

Verifier-Aligned Canonical Source Binding for Homomorphic Encryption Computations

Subeen Cho¹[0009–0004–4352–8021], Jiwon Bang¹[0009–0006–5453–1722],
Minseo Kim¹[0009–0008–1979–5778], Seungwon Lee¹[0009–0009–9583–1387], and
Hwajeong Seo¹[0000–0003–0069–9061]

¹Department of Convergence Security,
Hansung University, Seoul (02876), South Korea,
{chosubin1208, bgjiw0n754, msqoo0212, dkajdfhd1, hwajeong84}@gmail.com

Abstract. q0 preprocessing verification and native HE computation verification can each succeed without establishing that their results refer to the same canonical source. To address this source mismatch across verification paths, we propose a verifier-aligned public-input transfer procedure that links heterogeneous authenticated artifacts to a single canonical-source relation. The public verifier reconstructs challenges and targets from the actual statements, roots, approved verification keys, and authenticated positions. The chunk circuits then verify the source and mask commitments together with the canonical range, sign, and partial-sum relations. The full-input implementation processes 524,288 coefficients across 128 blocks and 4,096 native-bridge chunks. Public reconstruction and verification without stored challenge caches take 184.77 seconds, with 27,345,153 B of explicit public data. In a representative-block comparison for the same relation, sharing duplicate hash, range, and sign computations reduces the number of constraints by 33.53% and proving time by 31.10%, while preserving the proof-body size. In contrast, three upstream proof artifacts account for 88.44% of the total public data. These results show that eliminating redundant circuit computations is effective for reducing prover-side cost, whereas the dominant public-data cost originates from separate upstream proof artifacts. Prover computation and public-data size should therefore be treated as distinct optimization targets in source-binding verification. This work specifies the common-source relation between heterogeneous HE verification results and evaluates its implementation at full input scale.

Keywords: Homomorphic encryption · Verifiable computation · Canonical source binding · Cross-modulus consistency · Proof composition

1 Introduction

Homomorphic encryption enables computation on encrypted data without decryption [13,7,11]. However, returning a ciphertext result does not by itself establish that the intended computation relation was evaluated correctly. Verifiable HE addresses this requirement through proofs of encrypted computation [2,9,21].

When an HE computation is divided into preprocessing, representation conversion, and native computation stages, verifying each stage independently is not sufficient to establish consistency across the entire computation path.

In multi-stage HE verification, proofs generated from different sources or authenticated contexts can still be accepted individually when each satisfies its own verification relation. Valid proofs from separate executions, or positions, commitments, challenges, and targets derived from different roots, may therefore be combined into a single computation path. Each stage can pass verification while the resulting path does not refer to the same source. Joint verification must consequently establish not only individual proof validity but also that the accepted artifacts refer to the same source and authenticated computation context.

Prior work provides techniques for verifying HE computation relations, canonical-value consistency across moduli, and proof composition through committed values [19,8,16]. These techniques establish the validity of individual relations and representations. However, when authenticated artifacts accepted by heterogeneous HE verifiers are linked, an explicit transfer condition is still required to preserve their roots, indices, commitments, challenges, targets, and verification-key context while binding them to the same canonical source.

We consider this problem between a preprocessing path and a native computation path in HE-based recursive computation. The two paths operate on the same canonical coefficients but use different representations and block orders. An explicit index alignment is therefore required to map source blocks to their authenticated native columns. Establishing a common source requires not only source commitments but also block and chunk positions, masks, authenticated targets, and verification keys to remain consistent with the accepted context.

To address this problem, we propose a verifier-aligned public-input transfer procedure that links authenticated artifacts from different verification paths to a single canonical-source relation. The public verifier checks the accepted preprocessing, native-computation, and forward proofs and reconstructs challenges and targets from authenticated positions. Chunk arguments open the source and mask commitments and enforce canonical ranges, signs, and partial-sum relations. The verifier then checks that the aggregated chunk results remain consistent with the approved public context.

1.1 Main Contributions

The main contributions of this work are summarized as follows.

- **Specification.** We define a verifier-aligned public-input transfer relation that binds roots, indices, commitments, authenticated targets, and verification context accepted by heterogeneous HE verifiers to a single canonical source.
- **Formalization.** We distinguish individual proof validity from cross-stage source consistency and state a conditional uniqueness proposition for joint acceptance under component soundness, commitment binding, knowledge soundness, and fixed verification context.

