix, while heterogeneous commitment layouts increase its proof size and verification cost.

2. Related Work

Matrix multiplication. Pinocchio [9] and Groth16 [1] provide constant-size proofs and constant verifier time, but directly applying them to matrix multiplication requires arithmetizing the entire computation. Thus, verifying the multiplication of two $k \times k$ matrices requires $\mathcal{O}(k^3)$ constraints, resulting in $\mathcal{O}(k^3 \log k)$ prover computation.

To overcome this limitation, several proof systems have studied how to verify matrix multiplication more efficiently without directly arithmetizing the full cubic computation. Thaler's sum-check-based protocol [10] reduces matrix multiplication to checks over low-degree extensions, achieving $\mathcal{O}(k^2)$ prover work. However, the verifier computation still grows linearly with the matrix dimension, which can be costly for succinct verification of large matrices.

LegoSNARK [11] provides a framework for composing commit-and-prove proof systems, including a commit-and-prove variant of Groth16 and specialized proof gadgets. Its CPmm construction proves matrix multiplication with committed inputs and output using sum-check and polynomial evaluation proofs over multilinear extensions. For $k \times k$ matrices, it achieves $\mathcal{O}(k^2)$ prover work and $\mathcal{O}(\log k)$ verifier work and proof size, without arithmetizing matrix multiplication as an arithmetic circuit. This construction is distinct from proving a Freivalds circuit with a commit-and-prove variant of Groth16.

Other specialized matrix-multiplication arguments offer different trade-offs. zkMatrix [12] reduces matrix multiplication to four inner-product relations and adapts the techniques of Bulletproofs [13], achieving $\mathcal{O}(k^2)$ prover complexity and $\mathcal{O}(\log k)$ verifier complexity and proof size. DualMatrix [14], a follow-up work by the authors of zkMatrix, improves the prover complexity to $\mathcal{O}(K + k)$, where K denotes the number of non-zero entries. However, for dense matrices with $K = \Theta(k^2)$, its prover complexity remains $\mathcal{O}(k^2)$. zkMaP [15] employs KZG commitments to achieve

System	Prover Work	Circuit Constraints	Verifier Work	Proof
Groth16	$\mathcal{O}(k^3 \log k)$	$\mathcal{O}(k^3)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
LegoSNARK	$\mathcal{O}(k^2)$	–	$\mathcal{O}(\log k)$	$\mathcal{O}(\log k)$
zkMatrix	$\mathcal{O}(k^2)$	–	$\mathcal{O}(\log k)$	$\mathcal{O}(\log k)$
DualMatrix	$\mathcal{O}(K+k)$	–	$\mathcal{O}(\log k)$	$\mathcal{O}(\log k)$
zkMaP	$\mathcal{O}(k^2)$	–	$\mathcal{O}(1)$	$\mathcal{O}(1)$
LAMP	$\mathcal{O}(k^2)^\dagger$	$\mathcal{O}(tk)$	$\mathcal{O}(t \log k)$	$\mathcal{O}(t \log k)$

TABLE 1. ASYMPTOTIC COMPARISON WITH EXISTING SYSTEMS. LEGOSNARK REFERS TO CPMM FOR COMMITTED INPUTS AND OUTPUT, NOT A FREIVALDS CIRCUIT. "–" MEANS CIRCUIT CONSTRAINTS ARE NOT APPLICABLE. COSTS EXCLUDE COMPUTING $C = AB$ AND SETUP. †: LAMP'S PROVER ENTRY ASSUMES FIXED t AND $n = \Theta(k)$; ENCODING COST $E_{\mathsf{Enc}}(k, n)$ IS ADDITIONAL (SECTION 5). ITS CIRCUIT ENTRY USES $E_{\mathsf{code}} = \mathcal{O}(k)$ IN THIS SETTING.

$\mathcal{O}(k^2)$ prover work together with constant-size proofs and constant verifier time.

Composing a matrix-multiplication argument with proofs of other operations requires consistency of the shared witnesses. LegoSNARK explicitly supports such composition through compatible commitment interfaces and linking proofs. In LAMP, CP-Link binds the sampled circuit witnesses to the externally committed encoded data; the reduction in circuit size comes from checking sampled encoded-column relations.

Table 1 compares the computational complexities and proof characteristics of existing proof systems for matrix multiplication, including our approach.

Verifiable AI. Matrix multiplication is also a dominant cost in verifiable machine-learning systems. Systems such as vCNN [16], pvCNN [17], and zkVC [18] optimize verifiable inference for convolutional neural networks by reducing the number of multiplication constraints induced by QAP-based representations. Other systems use sum-check-style techniques. zkCNN [19] proposes a sum-check-based protocol for proving convolutions with linear prover time. In addition, zkGPT [20] and zkLLM [21] introduce protocols specialized for LLM architectures and use sum-check protocols to prove the correctness of linear layers efficiently. Meanwhile, Mystique [22] and zkML [23] use Freivalds' randomized checks.

Code-based proximity testing. LAMP also builds on the use of linear error-correcting codes and proximity tests in succinct proof systems. FRI [24] and related works [25], [26] construct proximity IOPs for Reed–Solomon codewords. Recent work further extends this viewpoint to more general foldable codes [27]. Our proximity layer is closer in spirit to the local codeword tests used in Ligero [28], Brakedown [29], and Blaze [30]. In particular, our use of in-circuit encoding checks is similar in spirit to Orion [31] and related work [32].

3. Preliminaries

We introduce the notation and building blocks used throughout the paper. Let $\mathbb{F}$ be a finite field of prime order p, let λ be the security parameter, and write $\mathsf{negl}(\lambda)$ and PPT for a negligible function and probabilistic polynomial time, respectively.

3.1. Notation

We use bold uppercase letters for matrices and bold lowercase letters for vectors, which are row vectors unless stated otherwise. For a matrix $\mathbf{M}$, $\mathbf{M}[i, j]$, $\mathbf{M}[i, *]$, and $\mathbf{M}[*, j]$ denote an entry, a row, and a column, respectively. We write $[n] = \{0, 1, \ldots, n-1\}$; $I \leftarrow\$ [n]^t$ denotes t independently sampled indices from $[n]$, and $r \leftarrow\$ \mathbb{F}$ denotes a uniform field element.

3.2. Linear Error-Correcting Codes

A linear $[n, k]$ code $\mathcal{C} \subseteq \mathbb{F}^n$ is the image of a linear encoder $\mathsf{Enc} : \mathbb{F}^k \to \mathbb{F}^n$. Its rate is $\rho = k/n$. Let d_H denote Hamming distance and $\Delta(\mathbf{u}, \mathbf{v}) = d_H(\mathbf{u}, \mathbf{v})/n$ relative Hamming distance. If $d_{\min}$ is the minimum Hamming distance of $\mathcal{C}$, then its relative distance and unique-decoding radius are $\delta = d_{\min}/n$ and $\tau_{\mathsf{UD}} = n^{-1}\lfloor (d_{\min} - 1)/2 \rfloor$, respectively. We use a proximity threshold $\tau \leq \tau_{\mathsf{UD}}$.

For $\mathbf{M} \in \mathbb{F}^{k \times k}$, its row-wise encoding $\widehat{\mathbf{M}} = \mathsf{Enc}(\mathbf{M}) \in \mathbb{F}^{k \times n}$ has rows $\mathsf{Enc}(\mathbf{M}[i, *])$ for $i \in [k]$.

Definition 3.1 (Interleaved code)**.** For $\ell \geq 1$, the ℓ-fold interleaved code $\mathcal{C}^{\langle \ell \rangle}$ groups ℓ codewords position-wise: its j-th symbol is $(\mathbf{c}^{(0)}[j], \ldots, \mathbf{c}^{(\ell-1)}[j]) \in \mathbb{F}^\ell$. Thus $\widehat{\mathbf{M}}$ is a k-fold interleaved codeword whose j-th symbol is $\widehat{\mathbf{M}}[*, j]$.

For $\mathbf{u} \in \mathbb{F}^k$, linearity gives

$$\mathsf{Enc}(\mathbf{u}\mathbf{M})[j] = \sum_{i \in [k]} u_i \widehat{\mathbf{M}}[i, j] =: \mathsf{Fold}(\mathbf{u}, \widehat{\mathbf{M}}[*, j]).$$

Our construction applies to linear codes satisfying the required proximity-gap property. The implementation uses an $[n, k]$ Reed–Solomon code with relative distance $\delta_{\mathsf{RS}} = (n - k + 1)/n$.

3.3. Commitment Schemes

A commitment scheme binds a committer to a message while optionally hiding it. We use Merkle trees for position binding and Pedersen commitments for computational binding, perfect hiding, and additive homomorphism.

Merkle trees.. For a collision-resistant hash function H, we write $\mathsf{rt} \leftarrow \mathsf{MT.Com}((v_j)_{j \in [n]}; \perp)$ for a Merkle commitment with deterministic leaves, $\pi_j \leftarrow \mathsf{MT.Open}((v_i)_{i \in [n]}, j)$ for an authentication path, and $\mathsf{MT.Verify}(\mathsf{rt}, j, v_j, \pi_j)$ for verification. Under collision resistance, a fixed root is position-binding: it cannot be opened to two different values at the same index except with negligible probability.

Pedersen commitments.. Let $\mathbb{G}$ be a group of order $|\mathbb{F}|$ and $\mathsf{ck} = (G_0, \ldots, G_{\ell-1}, H)$ a commitment key. For $\mathbf{v} \in \mathbb{F}^\ell$ and $o \leftarrow\!\$\, \mathbb{F}$, define

$$\mathsf{Ped.Com}(\mathsf{ck}, \mathbf{v}; o) = \sum_{i \in [\ell]} \mathbf{v}[i] G_i + oH.$$

Pedersen commitments are perfectly hiding, computationally binding under the discrete-logarithm assumption, and additively homomorphic.

3.4. Succinct Non-interactive Arguments of Knowledge

A SNARK (succinct non-interactive argument of knowledge) for an NP relation $\mathcal{R}$ is a tuple $\mathbf{\Pi} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$. The setup algorithm generates public parameters, the prover produces a proof for a statement–witness pair in $\mathcal{R}$, and the verifier decides whether to accept the proof. We use the standard notions of completeness, knowledge soundness, and zero knowledge, recalled formally in Appendix A.

3.4.1. Commit-and-Prove SNARKs. A commit-and-prove SNARK (CP-SNARK) [11], [33] proves relations over witnesses bound to an external *witness commitment*. Let $\mathcal{R}$ be a relation over $(\mathsf{x}, \mathsf{u}, \omega)$, where u is the committed witness and ω is the remaining private witness. The prover commits as $\widetilde{\mathsf{cm}} = \mathsf{Com}(\mathsf{ck}_S, \mathsf{u}; \widetilde{o})$ and proves the relation with respect to the extended statement $\mathsf{x}_{\mathsf{cp}} = (\mathsf{x}, \widetilde{\mathsf{cm}})$.

We write $\mathbf{\Pi}_{\mathsf{cp}} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Prove}, \mathsf{Verify})$. The algorithms are defined as follows.

- $\mathsf{pp} \leftarrow \mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Setup}(1^\lambda, \mathcal{R})$ generates $\mathsf{pp} = (\mathsf{pk}, \mathsf{vk}, \mathsf{ck}_S)$, consisting of the proving, verification, and witness-commitment keys.
- $\widetilde{\mathsf{cm}} \leftarrow \mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Com}(\mathsf{pp}, \mathsf{u}; \widetilde{o})$ commits to u using opening randomness $\widetilde{o}$.
- $\pi_{\mathsf{cp}} \leftarrow \mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Prove}(\mathsf{pp}, \mathsf{x}_{\mathsf{cp}}, \mathsf{w})$, where $\mathsf{w} = (\mathsf{u}, \omega)$, proves that $(\mathsf{x}, \mathsf{u}, \omega) \in \mathcal{R}$ relative to $\widetilde{\mathsf{cm}}$.
- $b \leftarrow \mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Verify}(\mathsf{pp}, \mathsf{x}_{\mathsf{cp}}, \pi_{\mathsf{cp}})$ returns $b = 1$ if the proof is accepted and $b = 0$ otherwise. Equivalently, we may write this as $\mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Verify}(\mathsf{pp}, \mathsf{x}, \widetilde{\mathsf{cm}}, \pi_{\mathsf{cp}})$.

When clear from context, we abbreviate x_{cp} as x.

3.5. CP-Link

A CP-Link proof links commitments created under different commitment keys. In LAMP, it shows that a SNARK-side commitment to an ordered concatenation of sampled blocks is consistent with external commitments to the individual blocks, without revealing the blocks or their opening randomness.

Let Com be a commitment scheme. For a SNARK-side key ck_S, an external key ck_E, a SNARK-side commitment $\widetilde{\mathsf{cm}}$, and external commitments $\mathsf{cm}_0, \ldots, \mathsf{cm}_{q-1}$, define

$$\mathsf{x}_{\mathsf{link}} = ((\mathsf{ck}_S, \widetilde{\mathsf{cm}}), (\mathsf{ck}_E, \mathsf{cm}_0, \ldots, \mathsf{cm}_{q-1})),$$
$$\mathsf{w}_{\mathsf{link}} = ((\mathbf{m}_i)_{i=0}^{q-1}, \widetilde{o}, (o_i)_{i=0}^{q-1}).$$

The pair $(\mathsf{x}_{\mathsf{link}}, \mathsf{w}_{\mathsf{link}})$ belongs to $\mathcal{R}_{\mathsf{link}}$ if and only if

$$\widetilde{\mathsf{cm}} = \mathsf{Com}(\mathsf{ck}_S, \mathbf{m}_0 \| \cdots \| \mathbf{m}_{q-1}; \widetilde{o}),$$
$$\mathsf{cm}_i = \mathsf{Com}(\mathsf{ck}_E, \mathbf{m}_i; o_i) \quad \text{for all } i \in [q].$$

Thus, an accepting proof links each externally committed block to its corresponding position in the SNARK-side commitment.

We write $\mathbf{\Pi}_{\mathsf{link}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$. The algorithms are defined as follows.

- $\mathsf{pp}_{\mathsf{link}} \leftarrow \mathbf{\Pi}_{\mathsf{link}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{link}})$ generates public parameters for the linking relation.
- $\pi_{\mathsf{link}} \leftarrow \mathbf{\Pi}_{\mathsf{link}}.\mathsf{Prove}(\mathsf{pp}_{\mathsf{link}}, \mathsf{x}_{\mathsf{link}}, \mathsf{w}_{\mathsf{link}})$ produces a proof that the statement and witness satisfy the linking relation.
- $b \leftarrow \mathbf{\Pi}_{\mathsf{link}}.\mathsf{Verify}(\mathsf{pp}_{\mathsf{link}}, \mathsf{x}_{\mathsf{link}}, \pi_{\mathsf{link}})$ returns $b = 1$ if the proof is accepted and $b = 0$ otherwise.