- **Empirical Validation.** We evaluate the relation at a full-input scale of 524,288 coefficients. Sharing duplicate circuit computations reduces constraints by 33.53% and proving time by 31.10%, while three upstream proof artifacts account for 88.44% of total public data. The results show that prover computation and public-data size require distinct optimization targets.

2 Background and Related Work

2.1 HE Representations and Conversion

Lattice-based HE represents a polynomial in different forms depending on the computation stage, including the coefficient domain, the transform domain, and RNS components over smaller moduli. These representations support efficient polynomial arithmetic, but equality across representations does not by itself determine the same underlying integer. In particular, modular equality cannot distinguish different integers in the same residue class. A canonical range and sign convention are therefore required when relating these representations. RNS representation and NTT-based computation are widely used for this purpose in HE implementations [3,10].

Digit and native-residue representations must likewise be derived from the same integer. For a canonical coefficient x , the relation between the two representations is expressed as follows.

$$x = \sum_{j=0}^{t-1} d_j B^j, \quad 0 \leq d_j < B, \quad r_i \equiv x \pmod{p_i}.$$

In this relation, d_j denotes a digit in base B , and r_i denotes the residue modulo the native modulus p_i . The fixed digit range and canonical sign convention determine whether the two representations correspond to the same integer.

These representation differences also arise at the interface between the pre-processing and native computation paths considered in this work. Both paths are derived from the same canonical coefficients but use different representations and data layouts. The subsequent source-binding relation therefore connects not only the representation conversion but also the corresponding coefficient positions and authenticated context.

2.2 Polynomial and Circuit Proofs

IOP-based techniques such as FRI test whether a public oracle is close to a low-degree Reed–Solomon codeword, while Merkle openings authenticate values at specific committed positions. However, correctly authenticated positions alone do not establish that values used by different verification paths originate from the same polynomial or source [4,6,5].

SNARKs such as Groth16 provide succinct arguments for relations expressed as arithmetic circuits [15]. Such circuits can enforce range, sign, commitment-opening, and arithmetic relations, but the authenticated context from which

their public inputs originate remains a separate verification condition. Circuit soundness and provenance of public inputs must therefore be considered separately.

In non-interactive proofs, the Fiat–Shamir transform derives challenges from the public transcript, and the statement, roots, and commitments must be fixed before challenge derivation [14,5]. In this work, upstream authenticated artifacts are linked to chunk-level circuit arguments by reconstructing the required targets and challenges from authenticated positions and context. This ensures that both verification layers refer to the same approved public context.

2.3 Threat Model and Verification Objectives

The prover submits proofs for the preprocessing and native computation paths, source and mask commitments, chunk arguments, and partial sums. The verifier is assumed to know the approved relation, verification keys, and public-input format for each path. Different paths may use different roots and verification keys, but every artifact used for linking must belong to its approved context and refer to the same canonical source.

An adversarial prover may combine valid artifacts from different executions or replace source commitments, block or chunk order, masks, sign or modulus rules, and challenges or targets derived from another root. In such cases, each proof may remain individually valid while the combined computation path no longer refers to the same source. Figure 1 illustrates the distinction between individual verification acceptance and the additional common-source binding condition.

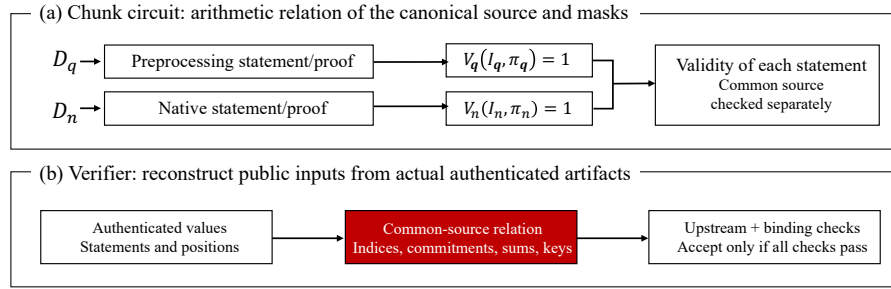


Fig. 1. Scope of individual statement verification and source-binding checks. (a) Individual acceptance of two statements does not additionally check for a common source. (b) Each path’s authenticated context is compared with the bridge circuit’s public input. The roots and verification keys themselves need not have identical values.