The required CP-Link security properties are defined in Appendix B.

3.6. Freivalds' Algorithm

Freivalds' algorithm [8] is a classical randomized algorithm for verifying matrix multiplication. Given $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{k \times k}$, it samples $\mathbf{r} \leftarrow\!\$\, \mathbb{F}^k$ and checks whether $\mathbf{r}(\mathbf{AB}) = \mathbf{rC}$. The correctness relies on the following:

Lemma 3.2 (Schwartz–Zippel [34], [35])**.** *Let $P(X_1, \ldots, X_m)$ be a non-zero polynomial of total degree d over a finite field $\mathbb{F}$. Then for $r_1, \ldots, r_m \leftarrow\!\$\, \mathbb{F}$ chosen uniformly and independently,*

$$\Pr[P(r_1, \ldots, r_m) = 0] \le \frac{d}{|\mathbb{F}|}.$$

In our protocol, we use a structured variant: instead of sampling $\mathbf{r} \leftarrow\!\$\, \mathbb{F}^k$ uniformly, we sample one scalar $r \leftarrow\!\$\, \mathbb{F}$ and set $\mathbf{r} = (1, r, r^2, \ldots, r^{k-1})$. This keeps the Freivalds challenge compact in the transcript and in the SNARK statement: the verifier records one field element rather than k independent ones. If $\mathbf{D} = \mathbf{AB} - \mathbf{C} \neq \mathbf{0}$, then for any non-zero column $\mathbf{d}_j$ of $\mathbf{D}$, the value $\langle \mathbf{r}, \mathbf{d}_j \rangle$ is a non-zero univariate polynomial in r of degree at most $k-1$. By Lemma 3.2, it vanishes with probability at most $(k-1)/|\mathbb{F}|$.

4. Overview of LAMP

This section gives a technical overview of LAMP before the formal construction in Section 5. The main message is that LAMP does not ask the SNARK circuit to compute a matrix product. Instead, the prover commits to encoded data, and the circuit checks only a small number of local consistency equations at randomly sampled codeword positions.

4.1. Problem Statement and Relation Reduction Chain

The starting point is the matrix multiplication relation

$$\mathbf{AB} = \mathbf{C}, \qquad \mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{k \times k}.$$

A direct SNARK arithmetization of this relation computes every entry of $\mathbf{AB}$. This requires k^3 scalar multiplications and therefore $\mathcal{O}(k^3)$ arithmetic constraints. This is the original relation:

$$\mathcal{R}_{\mathsf{orig}} = \{(\mathbf{A}, \mathbf{B}, \mathbf{C}) : \mathbf{AB} = \mathbf{C}\}.$$

Freivalds' algorithm reduces the cubic check to a vector-matrix check. The classical algorithm samples a challenge vector $\mathbf{r} \in \mathbb{F}^k$ and checks $\mathbf{rAB} = \mathbf{rC}$.

In LAMP, the verifier derives a scalar $r \in \mathbb{F}$ and sets $\mathbf{r} = (1, r, \ldots, r^{k-1})$, which keeps the public challenge compact. At the same transcript point, the verifier derives an independent scalar $s \in \mathbb{F}$ and sets $\mathbf{s} = (1, s, \ldots, s^{k-1})$. The role of this additional challenge in testing the proximity of the committed encoding of $\mathbf{B}$ is described in Section 4.2. Equivalently, the prover introduces intermediate vectors

$$\mathbf{x} = \mathbf{rA}, \qquad \mathbf{y} = \mathbf{xB}, \qquad \mathbf{z} = \mathbf{rC},$$

and the verifier checks $\mathbf{y} = \mathbf{z}$. This gives the Freivalds relation

$$\mathcal{R}_{\mathsf{Fr}} = \Big\{(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{x}, \mathbf{y}, \mathbf{z}) : \mathbf{x} = \mathbf{rA}, \quad \mathbf{y} = \mathbf{xB}, \quad \mathbf{z} = \mathbf{rC}, \quad \mathbf{y} = \mathbf{z}\Big\}.$$

This removes the cubic matrix-matrix computation, but a SNARK circuit that computes $\mathbf{rA}$, $\mathbf{xB}$, and $\mathbf{rC}$ directly still performs $\mathcal{O}(k^2)$ scalar multiplications.

LAMP applies one more reduction. Rather than computing the full vector-matrix products inside the circuit, it checks the products only at randomly sampled positions in an encoded domain. The prover performs the expensive native computation outside the SNARK, commits to the encoded matrices and intermediate vectors, and the SNARK checks local equations of the form

$$\mathsf{Enc}(\mathbf{x})[j] = \mathsf{Fold}(\mathbf{r}, \widehat{\mathbf{A}}[*, j]),$$

$$\mathsf{Enc}(\mathbf{y})[j] = \mathsf{Fold}(\mathbf{x}, \widehat{\mathbf{B}}[*, j]),$$

$$\mathsf{Enc}(\mathbf{z})[j] = \mathsf{Fold}(\mathbf{r}, \widehat{\mathbf{C}}[*, j]).$$

The prover additionally computes $\mathbf{b} = \mathbf{sB}$, and the circuit checks

$$\mathsf{Enc}(\mathbf{b})[j] = \mathsf{Fold}(\mathbf{s}, \widehat{\mathbf{B}}[*, j]).$$

An additional fold based on the independent challenge $\mathbf{s}$ is used to test the proximity of the committed encoding of $\mathbf{B}$, as described in Section 4.2. Only the queried columns $\widehat{\mathbf{A}}[*, j]$, $\widehat{\mathbf{B}}[*, j]$, $\widehat{\mathbf{C}}[*, j]$ are used by the circuit. Thus, for t queried positions, the fold part of the circuit has cost $\mathcal{O}(tk)$, which is linear in k for fixed t.

4.2. Encoding and Proximity Testing

We use the proximity threshold $\tau \leq \tau_{\mathsf{UD}}$ defined in Section 3.2. For a committed row-wise encoding, proximity is measured with respect to the corresponding interleaved code $\mathcal{C}^{\langle k \rangle}$.

For a matrix $\mathbf{M} \in \mathbb{F}^{k \times k}$, the prover row-wise encodes $\mathbf{M}$ as $\widehat{\mathbf{M}} = \mathsf{Enc}(\mathbf{M}) \in \mathbb{F}^{k \times n}$. By Definition 3.1, this row-wise encoding can also be viewed as a k-fold interleaved codeword, where the j-th interleaved symbol is the encoded column $\widehat{\mathbf{M}}[*, j]$.

The key property underlying the local checks is the linearity of the code. For any $\mathbf{u} = (u_0, \ldots, u_{k-1}) \in \mathbb{F}^k$ and every $j \in [n]$,

$$\mathsf{Enc}(\mathbf{uM})[j] = \sum_{i=0}^{k-1} u_i \widehat{\mathbf{M}}[i, j] = \mathsf{Fold}\big(\mathbf{u}, \widehat{\mathbf{M}}[*, j]\big).$$

Thus, each coordinate of an encoded vector–matrix product can be checked using only the corresponding encoded column. Applying this identity to the Freivalds intermediates $\mathbf{x} = \mathbf{rA}$, $\mathbf{y} = \mathbf{xB}$, and $\mathbf{z} = \mathbf{rC}$ gives, for every $j \in [n]$,

$$\widehat{\mathbf{x}}[j] = \mathsf{Fold}(\mathbf{r}, \widehat{\mathbf{A}}[*, j]),$$

$$\widehat{\mathbf{y}}[j] = \mathsf{Fold}(\mathbf{x}, \widehat{\mathbf{B}}[*, j]),$$

$$\widehat{\mathbf{z}}[j] = \mathsf{Fold}(\mathbf{r}, \widehat{\mathbf{C}}[*, j]).$$

The second equation alone, however, does not provide a proximity test for the committed encoding $\widehat{\mathbf{B}}$. The folding vector $\mathbf{x} = \mathbf{rA}$ depends on $\mathbf{A}$ and can be degenerate. For example, if $\mathbf{A} = 0$, then $\mathbf{x} = 0$ for every challenge r, and the resulting $\mathbf{x}$-fold reveals no information about whether $\widehat{\mathbf{B}}$ is close to a valid interleaved codeword. To obtain an independent proximity test for $\widehat{\mathbf{B}}$, the prover computes $\mathbf{b} = \mathbf{sB}$. By the same linearity property, $\widehat{\mathbf{b}}[j] = \mathsf{Fold}(\mathbf{s}, \widehat{\mathbf{B}}[*, j])$ for every $j \in [n]$. Thus, the $\mathbf{r}$-fold for $\mathbf{A}, \mathbf{C}$ and the $\mathbf{s}$-fold for $\mathbf{B}$ test the proximity of $\widehat{\mathbf{A}}, \widehat{\mathbf{C}}$ and $\widehat{\mathbf{B}}$, whereas the $\mathbf{x}$-fold checks consistency of the claimed relation $\mathbf{y} = \mathbf{xB}$.

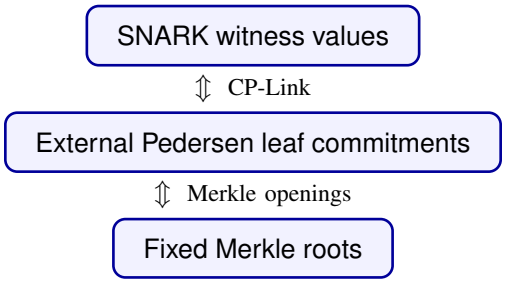
4.3. Sampled-Opening Layer

The sampled checks above are meaningful only if the values used inside the SNARK are bound to data fixed before the relevant challenges are sampled. LAMP achieves this through a sampled-opening layer. The prover first commits to the interleaved symbols, equivalently the encoded matrix columns, and fixes a Merkle root before the Freivalds challenge is derived. After the challenge is fixed, the prover computes the intermediate vectors, commits to their encoded symbols, and fixes a second Merkle root before the sampled positions are derived.

The SNARK circuit itself checks field equations: the sampled fold and equality checks, together with the code-consistency checks for the intermediate vectors. It does not verify Pedersen openings or Merkle paths internally. Instead, the commit-and-prove SNARK commits to the sampled witness blocks used by the circuit, and the prover supplies:

- Merkle openings showing that the external Pedersen commitments $\mathsf{cm}_{ABC,j}$ and $\mathsf{cm}_{XYZ,j}$ are authenticated under rt_{ABC} and rt_{XYZ};
- CP-Link proofs showing that the witness commitments from CP-SNARK and the external Pedersen commitments open to the corresponding sampled blocks.

Thus the verifier obtains a chain of consistency:



This separation is essential for efficiency. The SNARK checks only arithmetic relations on field elements, while Merkle authentication and commitment linking are verified outside the circuit.

4.4. Soundness Intuition

The soundness argument has three layers.

Freivalds soundness. If $\mathbf{AB} \neq \mathbf{C}$, then $\mathbf{D} = \mathbf{AB} - \mathbf{C}$ is nonzero. With the structured challenge $\mathbf{r} = (1, r, \ldots, r^{k-1})$, the vector $\mathbf{rD}$ is nonzero except with probability at most $(k-1)/|\mathbb{F}|$, by Schwartz–Zippel. Therefore, except with this probability, at least one of the vector relations in the Freivalds reduction is false.

Proximity soundness. The committed matrix encodings need not be exact codewords. If a committed encoding is τ-far from the interleaved code, the proximity-gap [36] property ensures, except with a small failure probability over the folding challenge, that its folded word remains τ-far from every valid codeword. Otherwise, if it is τ-close to an underlying codeword but the prover claims an inconsistent vector relation, the code distance δ guarantees that the folded word and the claimed encoding differ on at least a $(\delta - \tau)$-fraction of positions. Since $\tau \leq \tau_{\mathsf{UD}}$ implies $\delta - \tau \geq \tau$, an invalid claim leaves at least a τ-fraction of inconsistent positions in either case. Therefore, t random queries miss all inconsistencies with probability at most $(1-\tau)^t$. The proximity-gap failure over the folding challenges is accounted for separately in the analysis in Section 6.

Commitment and linking soundness. The distance argument applies only to values fixed before the query positions are sampled. Merkle position binding prevents the prover from opening a root to two different leaves at the same position. Pedersen binding prevents a committed leaf from being opened to two different messages. CP-Link ensures that the values used by the SNARK are the same values authenticated by the external commitments. Therefore, except for the binding and linking failure probabilities, any accepting proof corresponds to fixed encoded objects that pass the sampled proximity checks.

Putting the layers together, a cheating prover must either break one of the commitment/linking mechanisms, violate SNARK soundness, hit the Freivalds failure event, or avoid all sampled disagreements in the encoded domain. The formal theorem in Section 6 makes this union bound explicit.

5. The LAMP Protocol

Section 4 reduced the original matrix-multiplication relation to the Freivalds relation and then to sampled encoded checks. This section formalizes the relation checked by LAMP. The important point is that LAMP is not a single arithmetic-circuit relation: the arithmetic checks are proved by an inner CP-SNARK, while Merkle membership and CP-Link are verified outside the circuit.

5.1. The Composite LAMP Relation

Let $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{k \times k}$, and let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code with relative distance δ. The target matrix-multiplication relation is

$$\mathcal{R}_{\mathsf{orig}} = \{(\mathbf{A}, \mathbf{B}, \mathbf{C}) : \mathbf{AB} = \mathbf{C}\}.$$

The verifier does not receive the original matrices as public input. Instead, the prover commits to row-wise encodings of the matrices and proves that the committed objects satisfy the multiplication relation.

The relation checked by LAMP is the composite relation

$$\mathcal{R}_{\mathsf{LAMP}} = \mathcal{R}^{\mathsf{on}}_{\mathsf{LAMP}} \wedge \mathcal{R}^{\mathsf{off}}_{\mathsf{LAMP}}.$$

Here $\mathcal{R}^{\mathsf{on}}_{\mathsf{LAMP}}$ is the online relation proved by the CP-SNARK after the Freivalds challenge, query positions, and code-check points are sampled. The relation $\mathcal{R}^{\mathsf{off}}_{\mathsf{LAMP}}$ is the offline relation: it captures the values fixed by commitments before the online queries are known, and the verifier checks it through Merkle openings and CP-Link proofs. Sections 5.3 and 5.4 define these two relations explicitly. The circuit checks field equations on sampled encoded values, while the auxiliary verifier checks that those values are the ones bound by the pre-query commitments.