The verification objectives are completeness for correctly linked inputs and source-binding soundness against acceptance that combines different sources or authenticated contexts as one computation. The linking relation therefore checks not only the source but also positions, commitments, masks, authenticated targets, and verification-key context. These guarantees assume sound upstream

proofs, binding commitments, knowledge-sound chunk arguments, and a fixed public context before challenge derivation. Section 3.7 states the resulting security scope.

2.4 Related Works and Scope of This Work

Prior work on HE verification has addressed arithmetic relations, representation conversion, cross-modulus consistency, and proof composition. These systems differ in their target HE operations, verification relations, and verifier requirements. More recent studies also consider consistency across representations or composition between specialized proof systems. Table 1 summarizes the main verification targets and functional scope of these related approaches together with this work.

Table 1. Main functions and scope of related approaches and this work.

Approach	HE operations	Cross-representation	Argument composition	Main scope
HELIOPOLIS [2]	✓	—	—	HE verification
PRISM [21]	✓	✓	—	RNS-HE relations
GlueLUT [19]	—	✓	—	Residue-ring lookup
LegoSNARK [8]	—	—	✓	Proof composition
zkFFT [16]	—	✓	—	FFT relation
This work	✓	✓	$\triangle$	q0/native source binding

✓ : directly constructed, $\triangle$: application using existing components,
 — : not a primary target. Symbols indicate comparison scope only.

HELIOPOLIS [2] combines plaintext-space relations with IOPs and requires the verifier to use the decryption key in its basic construction. Approximate HE verification [9] covers CKKS arithmetic and maintenance operations. Lasagne [20] and HasteBoots [17] target SIMD computation in BGV/BFV and programmable bootstrapping in TFHE, respectively. PRISM [21] provides cross-field relations and NTT verification for RNS-based HE, while GlueLUT [19] uses canonical-integer cross-modulus consistency to relate different representations. LegoSNARK [8] supports proof composition through committed values, and zkFFT [16] proves FFT relations between committed vectors.

These studies provide foundations for verifying individual HE computations, consistency across representations, and composition through committed values. However, when artifacts already accepted by heterogeneous HE verifiers are combined, the public-input transfer condition that preserves their roots, indices, commitments, challenges, targets, and verification-key context while binding them to the same canonical source remains a separate verification problem.

This work therefore does not introduce a new CMC or proof-composition primitive. Instead, it retains the existing preprocessing and native verification paths and focuses on binding their accepted authenticated artifacts to a single

canonical-source relation. We specify a verifier-aligned public-input transfer relation and evaluate its implementation using positions and targets reconstructed from the authenticated context.

3 Canonical-Source Binding Construction

3.1 Public Statement and Canonical Input

Source binding is defined with respect to a canonical integer vector D of length N . Although the preprocessing and native computation paths use different representations and layouts, every value used by these paths must correspond to a specific position in this canonical vector. The partitioning and coefficient range used in the implementation are defined in Eq. (1).

$$\begin{aligned} D &= (D_0, \dots, D_{N-1}) \in \{0, \dots, t-1\}^N, \\ N &= Bn, \quad n = Ch, \\ (B, n, C, h, t) &= (128, 4096, 32, 128, 786433). \end{aligned} \tag{1}$$

In Eq. (1), B is the number of blocks, n is the number of coefficients per block, C is the number of chunks per block, and h is the number of coefficients per chunk. This two-level partition maps positions in the full canonical vector to both native blocks and chunk arguments. The relation $N = Bn$ divides the full input into 128 native blocks, while $n = Ch$ further divides each block into 32 chunks. The source value at position i of chunk c in block b is therefore $d_{b,c,i} = D_{bn+ch+i}$. This indexing fixes the position in D verified by each chunk and gives non-overlapping coverage of the full canonical vector.

The public statement I contains the HE parameters, per-stage statements, proofs and roots, an ordered source-commitment list, approved verification-key identifiers, and chunk public inputs. The private witness W contains the canonical source D , sign information, masks, salts, and quotients. Merkle openings are treated as public authentication artifacts. This separation keeps the source values and masks in the witness while allowing the verifier to check consistency between the authenticated upstream context and the public inputs of the chunk arguments. Table 2 summarizes the notation used in the source-binding relation.