5.2. Commitment Layout and Public Challenges

The prover first encodes the matrices row-wise: $\widehat{\mathbf{A}} = \mathsf{Enc}(\mathbf{A}), \widehat{\mathbf{B}} = \mathsf{Enc}(\mathbf{B}), \widehat{\mathbf{C}} = \mathsf{Enc}(\mathbf{C})$. For each codeword position $j \in [n]$, it packs the three k-dimensional interleaved symbols, equivalently the three encoded columns, into $\mathbf{m}_{ABC,j} := (\widehat{\mathbf{A}}[*, j] \,\|\, \widehat{\mathbf{B}}[*, j] \,\|\, \widehat{\mathbf{C}}[*, j]) \in \mathbb{F}^{3k}$. Using a Pedersen commitment key ck_{ABC}, the prover samples $o_{ABC,j} \leftarrow\!\!\$\ \mathbb{F}$ and computes $\mathsf{cm}_{ABC,j} \leftarrow \mathsf{Ped.Com}(\mathsf{ck}_{ABC}, \mathbf{m}_{ABC,j}; o_{ABC,j})$. The matrix root is the Merkle commitment to these Pedersen commitments: $\mathsf{rt}_{ABC} \leftarrow \mathsf{MT.Com}((\mathsf{cm}_{ABC,j})_{j \in [n]}; \perp)$. Here and below, $\perp$ indicates that no additional randomness is used in the Merkle tree, since the leaves are Pedersen commitments that already provide hiding. The root rt_{ABC} is fixed before the Freivalds challenge is derived.

After receiving rt_{ABC}, the verifier samples $r, s \leftarrow\!\!\$\ \mathbb{F}$. The prover sets $\mathbf{r} = (1, r, r^2, \ldots, r^{k-1})$, $\mathbf{s} = (1, s, \ldots, s^{k-1})$, and computes $\mathbf{x} = \mathbf{rA}, \mathbf{y} = \mathbf{xB}, \mathbf{z} = \mathbf{rC}, \mathbf{b} = \mathbf{sB}$. It then

encodes the intermediate vectors: $\widehat{\mathbf{x}} = \mathsf{Enc}(\mathbf{x}), \widehat{\mathbf{y}} = \mathsf{Enc}(\mathbf{y}), \widehat{\mathbf{z}} = \mathsf{Enc}(\mathbf{z}), \widehat{\mathbf{b}} = \mathsf{Enc}(\mathbf{b})$. The encoded intermediate vectors form a 4-fold interleaved codeword. For each $j \in [n]$, the prover packs its j-th interleaved symbol into $\mathbf{m}_{XYZ,j} := (\widehat{\mathbf{x}}[j] \,\|\, \widehat{\mathbf{y}}[j] \,\|\, \widehat{\mathbf{z}}[j] \,\|\, \widehat{\mathbf{b}}[j]) \in \mathbb{F}^4$. Using a Pedersen commitment key ck_{XYZ}, the prover samples $o_{XYZ,j} \leftarrow\$ \mathbb{F}$, computes $\mathsf{cm}_{XYZ,j} \leftarrow \mathsf{Ped.Com}(\mathsf{ck}_{XYZ}, \mathbf{m}_{XYZ,j}; o_{XYZ,j})$, and sends the vector-symbol root $\mathsf{rt}_{XYZ} \leftarrow \mathsf{MT.Com}((\mathsf{cm}_{XYZ,j})_{j\in[n]}; \bot)$ to the verifier.

The prover also sends $\widetilde{\mathsf{cm}}_{\mathsf{int}} \leftarrow \mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Com}(\mathsf{pp}, \mathbf{m}_{\mathsf{int}}; \widetilde{o}_{\mathsf{int}})$, committing to the complete intermediate witness defined in Section 5.3. The verifier uses this same commitment in CP-SNARK verification.

Only after receiving rt_{XYZ} and $\widetilde{\mathsf{cm}}_{\mathsf{int}}$ does the verifier sample the query tuple $I = (j_1, \ldots, j_t) \leftarrow\$ [n]^t$. It also samples public code-check points $\alpha_x, \alpha_y, \alpha_z, \alpha_b \leftarrow\$ \mathbb{F} \setminus D$, where D is the code evaluation domain. These points are used by the circuit to check that $\widehat{\mathbf{x}}, \widehat{\mathbf{y}}, \widehat{\mathbf{z}}, \widehat{\mathbf{b}}$ are encodings of the claimed intermediate vectors. For the Reed–Solomon code used in our implementation, this check is the barycentric out-of-domain consistency test recalled in Appendix D.

5.3. The Online Relation

The online relation is the arithmetic relation proved inside the inner CP-SNARK. Its role is not to verify Merkle paths or Pedersen openings. Instead, it checks only the field equations that certify the sampled encoded positions are consistent with the Freivalds reduction.

The public statement contains the Merkle roots and verifier-sampled challenges:

$$\mathsf{x} = \big(\mathsf{rt}_{ABC}, \mathsf{rt}_{XYZ}, r, s, I, \alpha_x, \alpha_y, \alpha_z, \alpha_b\big).$$

The roots are included in the statement to bind the CP-SNARK proof to the same transcript and offline checks; the online field equations themselves do not evaluate Merkle paths.

The complete SNARK witness is $\mathsf{w} = (\mathsf{u}, \bot)$, where the committed part of the witness is

$$\mathsf{u} = \big((\mathbf{m}_{ABC,j})_{j\in I}, (\mathbf{m}_{XYZ,j})_{j\in I}, \mathbf{m}_{\mathsf{int}}\big),$$

where the committed intermediate witness is $\mathbf{m}_{\mathsf{int}} = \big(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{b}, \widehat{\mathbf{x}}, \widehat{\mathbf{y}}, \widehat{\mathbf{z}}, \widehat{\mathbf{b}}\big)$. The CP-SNARK treats u as the committed part of the witness. The prover first computes the SNARK-side witness commitments $\widetilde{\mathsf{cm}}_{ABC}$ and $\widetilde{\mathsf{cm}}_{XYZ}$ using $\mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Com}$, and then proves the online relation with witness $\mathsf{w} = (\mathsf{u}, \bot)$. These witness commitments are not checked against the Merkle roots inside the circuit. Instead, they are connected to the externally committed Merkle leaves by the offline relation in Section 5.4. For the CP-Link statements below, let $\mathsf{ck}_{\mathsf{cp},T}$ denote the public commitment key used by the CP-SNARK witness commitment $\widetilde{\mathsf{cm}}_T$, for $T \in \{ABC, XYZ\}$. Write $\mathsf{EncCheck}_{\mathcal{C}}(\widehat{\mathbf{v}}, \mathbf{v}; \alpha) = 1$ when the out-of-domain code-consistency check accepts that $\widehat{\mathbf{v}}$ is consistent with $\mathsf{Enc}(\mathbf{v})$ at the public point α. The online relation is defined in Figure 1.

The first four checks certify that $\widehat{\mathbf{x}}, \widehat{\mathbf{y}}, \widehat{\mathbf{z}}, \widehat{\mathbf{b}}$ are valid encodings of the claimed intermediate vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{b}$.

$\circ\ \mathcal{R}^{\mathsf{on}}_{\mathsf{LAMP}}(\mathsf{x}; \mathsf{w})$:
- parse
  - $\mathsf{x} = \big(\mathsf{rt}_{ABC}, \mathsf{rt}_{XYZ}, r, s, I, \alpha_x, \alpha_y, \alpha_z, \alpha_b\big)$
  - $\mathsf{w} = (\mathsf{u}, \bot)$
  - $\mathsf{u} = \big((\mathbf{m}_{ABC,j})_{j\in I}, (\mathbf{m}_{XYZ,j})_{j\in I}, \mathbf{m}_{\mathsf{int}}\big)$
  - $\mathbf{m}_{\mathsf{int}} = \big(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{b}, \widehat{\mathbf{x}}, \widehat{\mathbf{y}}, \widehat{\mathbf{z}}, \widehat{\mathbf{b}}\big)$
- $\mathbf{r} \leftarrow (1, r, \ldots, r^{k-1})$
- $\mathbf{s} \leftarrow (1, s, \ldots, s^{k-1})$
- $\mathsf{EncCheck}_{\mathcal{C}}(\widehat{\mathbf{x}}, \mathbf{x}; \alpha_x) = 1$
- $\mathsf{EncCheck}_{\mathcal{C}}(\widehat{\mathbf{y}}, \mathbf{y}; \alpha_y) = 1$
- $\mathsf{EncCheck}_{\mathcal{C}}(\widehat{\mathbf{z}}, \mathbf{z}; \alpha_z) = 1$
- $\mathsf{EncCheck}_{\mathcal{C}}(\widehat{\mathbf{b}}, \mathbf{b}; \alpha_b) = 1$
- $\forall j \in I$:
  - parse $\mathbf{m}_{ABC,j}$ as $\big(\widehat{\mathbf{A}}[*, j] \,\|\, \widehat{\mathbf{B}}[*, j] \,\|\, \widehat{\mathbf{C}}[*, j]\big)$
  - parse $\mathbf{m}_{XYZ,j}$ as $\big(\widehat{\mathbf{x}}[j] \,\|\, \widehat{\mathbf{y}}[j] \,\|\, \widehat{\mathbf{z}}[j] \,\|\, \widehat{\mathbf{b}}[j]\big)$
  - $\widehat{\mathbf{x}}[j] = \mathsf{Fold}(\mathbf{r}, \widehat{\mathbf{A}}[*, j])$
  - $\widehat{\mathbf{y}}[j] = \mathsf{Fold}(\mathbf{x}, \widehat{\mathbf{B}}[*, j])$
  - $\widehat{\mathbf{z}}[j] = \mathsf{Fold}(\mathbf{r}, \widehat{\mathbf{C}}[*, j])$
  - $\widehat{\mathbf{b}}[j] = \mathsf{Fold}(\mathbf{s}, \widehat{\mathbf{B}}[*, j])$
- $\mathbf{y} = \mathbf{z}$

Figure 1. The Online Relation

The remaining checks are local equations on the sampled codeword positions. For each queried position j, the circuit unpacks the sampled matrix-column block $\mathbf{m}_{ABC,j}$ and the sampled vector-symbol block $\mathbf{m}_{XYZ,j}$, and then verifies the four fold relations together with the final equality $\widehat{\mathbf{y}} = \widehat{\mathbf{z}}$.

Thus, the online relation only checks arithmetic consistency of the sampled values. It does not by itself guarantee that these sampled values came from the pre-query Merkle roots rt_{ABC} and rt_{XYZ}. That binding is provided by the offline relation in Section 5.4.

5.4. The Offline Relation

The offline relation connects the sampled witnesses used by the CP-SNARK to the commitments that were fixed before the query positions were sampled. The term "offline" indicates that these checks are performed outside the SNARK circuit. In our system, it also reflects that the committed data are fixed before the online sampled checks are carried out.

For each queried position $j \in I$, the verifier receives Merkle openings for the external Pedersen leaves $\mathsf{cm}_{ABC,j}$ and $\mathsf{cm}_{XYZ,j}$. The Merkle openings certify that these leaves are contained in the roots rt_{ABC} and rt_{XYZ}, respectively. In addition, the verifier checks CP-Link proofs connecting the CP-SNARK witness commitments $\widetilde{\mathsf{cm}}_{ABC}$ and $\widetilde{\mathsf{cm}}_{XYZ}$ to the corresponding external Pedersen leaves.

$$
\begin{aligned}
&\circ\ \mathcal{R}^{\mathsf{off}}_{\mathsf{LAMP}}\big(\ \mathsf{rt}_{ABC}, \mathsf{rt}_{XYZ}, I, \widetilde{\mathsf{cm}}_{ABC}, \widetilde{\mathsf{cm}}_{XYZ},\\
&\quad \pi^{ABC}_{\mathsf{link}}, \pi^{XYZ}_{\mathsf{link}},\\
&\quad (\mathsf{cm}_{ABC,j}, \pi^{ABC}_{\mathsf{mem},j})_{j\in I}, (\mathsf{cm}_{XYZ,j}, \pi^{XYZ}_{\mathsf{mem},j})_{j\in I}\big) :\\
&\quad \forall j \in I :\\
&\qquad \mathsf{MT.Verify}(\mathsf{rt}_{ABC}, j, \mathsf{cm}_{ABC,j}, \pi^{ABC}_{\mathsf{mem},j}) = 1\\
&\qquad \mathsf{MT.Verify}(\mathsf{rt}_{XYZ}, j, \mathsf{cm}_{XYZ,j}, \pi^{XYZ}_{\mathsf{mem},j}) = 1\\
&\quad \mathbf{\Pi}_{\mathsf{link}}.\mathsf{Verify}(\mathsf{pp}_{\mathsf{link}}, \mathsf{x}^{ABC}_{\mathsf{link}}, \pi^{ABC}_{\mathsf{link}}) = 1\\
&\quad \mathbf{\Pi}_{\mathsf{link}}.\mathsf{Verify}(\mathsf{pp}_{\mathsf{link}}, \mathsf{x}^{XYZ}_{\mathsf{link}}, \pi^{XYZ}_{\mathsf{link}}) = 1
\end{aligned}
$$

Figure 2. The Offline Relation

The CP-Link relation is interpreted blockwise. Each block $\mathbf{m}_{T,j}$ is the serialized interleaved symbol opened at position j for the corresponding committed object T. For $T \in \{ABC, XYZ\}$, define the ordered sampled message vector

$$\mathbf{M}_{T,I} \coloneqq \big(\mathbf{m}_{T,j_1} \,\|\, \mathbf{m}_{T,j_2} \,\|\, \cdots \,\|\, \mathbf{m}_{T,j_t}\big),\ I = (j_1, \ldots, j_t).$$

The SNARK-side witness commitment $\widetilde{\mathsf{cm}}_T$ commits to the concatenated sampled witness $\mathbf{M}_{T,I}$, while each external Pedersen leaf cm_{T,j_ℓ} commits to the corresponding block $\mathbf{m}_{T,j_\ell}$.

For $T \in \{ABC, XYZ\}$, let $\mathsf{ck}_{\mathsf{cp},T}$ denote the public commitment key used for the CP-SNARK witness commitment $\widetilde{\mathsf{cm}}_T$, and let ck_T denote the external Pedersen commitment key used for Merkle leaves of type T. Using this notation, Figure 2 writes the public CP-Link statement compactly as

$$\mathsf{x}^T_{\mathsf{link}} \coloneqq \big((\mathsf{ck}_{\mathsf{cp},T}, \widetilde{\mathsf{cm}}_T), (\mathsf{ck}_T, \mathsf{cm}_{T,j_\ell})_{\ell\in[t]}\big).$$

Therefore, the CP-Link proof for T proves the existence of openings such that $\widetilde{\mathsf{cm}}_T = \mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Com}(\mathsf{pp}, \mathbf{M}_{T,I}; \widetilde{o}_T)$, and, for every $\ell \in [t]$, $\mathsf{cm}_{T,j_\ell} = \mathsf{Ped.Com}\big(\mathsf{ck}_T, \mathbf{m}_{T,j_\ell}; o_{T,j_\ell}\big)$. This blockwise interpretation is important: the CP-Link proof claims that the ordered blocks committed inside the CP-SNARK are exactly the same as the ordered blocks committed by the corresponding external Pedersen leaves.

The offline relation is defined in Figure 2.

By combining the Merkle membership checks with the CP-Link checks, we obtain the consistency chain described in Section 4.3. For each $T \in \{ABC, XYZ\}$,

$$\begin{array}{ccc} \widetilde{\mathsf{cm}}_T & \overset{\pi^T_{\mathsf{link}}}{\longleftrightarrow} & (\mathsf{cm}_{T,j})_{j\in I} \\ (\mathsf{cm}_{T,j})_{j\in I} & \overset{(\pi^T_{\mathsf{mem},j})_{j\in I}}{\longleftrightarrow} & \mathsf{rt}_T \end{array}$$

Here $T = ABC$ uses the packed sampled matrix-column messages $(\mathbf{m}_{ABC,j})_{j\in I}$ and the root rt_{ABC}, while $T = XYZ$ uses the packed sampled vector-symbol messages $(\mathbf{m}_{XYZ,j})_{j\in I}$ and the root rt_{XYZ}.

The upper line states that the SNARK-side committed witness and the opened external Pedersen leaves contain the same ordered sampled blocks. The lower line states that those external leaves are authenticated under the corresponding pre-query Merkle root. Therefore, once both $\mathcal{R}^{\mathsf{on}}_{\mathsf{LAMP}}$ and $\mathcal{R}^{\mathsf{off}}_{\mathsf{LAMP}}$ hold, the verifier knows that the arithmetic checks performed inside the CP-SNARK were applied to values already bound by rt_{ABC} and rt_{XYZ}. This separation is what allows the SNARK circuit to avoid in-circuit Pedersen verification and in-circuit Merkle path verification.

5.5. Protocol Algorithms and Complexity

Figure 3 presents the public-coin version of LAMP. The protocol can be made non-interactive via the Fiat–Shamir transform, by deriving r, s after rt_{ABC} and deriving $I, \alpha_x, \alpha_y, \alpha_z, \alpha_b$ after rt_{XYZ} and $\widetilde{\mathsf{cm}}_{\mathsf{int}}$.

The proof consists of the CP-SNARK proof, the sampled leaf commitments and Merkle openings, and the CP-Link proofs. The sampled matrix-column blocks and vector-symbol blocks are private CP-SNARK witnesses; they are not serialized as public proof material. The public statement contains the roots $\mathsf{rt}_{ABC}, \mathsf{rt}_{XYZ}$ and the verifier-sampled challenges.

For every $T \in \mathcal{T} = \{ABC, XYZ\}$, Figure 3 uses the following CP-Link statement and witness:

$$\begin{aligned} \mathsf{x}^T_{\mathsf{link}} &\coloneqq \big((\mathsf{ck}_{\mathsf{cp},T}, \widetilde{\mathsf{cm}}_T), (\mathsf{ck}_T, \mathsf{cm}_{T,j})_{j\in I}\big),\\ \mathsf{w}^T_{\mathsf{link}} &\coloneqq \big((\mathbf{m}_{T,j})_{j\in I}, \widetilde{o}_T, (o_{T,j})_{j\in I}\big). \end{aligned}$$

Thus the CP-Link API is used with the public linking parameters, the commitment keys and commitments as the statement, and the sampled messages together with their opening randomness as the witness.

Cost model. We report the cost of LAMP. The computation of the claimed output matrix $C = AB$ is outside this cost model. We do count the work performed by the LAMP prover after the matrices are available, including encoding, commitment generation, computation of the Freivalds intermediate witnesses, sampled Merkle openings, CP-SNARK proving, and CP-Link proving.

Let $E_{\mathsf{Enc}}(k, n)$ denote the cost of the corresponding encoding phase. Let $E_{\mathsf{code}} = E_{\mathsf{code}}(k, n)$ denote the constraint cost of the intermediate code-consistency checks; for the Reed–Solomon check in Appendix D, this is $\mathcal{O}(k + n)$ after domain-dependent coefficients are precomputed. We write $\mathbb{G}$ for group operations used by Pedersen commitments, $\mathbb{F}$ for field operations, and $\mathbb{H}$ for hash evaluations. Let $E^{\mathcal{P}}_{\mathsf{snark}}$ and $E^{\mathcal{V}}_{\mathsf{snark}}$ denote the CP-SNARK proving and verification costs, respectively. Let $E^{\mathcal{P}}_{\mathsf{link}}(k, t)$ and $E^{\mathcal{V}}_{\mathsf{link}}(k, t)$ denote the prover and verifier costs, respectively, of the selected CP-Link construction. For the QALink instantiation in Appendix E, each of the two link statements has $q = t$ external commitments, so $E^{\mathcal{V}}_{\mathsf{link}}(k, t)$ includes $2(t+2)$ Miller-loop inputs, batched into one final exponentiation, plus $\mathcal{O}(t)$ group scalar multiplications for batching.

Table 2 summarizes the asymptotic costs at the level of the main protocol components.

Component	Prover	Verifier
ABC encoding	$E_{\mathsf{Enc}}(k,n)$	–
ABC commit	$\mathcal{O}(3nk)\,\mathbb{G} + \mathcal{O}(n)\,\mathbb{H}$	–
Intermediate	$\mathcal{O}(k^2)\,\mathbb{F}$	–
Intermediate encoding	$E_{\mathsf{Enc}}(k,n)$	–
Intermediate commit	$\mathcal{O}(n)\,\mathbb{G} + \mathcal{O}(n)\,\mathbb{H}$	–
Merkle openings	$\mathcal{O}(t \log n)\mathbb{H}$	$\mathcal{O}(t \log n)\mathbb{H}$
CP-SNARK	$\mathcal{O}(tk) + E_{\mathsf{code}}$ constraints $E^{\mathcal{P}}_{\mathsf{snark}}$	$E^{\mathcal{V}}_{\mathsf{snark}}$
CP-Link	$E^{\mathcal{P}}_{\mathsf{link}}(k,t)$	$E^{\mathcal{V}}_{\mathsf{link}}(k,t)$

TABLE 2. ASYMPTOTIC COSTS FOR LAMP.

Prover work. The prover first encodes the three input matrices and commits to the packed encoded matrix columns. The ABC commitment phase consists of n Pedersen commitments to blocks of length $3k$, followed by a Merkle root over the resulting leaves. After the Freivalds challenge r is sampled, the prover computes the intermediate witnesses $\mathbf{x} = \mathbf{rA}, \mathbf{y} = \mathbf{xB}, \mathbf{z} = \mathbf{rC}, \mathbf{b} = \mathbf{sB}$ which costs $\mathcal{O}(k^2)$ field operations. This is distinct from the excluded native computation of $C = AB$. The prover then encodes $\mathbf{x}, \mathbf{y}, \mathbf{z}$, commits to their encoded symbols, opens the sampled Merkle paths, proves the online CP-SNARK relation, and produces the CP-Link proofs.

For each queried position $j \in I$, the CP-SNARK circuit checks four length-k fold equations:

$$\mathsf{Fold}(\mathbf{r}, \widehat{\mathbf{A}}[*, j]), \mathsf{Fold}(\mathbf{x}, \widehat{\mathbf{B}}[*, j]),$$

$$\mathsf{Fold}(\mathbf{r}, \widehat{\mathbf{C}}[*, j]), \mathsf{Fold}(\mathbf{s}, \widehat{\mathbf{B}}[*, j]).$$

Thus the sampled arithmetic checks contribute $\mathcal{O}(tk)$ constraints.

The intermediate-vector code-consistency checks add E_{code} constraints. For the fixed-rate Reed–Solomon setting used in the experiments, $n = 2k$, so this extra cost is linear in k and the total online relation remains linear in k for fixed t.

Verifier work. The verifier does not encode matrices, compute intermediate vectors, or build commitment roots. Its work consists of CP-SNARK verification, verification of the sampled Merkle authentication paths, and CP-Link verification. Therefore, using the notation above, the verifier cost is $E^{\mathcal{V}}_{\mathsf{snark}} + \mathcal{O}(t \log n)\mathbb{H} + E^{\mathcal{V}}_{\mathsf{link}}(k,t)$, where the QALink verifier contributes $\mathcal{O}(t)$ pairing terms.

Comparison. A naive SNARK arithmetization of matrix multiplication costs $\mathcal{O}(k^3)$ constraints, while a Freivalds-based SNARK baseline computes the vector-matrix products inside the circuit and costs $\mathcal{O}(k^2)$ constraints. LAMP moves the vector-matrix products to native prover work and checks sampled encoded-column relations plus intermediate code-consistency equations inside the CP-SNARK. As a result, the main circuit constraint count is $\mathcal{O}(tk + E_{\mathsf{code}})$, which is linear in k for the fixed-rate code used in our implementation.

6. Security Analysis

We prove perfect completeness and computational soundness for the public-coin interactive LAMP scheme. The proof is stated for the protocol in which the verifier samples the Freivalds challenge, the query positions, and the code-check points.

Statement and scope. Section 5 defines the checked relation as the composite relation

$$\mathcal{R}_{\mathsf{LAMP}} = \mathcal{R}^{\mathsf{on}}_{\mathsf{LAMP}} \wedge \mathcal{R}^{\mathsf{off}}_{\mathsf{LAMP}}.$$

The online component $\mathcal{R}^{\mathsf{on}}_{\mathsf{LAMP}}$ is enforced by the CP-SNARK, while the offline binding component $\mathcal{R}^{\mathsf{off}}_{\mathsf{LAMP}}$ is enforced by sampled Merkle membership checks and CP-Link proofs. This section proves that satisfying both components implies soundness for the committed matrix-multiplication statement.

The input root rt_{ABC}, fixed before the structured challenges, commits to three interleaved words

$$\widehat{\mathbf{A}}, \widehat{\mathbf{B}}, \widehat{\mathbf{C}} \in \mathbb{F}^{k \times n};$$

these words need not be codewords. Let $0 < \tau \leq \tau_{\mathsf{UD}}$, where

$$\tau_{\mathsf{UD}} = \frac{1}{n} \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$$

is the unique-decoding radius of the $[n, k]$ Reed–Solomon code. We call the committed instance false if some input word is more than τ-far from $\mathcal{C}^k$, or if the three words uniquely decode to matrices $\mathbf{A}^*, \mathbf{B}^*, \mathbf{C}^*$ for which $\mathbf{A}^*\mathbf{B}^* \neq \mathbf{C}^*$.

After rt_{ABC} is fixed, the verifier samples independent $r, s \leftarrow\!\$\, \mathbb{F}$ and sets

$$\mathbf{r} = (1, r, \ldots, r^{k-1}), \qquad \mathbf{s} = (1, s, \ldots, s^{k-1}).$$

Before the query tuple $I = (j_1, \ldots, j_t) \leftarrow\!\$\, [n]^t$ is sampled, the prover commits to $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{b}$ and their encoded views. For each $j \in I$, the circuit checks

$$\widehat{\mathbf{x}}[j] = \mathsf{Fold}(\mathbf{r}, \widehat{\mathbf{A}}[*, j]), \quad \widehat{\mathbf{y}}[j] = \mathsf{Fold}(\mathbf{x}, \widehat{\mathbf{B}}[*, j]),$$
$$\widehat{\mathbf{z}}[j] = \mathsf{Fold}(\mathbf{r}, \widehat{\mathbf{C}}[*, j]), \quad \widehat{\mathbf{b}}[j] = \mathsf{Fold}(\mathbf{s}, \widehat{\mathbf{B}}[*, j])$$

together with the full-vector equality $\mathbf{y} = \mathbf{z}$. In particular, the relation $\mathbf{b} = \mathbf{sB}$ is one of the four sampled relations; the equality $\mathbf{y} = \mathbf{z}$ is checked in full and incurs no sampling error.

Definition 6.1 (LAMP backend assumptions)**.** Let $\epsilon_{\mathsf{cp}}(\lambda)$, $\epsilon_{\mathsf{open}}(\lambda)$, and $\epsilon_{\mathsf{link}}(\lambda)$ denote, respectively, the soundness errors of the CP-SNARK, the Merkle and Pedersen openings, and CP-Link. Let $\epsilon_{\mathsf{code}}(\lambda)$ denote the soundness error of an encoding-consistency check for $\widehat{\mathbf{x}}, \widehat{\mathbf{y}}, \widehat{\mathbf{z}}$, and $\widehat{\mathbf{b}}$, where the checked values are fixed before the corresponding check point is sampled. For the barycentric Reed–Solomon check,

$$\epsilon_{\mathsf{code}} \leq \frac{n-1}{|\mathbb{F}| - n}.$$

Theorem 6.2 (Completeness, soundness, and zero knowledge)**.** *Let $\mathcal{C}$ be the $[n, k]$ Reed–Solomon code with relative*