Figure 2 shows the overall data flow among the prover, public authenticated artifacts, upstream verifiers, and joint verifier. The verifier does not read the source file directly. Instead, it reconstructs chunk-level inputs from accepted upstream proofs and public openings. V_q checks the preprocessing and digit chains, V_n checks native CMC, V_f checks the forward relation for the same native statement, and V_b checks chunk commitments and weighted-sum relations. Joint acceptance requires both acceptance by all four verifiers and consistency of the public inputs transferred between them.

Table 2. Main notation and implementation parameters. q_0 names the preprocessing stage, whereas p_ℓ and r are moduli serving different purposes.

Symbol	Meaning or value
D, N	Canonical coefficient vector and length 524, 288
B, n, C, h	128 blocks, 4, 096 coeff./block, 32 chunks/block, 128 coeff./chunk
t, θ	Canonical range $t = 786433$, sign threshold $\theta = 393217$
p_0, p_1	Native primes 34352398337 and 33015463937
r	BLS12-381 scalar-field prime used by the Groth16 circuit
m, R	340 authenticated rows/limb and 6 weighted-sum rounds
$\pi(b)$	Source-to-native block permutation
$\mu_{\ell,b,i}$	Private native-mask coefficient for limb ℓ
$w_{\ell,k,b,i}$	Weight rederived from authenticated positions and challenges
$P_{\ell,k,b,c}$	Public partial sum for a chunk

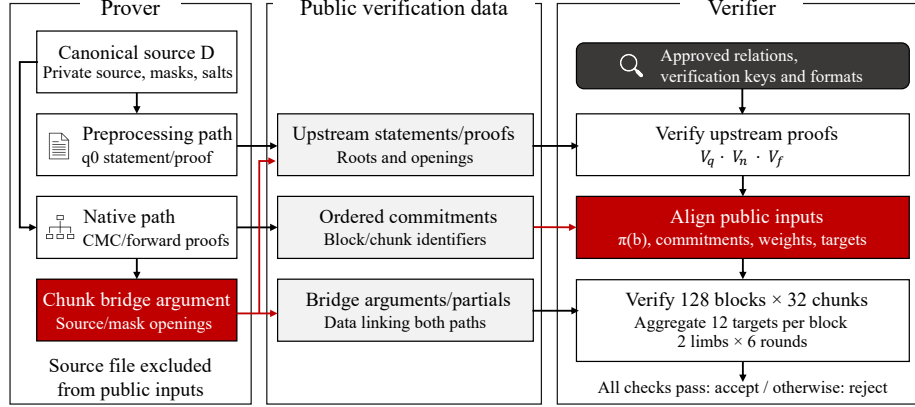


Fig. 2. Data flow between proof generation and verification. The verifier checks upstream proofs using approved keys, aligns indices, commitments, weights, and targets, and then checks all chunks and block targets. The 12 targets correspond to six rounds for each of the two limbs in a block.

Upstream Verifier Interfaces. Joint verification uses the acceptance results and exported authenticated artifacts of the existing upstream verifiers without reimplementing them. V_q takes the preprocessing FRI statement, proof and root, the ordered source commitments $C_{b,c}^D$, two digit chains, and an approved verification key. It checks FRI acceptance, the digit-chain chunks, and their mask endpoints, and links each digit's global target to the same ordered commitment list. Each chain contains 2,048 source chunks and one endpoint, giving 4,098 proofs in total. These proofs are distinct from the 4,096 native-bridge chunks used later. After acceptance, V_q exports the preprocessing root, the order of the source commitments, and authenticated positions and targets.

V_n takes the native CMC statement, proof and root, first message, and aggregate. It verifies Merkle- and FRI-authenticated rows in both RNS limbs and exports $(x_{\ell,j}, y_{\ell,b,j})$ for subsequent public-target reconstruction. This interface binds the targets used by the chunk arguments to positions and values accepted by the native verification path rather than to independently supplied values.