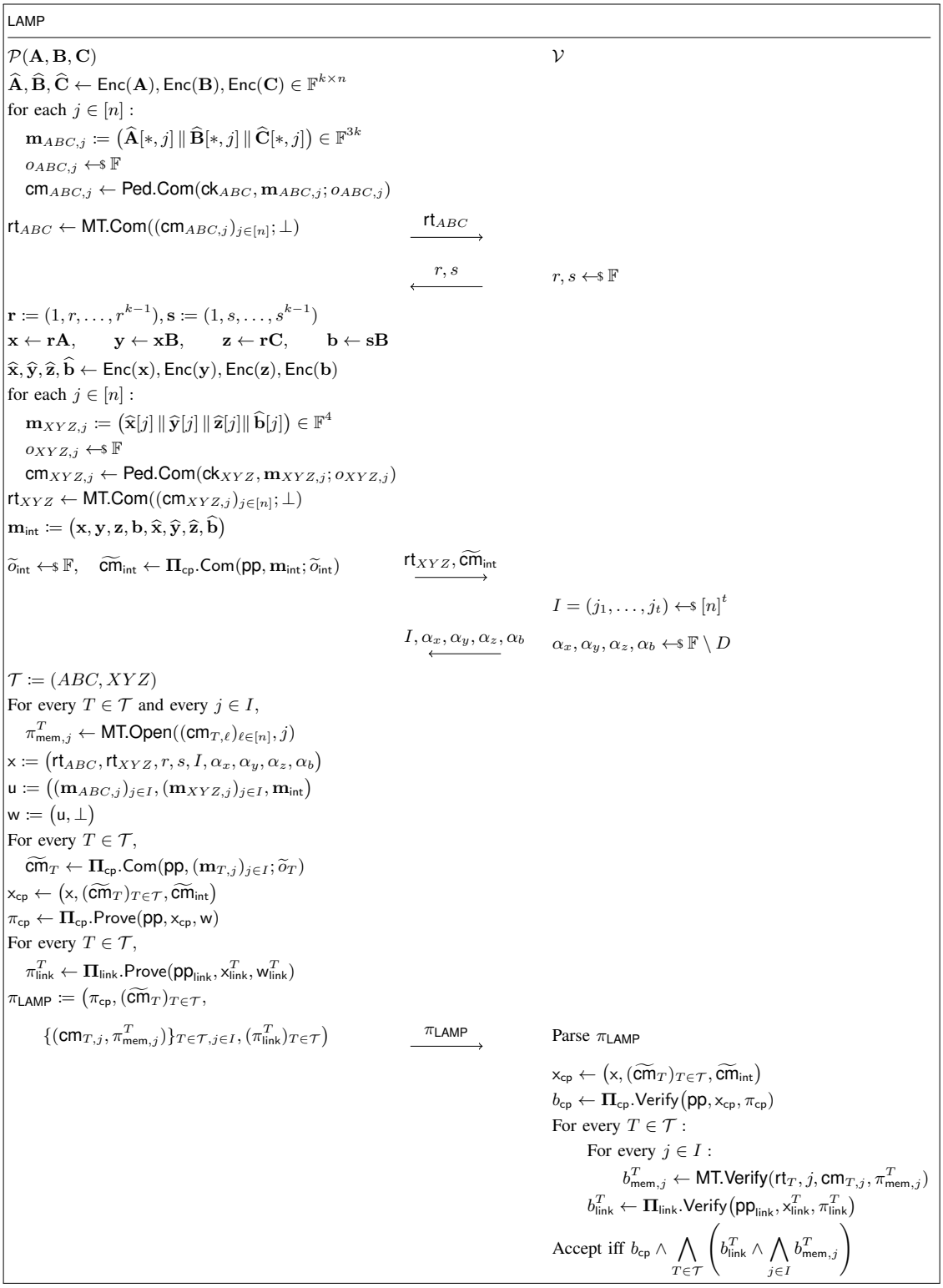


Figure 3. The LAMP protocol

distance $\delta_{\mathsf{RS}} = (n-k+1)/n$, and assume Definition 6.1. Then the public-coin interactive LAMP scheme satisfies:

(i) ***Perfect completeness.*** *Honest encodings of matrices satisfying $\mathbf{AB} = \mathbf{C}$ are accepted with probability one.*

(ii) ***Interactive computational soundness.*** *For every PPT prover and every false committed instance fixed before r, s,*

$$\begin{aligned}\Pr[\mathsf{Acc}] \le \epsilon_{\mathsf{cp}}(\lambda) + \epsilon_{\mathsf{open}}(\lambda) + \epsilon_{\mathsf{link}}(\lambda) + \epsilon_{\mathsf{code}}(\lambda) \\ + \frac{(k-1)n}{|\mathbb{F}|} + \frac{k-1}{|\mathbb{F}|} \\ + \max\{(1-\tau)^t, (1-\delta_{\mathsf{RS}}+\tau)^t\}.\end{aligned}$$

(iii) ***Zero knowledge.*** *If the CP-SNARK is zero knowledge, CP-Link is witness zero knowledge, and Pedersen commitments are hiding, the verifier's view is computationally simulatable from the public transcript.*

Proof sketch. Set

$$\mathbf{x} = \mathbf{rA}, \qquad \mathbf{y} = \mathbf{xB}, \qquad \mathbf{z} = \mathbf{rC}, \qquad \mathbf{b} = \mathbf{sB}.$$

When $\mathbf{AB} = \mathbf{C}$, we have $\mathbf{y} = \mathbf{z}$; the four fold equations follow from the linearity of the code. This proves completeness.

For soundness, condition on the correctness of the four backend checks. If an input word is τ-far from $\mathcal{C}^k$, choose the first such word in the order $\widehat{\mathbf{A}}, \widehat{\mathbf{B}}, \widehat{\mathbf{C}}$. This choice is determined by rt_{ABC} and hence precedes r, s. Apply the structured-fold proximity bound to its $\mathbf{r}$-fold when the chosen word is $\widehat{\mathbf{A}}$ or $\widehat{\mathbf{C}}$, and to its $\mathbf{s}$-fold when it is $\widehat{\mathbf{B}}$. The exceptional probability is at most $(k-1)n/|\mathbb{F}|$. Otherwise the chosen fold is more than τ-far from the claimed intermediate codeword, and all t queries avoid the disagreement set with probability at most $(1-\tau)^t$.

It remains to consider the case in which all three words uniquely decode to $\mathbf{A}^*, \mathbf{B}^*, \mathbf{C}^*$. If $\mathbf{A}^*\mathbf{B}^* \neq \mathbf{C}^*$, Schwartz–Zippel gives

$$\Pr_r[\mathbf{r}(\mathbf{A}^*\mathbf{B}^* - \mathbf{C}^*) = 0] \le \frac{k-1}{|\mathbb{F}|}.$$

Except for this event, $\mathbf{y} = \mathbf{z}$ is incompatible with all four fold relations. This includes $\mathbf{b} = \mathbf{sB}^*$, which tests the middle input independently of $\mathbf{x}$. Choose a false fold relation, in a fixed order, after the intermediate commitment and before I is sampled. Its correct folded word is τ-close to one codeword, whereas the prover's claimed word is a distinct codeword. The two tested words therefore differ in at least $(\delta_{\mathsf{RS}} - \tau)n$ positions, giving failure probability at most $(1 - \delta_{\mathsf{RS}} + \tau)^t$. Since the relation is fixed before the queries, no union bound over the four relations is needed. The equality $\mathbf{y} = \mathbf{z}$ is a full-vector check rather than an additional sampled relation.

Zero knowledge follows from the simulators for the CP-SNARK and CP-Link and the hiding of the Pedersen commitments. Appendix C gives the full argument. □

Remark 1 (Fiat–Shamir). The theorem above analyzes the public-coin interactive protocol. A non-interactive variant can derive r, s, I, and the code-check points from domain-separated transcript hashes via the Fiat–Shamir transform. Proving that variant adds the standard random oracle model loss and is orthogonal to the interactive soundness statement above.

7. Implementation and Evaluation

We implement LAMP and evaluate its performance. The implementation follows the construction in Section 5. We instantiate the CP-SNARK with Groth16, using the commit-and-prove interface [11], [33]. For CP-Link, we use a pairing-based QA-NIZK proof [37]; the concrete instantiation used in our implementation is given in Appendix E. Implementation and comparison code are maintained at https://github.com/lysias9049/LAMP-artifact.

We evaluate circuit scaling, component costs, batching independent products, and a GPT-2 workload. We also compare against an independent implementation of zkMatrix [12].

Experimental setup and comparison scope. The LAMP/Freivalds/DualMatrix experiments use BN254 on Linux with an EPYC 7B13, 32 cores, and 256GB memory, reporting ten-run averages. LAMP uses Reed–Solomon rate $\rho = 1/2$ and $t = 309$. The independent LAMP/zkMatrix comparison uses these LAMP parameters and BN254 for both systems on the same Linux VM: EPYC 7B13, 32 vCPUs, 256GiB provisioned memory, Go 1.26.2, and 32 threads. We measure $k \in \{128, 256, 512, 1024, 2048\}$ and $q = 1, \ldots, 10$ independent 128×128 products, ten times per configuration. Tables 4 and 5 report ten-run means for timings and byte counts.

The independent zkMatrix uses the optimized four-IPA construction, zero-knowledge masking, structured SRS, pairing accelerator, and Algorithm 6 batching. Adapting its published BLS12-381 instantiation to BN254 does not claim equal security or reproduce the authors' code. Both systems use commit `b34d228`; LAMP's circuit and challenge order are preserved.[1] Proving includes encoding, commitments, witness preparation, and online proofs; setup, compilation, and computing $\mathbf{C} = \mathbf{AB}$ are excluded. Serialization is outside the timings. Proof bytes use canonical compressed payloads including framing. Statement (B) is the byte size of the encoded public input to verification (commitments or roots, including framing), reported separately from the proof. Matrix entries and SRS/keys are excluded from these byte counts. All 300 proofs verify, including after payload decoding. The LAMP/Freivalds/DualMatrix and component tables use phase-based timing and different G1 byte accounting; their sizes are not directly comparable to the canonically compressed sizes reported for LAMP/zkMatrix.

1. The artifact uses evaluation-basis Reed–Solomon encoding and derives query indices from the intermediate Merkle root; the code-consistency challenge follows the Groth16 witness commitment. Our security analysis proves the public-coin protocol, not Fiat–Shamir security of this implementation transcript.

k	System	Constraints	Prove	Verify	Proof
2^7	LAMP	167,065	1.61 s	0.13 s	44,324 B
	Freivalds	49,610	0.59 s	2 ms	196 B
	DualMatrix	–	3.57 s	0.13 s	71,288 B
2^8	LAMP	329,378	2.90 s	0.13 s	52,196 B
	Freivalds	197,194	2.02 s	2 ms	196 B
	DualMatrix	–	4.81 s	0.14 s	77,140 B
2^9	LAMP	655,246	5.75 s	0.13 s	65,316 B
	Freivalds	788,810	6.99 s	2 ms	196 B
	DualMatrix	–	7.98 s	0.15 s	82,992 B
2^{10}	LAMP	1,306,973	12.28 s	0.13 s	79,972 B
	Freivalds	3,156,298	25.91 s	1 ms	196 B
	DualMatrix	–	18.54 s	0.15 s	88,844 B
2^{11}	LAMP	2,609,194	27.46 s	0.13 s	96,932 B
	Freivalds	12,622,154	102.32 s	2 ms	196 B
	DualMatrix	–	57.02 s	0.16 s	94,696 B
2^{12}	LAMP	5,214,878	70.73 s	0.13 s	116,004 B
	Freivalds	50,495,818	411.28 s	2 ms	196 B
	DualMatrix	–	205.21 s	0.17 s	100,548 B
2^{13}	LAMP	10,426,237	195.39 s	0.13 s	136,740 B
	Freivalds	–	–	–	–
	DualMatrix	–	707.43 s	0.18 s	106,400 B

TABLE 3. PERFORMANCE COMPARISON OF LAMP, THE FREIVALDS-IN-SNARK BASELINE, AND DUALMATRIX FOR SQUARE MATRIX MULTIPLICATION. "FREIVALDS" DENOTES THE GROTH16-BASED CIRCUIT BASELINE, NOT LEGOSNARK'S CPMM.

k	System	Prove (s)	Verify (ms)	Proof (B)	Statement (B)
128	LAMP	1.626	130.55	24,364.8	64
128	zkMatrix	0.655	9.28	4,096	112
256	LAMP	3.000	130.31	32,582.4	64
256	zkMatrix	2.193	9.96	4,544	112
512	LAMP	6.095	130.37	45,113.6	64
512	zkMatrix	7.979	10.86	4,992	112
1024	LAMP	12.726	130.88	61,254.4	64
1024	zkMatrix	29.513	11.91	5,440	112
2048	LAMP	28.310	131.08	78,585.6	64
2048	zkMatrix	113.180	12.69	5,888	112

TABLE 4. INDEPENDENT EPYC COMPARISON, INCLUDING COMMITMENT GENERATION. TIMES AND BYTE COUNTS ARE MEANS OVER TEN RUNS. STATEMENT IS THE ENCODED PUBLIC INPUT, REPORTED SEPARATELY FROM THE CANONICALLY COMPRESSED PROOF.

7.1. Comparison with Matrix-Multiplication Proof Systems

Table 3 compares LAMP with a Freivalds-in-SNARK baseline and the public DualMatrix implementation. The Freivalds baseline proves the vector–matrix checks inside an arithmetic circuit using our Groth16-based backend. It is distinct from LegoSNARK's specialized CPmm construction [11], for which we report no experimental measurements. The speedups over Freivalds below apply to this circuit baseline. DualMatrix was selected for code availability; it need not represent the fastest current system. It has no R1CS constraint count. Proving times include commitments and proof sizes include the corresponding commitment data.

LAMP's fixed overhead makes Freivalds faster at $k = 128, 256$; LAMP becomes faster from $k = 512$. At $k = 4096$, LAMP uses $9.68\times$ fewer constraints and proves $5.81\times$ faster. Freivalds runs out of memory at $k = 8192$, where LAMP uses approximately 49.32GB and proves in 195.39s. Freivalds produces 196B proofs and approximately 1–2ms verification. LAMP proves faster than DualMatrix at every shared dimension, by $3.62\times$ at $k = 8192$, with both verifying in under 0.2s. Under the byte accounting of Table 3, LAMP's proofs are smaller through $k = 1024$, but larger from $k = 2048$.

Independent zkMatrix comparison. In Table 4, zkMatrix proves faster at $k = 128, 256$, whereas LAMP proves faster at every measured dimension from 512 through 2048. At $k = 2048$, LAMP takes 28.310s versus 113.180s, a $4.00\times$ speedup. Conversely, zkMatrix verifies in 12.69ms with 5,888B proofs, versus 131.08ms and a mean of 78,585.6B for LAMP. zkMatrix verifies faster with smaller proofs throughout this grid. LAMP's verification includes CP-Link to its SNARK witnesses; an additional link from standalone zkMatrix to an external SNARK is not measured. These results concern the specified implementations and workloads, not all current systems.

zkMaP reproduction. We could not obtain runnable author code for zkMaP [15] despite requesting it. Reproducing the published equations under roots-of-unity, integer-node, and coefficient interpretations left a nonzero witness polynomial remainder for valid products. This concerns our reproduction and encoding choices, not every interpretation of the published protocol. A modified variant accepts honest inputs but changes the protocol and fails statement binding: our tests accept proofs after the claimed product is changed. We exclude its measurements from the performance comparison; they cannot benchmark a faithful, sound implementation of published zkMaP. Reproduction code, failure logs, and diagnostic measurements are retained. Table 1 describes the published theoretical properties.

7.2. LAMP Component Breakdown

Table 7 in Appendix F reports the component measurements at $\rho = 1/2$. Groth16 proof generation dominates for small matrices; at $k = 8192$, encoding and commitments take 126.49s, exceeding 68.89s for proof generation. Trusted setup is required for Groth16 and CP-Link, with Groth16 dominating setup cost. CP-Link accounts for approximately 0.12–0.13s of verification; Groth16 and Merkle verification take only a few milliseconds. Under this table's byte accounting, Groth16 and CP-Link proofs remain 292B and 128B, while Merkle openings dominate proof growth.

Appendix F, Table 9, compares rates $1/8, 1/4, 1/2$ with $t = 155, 189, 309$, respectively. Shorter codewords reduce encoding and commitment costs, but their smaller relative distance requires more queries, increasing sampled-proof cost and size.

q	Prove (s)		Verify (ms)		Proof (B)		Statement (B)	
	LAMP	zkMatrix	LAMP	zkMatrix	LAMP	zkMatrix	LAMP	zkMatrix
1	1.659	0.652	129.88	9.66	24,281.6	4,100	64	112
2	2.889	0.702	129.16	11.76	24,377.6	4,936	64	224
3	3.951	0.746	129.36	14.02	24,332.8	5,772	64	336
4	5.267	0.798	131.45	16.10	24,268.8	6,608	64	448
5	6.362	0.846	132.34	18.45	24,390.4	7,444	64	560
6	7.289	0.896	129.82	20.43	24,288	8,280	64	672
7	8.759	0.944	129.30	23.15	24,326.4	9,116	64	784
8	9.738	0.992	129.24	24.57	24,236.8	9,952	64	896
9	10.836	1.039	128.89	27.16	24,300.8	10,788	64	1,008
10	11.796	1.088	128.96	28.85	24,371.2	11,624	64	1,120

TABLE 5. INDEPENDENT EPYC COMPARISON FOR BATCHES OF q INDEPENDENT 128×128 PRODUCTS. TIMES AND BYTE COUNTS ARE MEANS OVER TEN RUNS; COMMITMENT GENERATION IS INCLUDED. PROOF AND STATEMENT SIZES USE THE ACCOUNTING OF TABLE 4.

7.3. Batching Matrix-Multiplication Claims

LAMP shares Pedersen leaves and Merkle openings across independent claims. Table 8 in Appendix F reports the component costs: constraints and Groth16 proving time grow with q, while Merkle proof size and verification time remain nearly constant.

Table 5 measures batch protocols, including at $q = 1$, rather than summing single-product times. zkMatrix proves faster throughout $q = 1, \ldots, 10$, taking 1.088s versus LAMP's 11.796s at $q = 10$. It aggregates the three projection IPAs, retaining one final dot-product IPA per claim; LAMP's SNARK circuit grows per claim.

LAMP's compressed proof stays near 24.3KB and verification near 0.13s. zkMatrix's proof grows from 4,100B to 11,624B and verification from 9.66ms to 28.85ms. Public statements count three compressed commitments and framing per zkMatrix claim ($112q$ bytes), versus LAMP's two roots (64 bytes). LAMP has flatter growth, but zkMatrix has smaller proofs and faster verification throughout the measured range; we infer no crossover beyond it.

7.4. GPT-2 Workload

Table 6 reports the GPT-2 results for sequence length 2^{10} and 36 products, with $\rho = 1/2, t = 309$. DualMatrix's public implementation uses the same matrix structures. Timings include encoding and commitments, and sizes include commitment data using the same byte accounting as Table 3. LAMP reduces Freivalds' constraints from 76,807,671 to 62,460,189 and proving from 408.69s to 317.13s ($1.29\times$). DualMatrix takes 519.97s ($1.64\times$ slower than LAMP). Freivalds verifies fastest with the smallest proof; DualMatrix also verifies faster with smaller proofs than LAMP. Heterogeneous matrix dimensions require multiple commitment layouts, CP-Link proofs, and Merkle openings, increasing LAMP's proof and verification costs. We have no CPmm, zkMatrix, or zkMaP measurements for this workload.

System	Constraints	Prove	Verify	Proof
LAMP	62,460,189	317.13 s	11.58 s	5,990,212 B
Freivalds	76,807,671	408.69 s	0.49 s	196 B
DualMatrix	–	519.97 s	5.22 s	2,934,952 B

TABLE 6. GPT-2 AT SEQUENCE LENGTH 2^{10} WITH 36 MATRIX-MULTIPLICATION CLAIMS. "FREIVALDS" DENOTES THE FREIVALDS-IN-SNARK BASELINE DEFINED IN SECTION 7.1.

8. Conclusion

We proposed LAMP, which moves vector–matrix products outside the SNARK and verifies sampled fold relations over encoded data. Merkle openings and CP-Link bind the CP-SNARK witnesses to data committed before sampling.

For fixed t and linear code-consistency checks, the main relation has $\mathcal{O}(tk + E_{\mathsf{code}})$ constraints and therefore grows linearly in k, unlike the Freivalds-in-SNARK baseline's $\mathcal{O}(k^2)$ constraints.

At $k = 2^{12}$, LAMP uses $9.68\times$ fewer constraints and proves $5.81\times$ faster than the measured Freivalds-in-SNARK baseline. It also outperforms DualMatrix in proving time at every shared dimension, by $3.62\times$ at $k = 2^{13}$. The independent zkMatrix comparison shows a different trade-off: LAMP proves $4.00\times$ faster at $k = 2048$, but zkMatrix verifies faster and produces smaller proofs at every measured dimension.

Shared-layout batching keeps LAMP's proof size and verification nearly constant, but zkMatrix proves faster for the measured 128×128 batches of one to ten claims. LAMP proves faster than the Freivalds-in-SNARK baseline and DualMatrix on GPT-2, at higher verification and communication costs. We do not evaluate LegoSNARK's CPmm; these measurements do not establish a performance advantage over that construction. Modified zkMaP measurements are excluded because they cannot represent a faithful, sound benchmark of the published protocol.

Future work includes proving the security of a Fiat–Shamir-transformed, non-interactive LAMP and sharing commitment layouts and openings across consecutive products of different dimensions.

References

[1] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology – EUROCRYPT 2016*. Berlin, Heidelberg: Springer, 2016, pp. 305–326.

[2] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Paper 2019/953, 2019. [Online]. Available: https://eprint.iacr.org/2019/953

[3] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2019, pp. 2111–2128.

[4] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zkSNARKs with universal and updatable SRS," in *Advances in Cryptology – EUROCRYPT 2020*. Cham: Springer, 2020, pp. 738–768.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[8] R. Freivalds, "Probabilistic machines can use less running time," in *Information Processing 77 (Proceedings of IFIP Congress 77)*. Amsterdam, The Netherlands: North-Holland, 1977, pp. 839–842.

[9] B. Parno, C. Gentry, J. Howell, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," Cryptology ePrint Archive, Paper 2013/279, 2013. [Online]. Available: https://eprint.iacr.org/2013/279

[10] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Advances in Cryptology – CRYPTO 2013*. Berlin, Heidelberg: Springer, 2013, pp. 71–89.

[11] M. Campanelli, D. Fiore, and A. Querol, "LegoSNARK: Modular design and composition of succinct zero-knowledge proofs," Cryptology ePrint Archive, Paper 2019/142, 2019. [Online]. Available: https://eprint.iacr.org/2019/142

[12] M. Cong, T. H. Yuen, and S.-M. Yiu, "zkMatrix: Batched short proof for committed matrix multiplication," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2024, pp. 289–305.

[13] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy*. New York, NY, USA: IEEE, 2018, pp. 315–334.

[14] M. Cong, T. H. Yuen, and S. M. Yiu, "DualMatrix: Conquering zkSNARK for large matrix multiplication," *Cybersecurity*, vol. 9, no. 1, p. 126, 2026.

[15] B. Deressa and M. A. Hasan, "zkmap: Zero-knowledge succinct non-interactive matrix multiplication proofs," *IACR Communications in Cryptology*, vol. 2, no. 3, 2025.

[16] S. Lee, H. Ko, J. Kim, and H. Oh, "vCNN: Verifiable convolutional neural network based on zk-SNARKs," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 4, pp. 4254–4270, 2024.

[17] J. Weng, J. Weng, G. Tang, A. Yang, M. Li, and J.-N. Liu, "pvcnn: Privacy-preserving and verifiable convolutional neural network testing," 2023. [Online]. Available: https://arxiv.org/abs/2201.09186

[18] Y. Zhang, M. Zheng, X. Chen, J. Hu, W. Shi, L. Ju, Y. Solihin, and Q. Lou, "zkVC: Fast zero-knowledge proof for private and verifiable computing," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, IEEE. New York, NY, USA: IEEE, 2025, pp. 1–7.

[19] T. Liu, X. Xie, and Y. Zhang, "zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2021, pp. 2968–2985.

[20] W. Qu, Y. Sun, X. Liu, T. Lu, Y. Guo, K. Chen, and J. Zhang, "zkGPT: An efficient non-interactive zero-knowledge proof framework for LLM inference," Cryptology ePrint Archive, Paper 2025/1184, 2025. [Online]. Available: https://eprint.iacr.org/2025/1184

[21] H. Sun, J. Li, and H. Zhang, "zkllm: Zero knowledge proofs for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2404.16109

[22] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning," in *30th USENIX Security Symposium (USENIX Security 21)*. Berkeley, CA, USA: USENIX Association, 2021, pp. 501–518.

[23] B.-J. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang, "zkML: An optimizing system for ML inference in zero-knowledge proofs," in *Proceedings of the Nineteenth European Conference on Computer Systems*. New York, NY, USA: ACM, 2024, pp. 560–574.

[24] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast Reed-Solomon Interactive Oracle Proofs of Proximity," in *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, Eds., vol. 107. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, pp. 14:1–14:17. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2018.14

[25] G. Arnon, A. Chiesa, G. Fenzi, and E. Yogev, "STIR: Reed–solomon proximity testing with fewer queries," Cryptology ePrint Archive, Paper 2024/390, 2024. [Online]. Available: https://eprint.iacr.org/2024/390

[26] ——, "WHIR: Reed–solomon proximity testing with super-fast verification," Cryptology ePrint Archive, Paper 2024/1586, 2024. [Online]. Available: https://eprint.iacr.org/2024/1586

[27] H. Zeilberger, B. Chen, and B. Fisch, "BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes," Cryptology ePrint Archive, Paper 2023/1705, 2023. [Online]. Available: https://eprint.iacr.org/2023/1705

[28] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup," Cryptology ePrint Archive, Paper 2022/1608, 2022. [Online]. Available: https://eprint.iacr.org/2022/1608

[29] A. Golovnev, J. Lee, S. Setty, J. Thaler, and R. S. Wahby, "Brakedown: Linear-time and field-agnostic snarks for r1cs," in *Advances in Cryptology – CRYPTO 2023*. Cham: Springer, 2023, pp. 193–226.

[30] M. Brehm, B. Chen, B. Fisch, N. Resch, R. D. Rothblum, and H. Zeilberger, "Blaze: Fast SNARKs from interleaved RAA codes," Cryptology ePrint Archive, Paper 2024/1609, 2024. [Online]. Available: https://eprint.iacr.org/2024/1609

[31] T. Xie, Y. Zhang, and D. Song, "Orion: Zero knowledge proof with linear prover time," Cryptology ePrint Archive, Paper 2022/1010, 2022. [Online]. Available: https://eprint.iacr.org/2022/1010

[32] T. den Hollander and D. Slamanig, "A crack in the firmament: Restoring soundness of the orion proof system and more," Cryptology ePrint Archive, Paper 2024/1164, 2024. [Online]. Available: https://eprint.iacr.org/2024/1164

[33] A. Belling, A. Soleimanian, and O. Bégassat, "Recursion over public-coin interactive proof systems; faster hash verification," Cryptology ePrint Archive, Paper 2022/1072, 2022. [Online]. Available: https://eprint.iacr.org/2022/1072

[34] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Journal of the ACM*, vol. 27, no. 4, pp. 701–717, 1980.

[35] R. Zippel, "Probabilistic algorithms for sparse polynomials," in *Symbolic and Algebraic Computation (EUROSAM '79)*. Berlin, Heidelberg: Springer, 1979, pp. 216–226.

[36] E. Ben-Sasson, D. Carmon, Y. Ishai, S. Kopparty, and S. Saraf, "Proximity gaps for reed-solomon codes," Cryptology ePrint Archive, Paper 2020/654, 2020. [Online]. Available: https://eprint.iacr.org/2020/654

[37] E. Kiltz and H. Wee, "Quasi-adaptive NIZK for linear subspaces revisited," Cryptology ePrint Archive, Paper 2015/216, 2015. [Online]. Available: https://eprint.iacr.org/2015/216

Appendix A. SNARK Security Definitions

Let $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ be an NP-relation family and $\mathbf{\Pi} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ a SNARK for $\mathcal{R}$, as in Section 3.4. We recall the properties used in this work.

Completeness. For every $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\lambda$, completeness requires

$$\Pr_{\substack{\mathsf{pp}\leftarrow\mathbf{\Pi}.\mathsf{Setup}(1^\lambda,\mathcal{R}_\lambda)\\ \pi\leftarrow\mathbf{\Pi}.\mathsf{Prove}(\mathsf{pp},\mathsf{x},\mathsf{w})}}[\mathbf{\Pi}.\mathsf{Verify}(\mathsf{pp},\mathsf{x},\pi)=1] \geq 1-\mathsf{negl}(\lambda).$$

It is perfect if this probability is one.

Knowledge soundness. $\mathbf{\Pi}$ satisfies knowledge soundness if, for every PPT prover $\mathcal{A}$, there exists a PPT extractor $\mathsf{Ext}^{\mathcal{A}}$ such that

$$\Pr_{\substack{\mathsf{pp}\leftarrow\mathbf{\Pi}.\mathsf{Setup}(1^\lambda,\mathcal{R}_\lambda)\\ (\mathsf{x},\pi)\leftarrow\mathcal{A}(\mathsf{pp})}}\begin{bmatrix}\mathbf{\Pi}.\mathsf{Verify}(\mathsf{pp},\mathsf{x},\pi)=1\\ \wedge\ (\mathsf{x},\mathsf{Ext}^{\mathcal{A}}(\mathsf{pp},\mathsf{x}))\notin\mathcal{R}_\lambda\end{bmatrix}\leq\mathsf{negl}(\lambda).$$

Zero knowledge. $\mathbf{\Pi}$ satisfies zero knowledge if there exists a PPT simulator Sim such that, for every $(\mathsf{x},\mathsf{w})\in\mathcal{R}_\lambda$, real proofs generated by $\mathbf{\Pi}.\mathsf{Prove}(\mathsf{pp},\mathsf{x},\mathsf{w})$ and simulated proofs generated by $\mathsf{Sim}(\mathsf{pp},\mathsf{x})$ are computationally indistinguishable for honestly generated pp. A SNARK satisfying zero knowledge is called a zk-SNARK.

Appendix B. CP-Link Security Definitions

Let $\mathcal{R}_{\mathsf{link}} = \{\mathcal{R}_{\mathsf{link},\lambda}\}_{\lambda\in\mathbb{N}}$ be the blockwise relation defined in Section 3.5, and let $\mathbf{\Pi}_{\mathsf{link}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a CP-Link proof system for it. Its language is

$$\mathcal{L}_{\mathsf{link},\lambda} = \{\mathsf{x}_{\mathsf{link}} : \exists \mathsf{w}_{\mathsf{link}}(\mathsf{x}_{\mathsf{link}},\mathsf{w}_{\mathsf{link}})\in\mathcal{R}_{\mathsf{link},\lambda}\}.$$

We use completeness, link soundness, and, for the NIZK or QA-NIZK instantiation, witness zero knowledge [11], [37].

Link completeness. For every $(\mathsf{x}_{\mathsf{link}},\mathsf{w}_{\mathsf{link}})\in\mathcal{R}_{\mathsf{link},\lambda}$, generate $\mathsf{pp}_{\mathsf{link}}\leftarrow\mathbf{\Pi}_{\mathsf{link}}.\mathsf{Setup}(1^\lambda,\mathcal{R}_{\mathsf{link},\lambda})$ and $\pi_{\mathsf{link}}\leftarrow\mathbf{\Pi}_{\mathsf{link}}.\mathsf{Prove}(\mathsf{pp}_{\mathsf{link}},\mathsf{x}_{\mathsf{link}},\mathsf{w}_{\mathsf{link}})$. Completeness requires

$$\Pr[\mathbf{\Pi}_{\mathsf{link}}.\mathsf{Verify}(\mathsf{pp}_{\mathsf{link}},\mathsf{x}_{\mathsf{link}},\pi_{\mathsf{link}})=1]\geq 1-\mathsf{negl}(\lambda).$$

It is perfect if this probability is one.

Link soundness. $\mathbf{\Pi}_{\mathsf{link}}$ satisfies link soundness with error $\epsilon_{\mathsf{link}}(\lambda)$ if, for every PPT adversary $\mathcal{A}$, after sampling $\mathsf{pp}_{\mathsf{link}}$ honestly and $(\mathsf{x}^\star_{\mathsf{link}},\pi^\star_{\mathsf{link}})\leftarrow\mathcal{A}(\mathsf{pp}_{\mathsf{link}})$,

$$\Pr\begin{bmatrix}\mathbf{\Pi}_{\mathsf{link}}.\mathsf{Verify}(\mathsf{pp}_{\mathsf{link}},\mathsf{x}^\star_{\mathsf{link}},\pi^\star_{\mathsf{link}})=1\\ \wedge\ \mathsf{x}^\star_{\mathsf{link}}\notin\mathcal{L}_{\mathsf{link},\lambda}\end{bmatrix}\leq\epsilon_{\mathsf{link}}(\lambda).$$

If $\epsilon_{\mathsf{link}}(\lambda)=\mathsf{negl}(\lambda)$, we say that $\mathbf{\Pi}_{\mathsf{link}}$ is link-sound.

Witness zero knowledge. There exists a PPT simulator $\mathsf{Sim}_{\mathsf{link}}$ such that, for every $(\mathsf{x}_{\mathsf{link}},\mathsf{w}_{\mathsf{link}})\in\mathcal{R}_{\mathsf{link},\lambda}$, real proofs produced by $\mathbf{\Pi}_{\mathsf{link}}.\mathsf{Prove}$ and proofs produced by $\mathsf{Sim}_{\mathsf{link}}$ are computationally indistinguishable for honestly generated $\mathsf{pp}_{\mathsf{link}}$.

Appendix C. Detailed Proof of Theorem 6.2

Proof. We consider the public-coin interactive protocol.

Perfect completeness. Let the honest prover hold $\mathbf{A},\mathbf{B},\mathbf{C}$ such that $\mathbf{AB}=\mathbf{C}$. For

$$\mathbf{r}=(1,r,\dots,r^{k-1}),\qquad \mathbf{s}=(1,s,\dots,s^{k-1}),$$

set

$$\mathbf{x}=\mathbf{rA},\qquad \mathbf{y}=\mathbf{xB},\qquad \mathbf{z}=\mathbf{rC},\qquad \mathbf{b}=\mathbf{sB}.$$

Then $\mathbf{y}=\mathbf{z}$, and linearity of the code gives, for every $j\in I$,

$$\widehat{\mathbf{x}}[j]=\mathsf{Fold}(\mathbf{r},\widehat{\mathbf{A}}[*,j]),\quad \widehat{\mathbf{y}}[j]=\mathsf{Fold}(\mathbf{x},\widehat{\mathbf{B}}[*,j]),$$
$$\widehat{\mathbf{z}}[j]=\mathsf{Fold}(\mathbf{r},\widehat{\mathbf{C}}[*,j]),\quad \widehat{\mathbf{b}}[j]=\mathsf{Fold}(\mathbf{s},\widehat{\mathbf{B}}[*,j]).$$

All remaining checks are generated honestly and therefore accept.

Backend bad events. Write $\mathsf{Bad}_{\mathsf{cp}}$, $\mathsf{Bad}_{\mathsf{open}}$, $\mathsf{Bad}_{\mathsf{link}}$, and $\mathsf{Bad}_{\mathsf{code}}$ for failure of the CP-SNARK, commitment binding, CP-Link soundness, and encoding consistency, respectively, and let $\mathsf{Bad}_{\mathsf{back}}$ be their union. By Definition 6.1,

$$\Pr[\mathsf{Bad}_{\mathsf{back}}]\leq\epsilon_{\mathsf{cp}}(\lambda)+\epsilon_{\mathsf{open}}(\lambda)+\epsilon_{\mathsf{link}}(\lambda)+\epsilon_{\mathsf{code}}(\lambda).$$

For the remainder of the proof, condition on $\neg\mathsf{Bad}_{\mathsf{back}}$. The input words are fixed before r,s, the intermediate words are fixed before the query tuple, and each sampled value is bound to the value opened from its commitment.

Far-input case. For an interleaved word $M\in\mathbb{F}^{k\times n}$, write

$$\Delta_{\mathsf{IRS}}(M,\mathcal{C}^k)=\min_{M'\in\mathcal{C}^k}\frac{1}{n}|\{j\in[n]:M[*,j]\neq M'[*,j]\}|.$$

For $\rho=(1,\rho,\dots,\rho^{k-1})$, the structured-fold proximity bound is

$$\Pr_{\rho\leftarrow\mathbb{F}}[\Delta(\rho M,\mathcal{C})\leq\tau]\leq\frac{(k-1)n}{|\mathbb{F}|}$$

whenever $\Delta_{\mathsf{IRS}}(M,\mathcal{C}^k)>\tau$. Indeed, cancellation at a given position is governed by a nonzero polynomial in ρ of degree at most $k-1$; summing over the n positions gives the stated bound.

Suppose that an input word is τ-far from $\mathcal{C}^k$, and choose the first such word in the order $\widehat{\mathbf{A}}, \widehat{\mathbf{B}}, \widehat{\mathbf{C}}$. This choice is made from the committed words before the challenges are sampled. Use the $\mathbf{r}$-fold for $\widehat{\mathbf{A}}$ or $\widehat{\mathbf{C}}$, and the independent $\mathbf{s}$-fold for $\widehat{\mathbf{B}}$; the $\mathbf{x}\widehat{\mathbf{B}}$ fold is not needed in this case. Applying the bound to this single chosen word, the probability of a bad fold is at most

$$\frac{(k-1)n}{|\mathbb{F}|}.$$

Conditioned on the complementary event, the chosen fold is more than τ-far from the claimed intermediate codeword. Hence independent queries $I = (j_1, \ldots, j_t) \leftarrow\$ [n]^t$ all miss the disagreement set with probability at most

$$(1-\tau)^t.$$

Close-input case. Now suppose that the three words have unique decodings $\mathbf{A}^*, \mathbf{B}^*, \mathbf{C}^*$ such that

$$\Delta_{\mathsf{IRS}}(\widehat{\mathbf{A}}, \mathsf{Enc}(\mathbf{A}^*)) \le \tau,$$
$$\Delta_{\mathsf{IRS}}(\widehat{\mathbf{B}}, \mathsf{Enc}(\mathbf{B}^*)) \le \tau,$$
$$\Delta_{\mathsf{IRS}}(\widehat{\mathbf{C}}, \mathsf{Enc}(\mathbf{C}^*)) \le \tau.$$

Each error set contains at most τn interleaved columns, so

$$\mathbf{r}\widehat{\mathbf{A}} \text{ is } \tau\text{-close to } \mathsf{Enc}(\mathbf{r}\mathbf{A}^*),$$
$$\mathbf{x}\widehat{\mathbf{B}} \text{ is } \tau\text{-close to } \mathsf{Enc}(\mathbf{x}\mathbf{B}^*),$$
$$\mathbf{r}\widehat{\mathbf{C}} \text{ is } \tau\text{-close to } \mathsf{Enc}(\mathbf{r}\mathbf{C}^*),$$
$$\mathbf{s}\widehat{\mathbf{B}} \text{ is } \tau\text{-close to } \mathsf{Enc}(\mathbf{s}\mathbf{B}^*).$$

Assume $\mathbf{A}^*\mathbf{B}^* \neq \mathbf{C}^*$. Some coordinate of $\mathbf{r}(\mathbf{A}^*\mathbf{B}^* - \mathbf{C}^*)$ is a nonzero polynomial in r of degree at most $k-1$. Thus

$$\Pr_r[\mathbf{r}(\mathbf{A}^*\mathbf{B}^* - \mathbf{C}^*) = 0] \le \frac{k-1}{|\mathbb{F}|}.$$

Except with this probability, the equality $\mathbf{y} = \mathbf{z}$ and the four relations

$$\mathbf{x} = \mathbf{r}\mathbf{A}^*, \qquad \mathbf{y} = \mathbf{x}\mathbf{B}^*, \qquad \mathbf{z} = \mathbf{r}\mathbf{C}^*, \qquad \mathbf{b} = \mathbf{s}\mathbf{B}^*$$

cannot hold simultaneously. The fourth relation supplies an independent check of the middle input. The equality $\mathbf{y} = \mathbf{z}$ is checked over the complete vector and is not another sampled relation.

Choose the first false fold relation in a fixed order. This choice is made after the intermediate commitment but before I is sampled. Let c^* be the correct Reed–Solomon codeword for this relation, let $c \neq c^*$ be the claimed codeword, and let w be the folded input word. We have $\Delta(w, c^*) \le \tau$ and $\Delta(c, c^*) \ge \delta_{\mathsf{RS}}$, whence

$$\begin{aligned}\Delta(w, c) &\ge \Delta(c, c^*) - \Delta(w, c^*) \\ &\ge \delta_{\mathsf{RS}} - \tau.\end{aligned}$$

The probability that all t queries miss the disagreement set is at most

$$(1 - \delta_{\mathsf{RS}} + \tau)^t.$$

Acceptance requires this fixed relation to pass, so no union bound over the four relations is necessary.

Combining the bounds. Combining the two cases with the backend errors yields

$$\begin{aligned}\Pr[\mathsf{Acc}] \le{}& \epsilon_{\mathsf{cp}}(\lambda) + \epsilon_{\mathsf{open}}(\lambda) + \epsilon_{\mathsf{link}}(\lambda) + \epsilon_{\mathsf{code}}(\lambda) \\ &+ \frac{(k-1)n}{|\mathbb{F}|} + \frac{k-1}{|\mathbb{F}|} \\ &+ \max\{(1-\tau)^t, (1-\delta_{\mathsf{RS}}+\tau)^t\}.\end{aligned}$$

For the rate-one-half parameters $n = 2k$ with even k,

$$\delta_{\mathsf{RS}} = \frac{1}{2} + \frac{1}{2k}, \qquad \tau = \tau_{\mathsf{UD}} = \frac{1}{4}.$$

The larger sampling term is $(3/4)^t$, and

$$\left(\frac{3}{4}\right)^t \le 2^{-128} \quad \Longleftrightarrow \quad t \ge \frac{-128}{\log_2(3/4)} \approx 308.38.$$

Thus $t = 309$ gives 128-bit sampling error. This numerical choice omits the backend and field-size terms, which are assumed negligible.

Zero knowledge. Use the CP-SNARK and CP-Link simulators and replace the witness blocks by hiding Pedersen commitments. Since the Merkle layer authenticates only these commitments and reveals no openings, the resulting transcript is computationally indistinguishable from a real execution. □

Appendix D. Barycentric Reed–Solomon Consistency Check

This appendix specifies the Reed–Solomon instantiation of $\mathsf{EncCheck}_{\mathcal{C}}$ used in Section 5.3. Let

$$D = \{\omega_0, \ldots, \omega_{n-1}\} \subset \mathbb{F}$$

be the public evaluation domain. A message $\mathbf{v} = (v_0, \ldots, v_{k-1}) \in \mathbb{F}^k$ defines the degree-$< k$ polynomial

$$p_{\mathbf{v}}(X) = \sum_{\ell=0}^{k-1} v_\ell X^\ell, \qquad \mathsf{Enc}(\mathbf{v}) = (p_{\mathbf{v}}(\omega_i))_{i\in[n]}.$$

Given a claimed encoded word $\widehat{\mathbf{v}} \in \mathbb{F}^n$, let $P_{\widehat{\mathbf{v}}}$ be the unique degree-$< n$ polynomial satisfying

$$P_{\widehat{\mathbf{v}}}(\omega_i) = \widehat{\mathbf{v}}[i] \qquad \text{for all } i \in [n].$$

The check samples $\alpha \leftarrow\$ \mathbb{F} \setminus D$ after the claimed word is fixed and compares $P_{\widehat{\mathbf{v}}}(\alpha)$ with $p_{\mathbf{v}}(\alpha)$.

For efficient evaluation, define the domain polynomial and barycentric weights

$$L_D(X) = \prod_{i\in[n]} (X - \omega_i),$$
$$\lambda_i = \left(\prod_{\ell\in[n]\setminus\{i\}} (\omega_i - \omega_\ell)\right)^{-1}, \qquad i \in [n].$$

For every $\alpha \notin D$, barycentric interpolation gives

$$P_{\widehat{\mathbf{v}}}(\alpha) = L_D(\alpha) \sum_{i\in[n]} \lambda_i \frac{\widehat{\mathbf{v}}[i]}{\alpha - \omega_i}.$$

Thus the concrete Reed–Solomon consistency check is

$$\mathsf{EncCheck}_{\mathsf{RS}}(\widehat{\mathbf{v}}, \mathbf{v}; \alpha) = 1 \iff \sum_{\ell=0}^{k-1} v_\ell \alpha^\ell = L_D(\alpha) \sum_{i\in[n]} \lambda_i \frac{\widehat{\mathbf{v}}[i]}{\alpha - \omega_i}.$$

The right-hand side is a linear form in the claimed encoded symbols once α is fixed. Equivalently, the verifier can derive the interpolation coefficients

$$\gamma_i(\alpha) = L_D(\alpha)\lambda_i(\alpha - \omega_i)^{-1}$$

and the circuit checks

$$\sum_{\ell=0}^{k-1} v_\ell \alpha^\ell = \sum_{i\in[n]} \gamma_i(\alpha)\widehat{\mathbf{v}}[i].$$

This costs $\mathcal{O}(k+n)$ field operations per checked point, after the domain-dependent weights are precomputed.

Lemma D.1 (Barycentric code-consistency soundness)**.** *If $\widehat{\mathbf{v}} \neq \mathsf{Enc}(\mathbf{v})$ and $\alpha \leftarrow\$ \mathbb{F} \setminus D$ is sampled after $\widehat{\mathbf{v}}$ and $\mathbf{v}$ are fixed, then*

$$\Pr[\mathsf{EncCheck}_{\mathsf{RS}}(\widehat{\mathbf{v}}, \mathbf{v}; \alpha) = 1] \leq \frac{n-1}{|\mathbb{F}| - n}.$$

Proof. If $\widehat{\mathbf{v}} \neq \mathsf{Enc}(\mathbf{v})$, then the interpolation polynomial $P_{\widehat{\mathbf{v}}}$ differs from the degree-$< k$ polynomial $p_{\mathbf{v}}$. Their difference

$$Q(X) = P_{\widehat{\mathbf{v}}}(X) - p_{\mathbf{v}}(X)$$

is a nonzero polynomial of degree at most $n-1$. The check accepts exactly when $Q(\alpha) = 0$. Since α is uniform over $\mathbb{F}\setminus D$, at most $n-1$ choices of α can make the check accept, which proves the bound. □

In LAMP, this check is applied to $\widehat{\mathbf{x}}, \widehat{\mathbf{y}}, \widehat{\mathbf{z}}, \widehat{\mathbf{b}}$ at $\alpha_x, \alpha_y, \alpha_z, \alpha_b$, respectively.

Appendix E. CP-Link Instantiation

This appendix describes the CP-Link instantiation used in our implementation. We use a pairing-based quasi-adaptive link argument, denoted QALink, for linking one SNARK-side Pedersen commitment to several external Pedersen commitments. The verifier can also batch multiple QALink verification equations into one multi-pairing check.

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear pairing over prime-order groups of order $|\mathbb{F}|$. We write the group operation in $\mathbb{G}_1$ additively, and let $g_2 \in \mathbb{G}_2$ be a generator.

Statement and witness. Fix a block count q and a block length ℓ. Let $\mathsf{ck}_S = ((G^S_{i,\nu})_{i\in[q],\nu\in[\ell]}, H^S)$ be the SNARK-side commitment key for a commitment to q concatenated blocks, and let $\mathsf{ck}_E = ((G^E_\nu)_{\nu\in[\ell]}, H^E)$ be the external Pedersen commitment key for one block.

The public statement is $\mathsf{x}_{\mathsf{link}} = ((\mathsf{ck}_S, \widetilde{\mathsf{cm}}), (\mathsf{ck}_E, \mathsf{cm}_0, \ldots, \mathsf{cm}_{q-1}))$. The witness is

$$\mathsf{w}_{\mathsf{link}} = \big(\mathbf{m}_0, \ldots, \mathbf{m}_{q-1}, \widetilde{o}, o_0, \ldots, o_{q-1}\big),$$

where each block $\mathbf{m}_i \in \mathbb{F}^\ell$. The intended relation is that $\widetilde{\mathsf{cm}}$ commits to the ordered concatenation $\mathbf{m}_0\|\cdots\|\mathbf{m}_{q-1}$ under ck_S, while each cm_i commits to the corresponding block $\mathbf{m}_i$ under ck_E. Equivalently,

$$\widetilde{\mathsf{cm}} = \sum_{i\in[q]}\sum_{\nu\in[\ell]} \mathbf{m}_i[\nu] G^S_{i,\nu} + \widetilde{o} H^S,$$
$$\mathsf{cm}_i = \sum_{\nu\in[\ell]} \mathbf{m}_i[\nu] G^E_\nu + o_i H^E \qquad \text{for every } i \in [q].$$

Setup. The setup algorithm takes $(\mathsf{ck}_S, \mathsf{ck}_E, q, \ell)$ as input. It samples $k_0, k_1, \ldots, k_q, a \leftarrow\$ \mathbb{F}^*$ and defines

$$P_{i,\nu} := k_0 G^S_{i,\nu} + k_{i+1} G^E_\nu \qquad (i \in [q],\ \nu \in [\ell]).$$

It outputs

$$\mathsf{pk}_{\mathsf{link}} = \Big((P_{i,\nu})_{i\in[q],\,\nu\in[\ell]}, k_0 H^S, (k_{i+1}H^E)_{i\in[q]}\Big),$$
$$\mathsf{vk}_{\mathsf{link}} = (C_0, C_1, \ldots, C_q, A) = (ak_0 g_2, ak_1 g_2, \ldots, ak_q g_2, a g_2).$$

The trapdoors $k_0, \ldots, k_q, a$ are discarded after setup.

Proving. Given $\mathsf{x}_{\mathsf{link}}$ and $\mathsf{w}_{\mathsf{link}}$, the prover outputs one group element

$$\pi_{\mathsf{link}} := \sum_{i\in[q]}\sum_{\nu\in[\ell]} \mathbf{m}_i[\nu] P_{i,\nu} + \widetilde{o}(k_0 H^S) + \sum_{i\in[q]} o_i (k_{i+1} H^E).$$

By construction and by the homomorphism of Pedersen commitments,

$$\pi_{\mathsf{link}} = k_0 \widetilde{\mathsf{cm}} + \sum_{i\in[q]} k_{i+1} \mathsf{cm}_i.$$

This identity is the algebraic relation checked by the verifier.

Verification. The verifier accepts if and only if

$$e(\widetilde{\mathsf{cm}}, C_0) \cdot \prod_{i\in[q]} e(\mathsf{cm}_i, C_{i+1}) = e(\pi_{\mathsf{link}}, A).$$

Equivalently, the verifier checks

$$e(\widetilde{\mathsf{cm}}, C_0) \cdot \prod_{i\in[q]} e(\mathsf{cm}_i, C_{i+1}) \cdot e(\pi_{\mathsf{link}}, -A) = 1_{\mathbb{G}_T}.$$

By bilinearity, honestly generated proofs satisfy this equation. We rely on the quasi-adaptive linear-subspace assumption underlying QALink [37] for link soundness, as formalized in Appendix B.

Verifier cost. Verification uses $q+2$ pairing terms and one final exponentiation via multi-pairing; the proof contains one $\mathbb{G}_1$ element and the verification key contains $q+2$ $\mathbb{G}_2$ elements.

k	Setup		Commit			Prove			Verify			Proof		
	Groth16	CP-Link	Encode	ABC	XYZ	Merkle	Groth16	CP-Link	Groth16	Merkle	CP-Link	Merkle	Groth16	CP-Link
2^7	10.06 s	0.66 s	0.02 s	0.07 s	0.01 s	1 ms	1.47 s	0.04 s	2 ms	1 ms	0.12 s	43,904 B	292 B	128 B
2^8	20.06 s	1.29 s	0.04 s	0.21 s	0.02 s	1 ms	2.56 s	0.07 s	2 ms	1 ms	0.13 s	51,776 B	292 B	128 B
2^9	39.83 s	2.55 s	0.15 s	0.73 s	0.04 s	1 ms	4.71 s	0.11 s	2 ms	2 ms	0.12 s	64,896 B	292 B	128 B
2^{10}	80.40 s	4.99 s	0.49 s	2.63 s	0.16 s	1 ms	8.81 s	0.19 s	2 ms	2 ms	0.13 s	79,552 B	292 B	128 B
2^{11}	155.01 s	9.89 s	1.56 s	7.78 s	0.52 s	1 ms	17.15 s	0.46 s	2 ms	3 ms	0.12 s	96,512 B	292 B	128 B
2^{12}	320.11 s	19.55 s	5.31 s	27.13 s	3.34 s	1 ms	34.15 s	0.80 s	2 ms	3 ms	0.13 s	115,584 B	292 B	128 B
2^{13}	638.92 s	40.82 s	18.54 s	99.03 s	8.93 s	1 ms	67.37 s	1.52 s	2 ms	4 ms	0.13 s	136,320 B	292 B	128 B

TABLE 7. COMPONENT-LEVEL MEASUREMENTS FOR LAMP.

q	Constraints	Commit (s)			Prove (s)		Merkle proof (B)
		Encode	ABC	XYZ	Groth16	CP-Link	
1	167,065	0.04	0.07	0.01	1.42	0.04	43,776
2	328,126	0.04	0.11	0.01	2.55	0.07	43,328
3	489,187	0.04	0.15	0.01	3.60	0.10	43,712
4	650,248	0.05	0.20	0.02	4.79	0.11	43,648
5	811,309	0.05	0.23	0.02	5.72	0.14	43,776
6	972,370	0.05	0.27	0.02	6.66	0.18	43,712
7	1,133,431	0.05	0.32	0.02	7.83	0.17	43,904
8	1,294,492	0.06	0.37	0.02	8.69	0.18	44,096
9	1,455,553	0.06	0.38	0.02	9.81	0.23	43,648
10	1,616,614	0.06	0.42	0.03	10.72	0.22	43,712

TABLE 8. BATCHING q INDEPENDENT MATRIX-MULTIPLICATION CLAIMS $\{\mathbf{A}_i\mathbf{B}_i = \mathbf{C}_i\}_{i\in[q]}$ AT FIXED $k = 2^7$. FOR EVERY q, MERKLE PROVING TAKES 1 MS, AND THE GROTH16 AND CP-LINK PROOFS ARE 292 B AND 128 B, RESPECTIVELY.

Batching pairing equations. Suppose the verifier must check s independent QALink statements. For the h-th statement, let

$$E_h := e(\widetilde{\mathsf{cm}}^{(h)}, C_0^{(h)}) \cdot \prod_{i\in[q_h]} e(\mathsf{cm}_i^{(h)}, C_{i+1}^{(h)}) \cdot e(\pi_{\mathsf{link}}^{(h)}, -A^{(h)}).$$

The unbatched verifier checks $E_h = 1_{\mathbb{G}_T}$ for every h. The batched verifier first derives nonzero coefficients $\lambda_1, \ldots, \lambda_s \in \mathbb{F}^*$ by hashing all public statements, verifying keys, proofs, and the surrounding transcript context. It then checks the single equation

$$\prod_{h=1}^{s} E_h^{\lambda_h} = 1_{\mathbb{G}_T}.$$

Operationally, the verifier scales the $\mathbb{G}_1$ inputs by the batching coefficients and performs one multi-pairing check. This requires $\sum_{h=1}^{s}(q_h + 2)$ Miller-loop terms, the same number of scalar multiplications in $\mathbb{G}_1$, and one final exponentiation. If any individual equation is invalid, a fresh random nonzero coefficient vector satisfies the batched equation with probability at most $1/(|\mathbb{F}| - 1)$. For Fiat–Shamir-derived coefficients, this bound is interpreted in the random oracle model.

ρ	n	Constraints	Setup (s)	Encode (s)	Commit (s)	Prove (s)	Verify (s)	Proof (B)
$k = 2^7$								
1/2	256	167,065	10.72	0.02	0.09	1.51	0.13	44,324
1/4	512	107,503	6.73	0.03	0.15	0.95	0.08	37,252
1/8	1,024	95,917	5.97	0.04	0.30	0.86	0.07	41,796
$k = 2^8$								
1/2	512	329,378	21.35	0.04	0.23	2.63	0.13	52,196
1/4	1,024	211,457	13.31	0.07	0.46	1.80	0.08	46,404
1/8	2,048	188,616	11.76	0.09	0.89	1.61	0.07	49,540
$k = 2^9$								
1/2	1,024	655,246	42.38	0.15	0.78	4.82	0.13	65,316
1/4	2,048	420,118	26.47	0.21	1.52	3.12	0.08	56,644
1/8	4,096	374,649	22.94	0.32	3.04	2.84	0.07	56,964
$k = 2^{10}$								
1/2	2,048	1,306,973	85.39	0.49	2.79	9.00	0.13	79,972
1/4	4,096	837,449	52.61	0.68	5.42	5.87	0.08	67,844
1/8	8,192	746,715	46.28	1.09	10.75	5.22	0.07	67,972
$k = 2^{11}$								
1/2	4,096	2,609,194	164.90	1.56	8.29	17.61	0.13	96,932
1/4	8,192	1,671,349	105.49	2.26	16.18	10.90	0.08	80,772
1/8	16,384	1,490,212	90.18	3.94	31.82	9.51	0.07	79,044
$k = 2^{12}$								
1/2	8,192	5,214,878	339.66	5.31	30.47	34.95	0.13	116,004
1/4	16,384	3,339,902	208.45	8.36	57.37	21.26	0.08	93,060
1/8	32,768	2,977,841	182.63	14.80	111.69	18.70	0.07	87,556
$k = 2^{13}$								
1/2	16,384	10,426,237	679.74	18.54	107.96	68.89	0.13	136,740
1/4	32,768	6,677,017	409.98	32.11	208.03	42.26	0.08	102,980
1/8	65,536	5,953,099	366.02	60.31	405.16	36.60	0.07	97,732

TABLE 9. PERFORMANCE OF LAMP ACROSS CODE RATES. FOR $\rho = 1/2, 1/4, 1/8$, THE SAMPLE COUNT IS $t = 309, 189, 155$, RESPECTIVELY.

Appendix F. Additional Evaluation Measurements

Tables 7 and 8 preserve component and batch results; Table 9 compares code rates, all with original accounting. Setup sums Groth16 and CP-Link; commitment excludes encoding; proving sums Merkle, Groth16, and CP-Link generation, excluding encoding and commitments.