V_f takes the same native statement and a quotient-only strict-forward proof. It verifies the following forward relation over authenticated polynomials Q , D , Y , and H .

$$(QD) - Y = (X^{4096} + 1)H.$$

This equation expresses the native-block relation $QD \equiv Y \pmod{X^{4096} + 1}$ as a polynomial equality. The polynomial H is the quotient of $QD - Y$ by $X^{4096} + 1$. Verifying this equality checks that the authenticated Q , D , and Y satisfy the same forward-computation relation. The degree parameter 4096 corresponds to the number of coefficients n in each native block defined in Eq. (1). Acceptance by V_f alone does not bind the chunk public inputs to the same context. The joint verifier therefore separately checks that the positions and targets reconstructed from the accepted upstream context match the public inputs of the chunk arguments.

3.2 Index Mapping and Canonical Representation

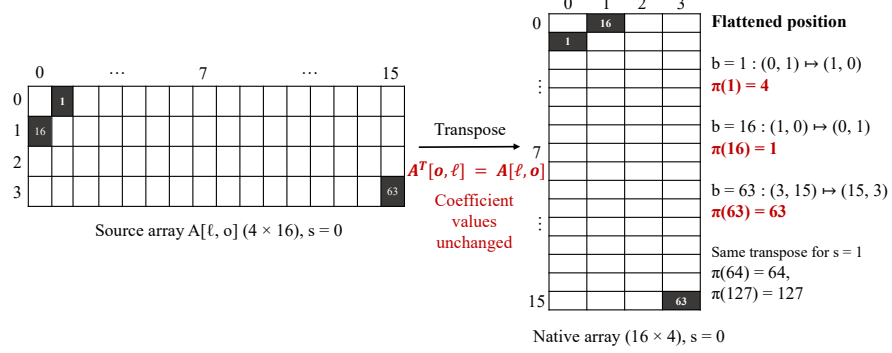
Source binding requires both a unique position mapping across verification paths and a canonical interpretation of the value at each position. In the preprocessing path, block b is assigned to digit $b \bmod 2$, and its position within that digit is $\lfloor b/2 \rfloor$. Independently, the native path uses a different block layout in which the leaf and output indices are transposed while the coefficient order within each block is preserved. The native block mapping is defined in Eq. (2).

$$\begin{aligned} s_b &= \lfloor b/64 \rfloor, & u_b &= b \bmod 64, \\ \ell_b &= \lfloor u_b/16 \rfloor, & o_b &= u_b \bmod 16, \\ \pi(b) &= 64s_b + 4o_b + \ell_b. \end{aligned} \tag{2}$$

In Eq. (2), the source block identifier $b = 64s_b + 16\ell_b + o_b$ is mapped to $64s_b + 4o_b + \ell_b$ for the same index components. This mapping transposes each 4×16 block array to 16×4 and then flattens it in row-major order. Figure 3 illustrates this native transpose and distinguishes it from the preprocessing digit split. It abbreviates s_b, ℓ_b, o_b as s, ℓ, o . For example, $b = 1$ maps to native polynomial 4, whereas $b = 16$ maps to polynomial 1. Directly using the source block number as the native column would therefore mislink both cases.

Lemma 1 (Bijective block mapping). *For $0 \leq b < 128$, π in Eq. (2) is a permutation of $\{0, \dots, 127\}$. The even- and odd-block lists of the two digits also cover all blocks without duplication.*

(a) Native mapping: unchanged elements, reordered traversal



(b) q0 mapping: digit and within-digit block position

$$\text{digit}(b) = b \bmod 2 \quad \text{offset}(b) = \lfloor b/2 \rfloor$$

Example: $b = 16 \rightarrow \text{digit } 0, \text{offset } 8$

Fig. 3. Two mappings of block identifiers. (a) The native path transposes the leaf-output array to locate authenticated columns. Shaded numbers are source block identifiers, and coefficient values do not change. (b) The even-odd digit split in q0 is a separate mapping.

Proof. Each b has a unique representation with $s_b \in \{0, 1\}$, $\ell_b \in \{0, 1, 2, 3\}$, and $o_b \in \{0, \dots, 15\}$. The components s_b , o_b , and ℓ_b can be recovered from $\pi(b)$ using quotients and remainders with respect to 64 and 4, so the mapping is bijective. The digit split partitions the block indices by parity, and the two sets are disjoint and cover the full range. Combining this mapping with the within-chunk positions in Eq. (1) therefore uniquely identifies every coefficient position.

Once the position is fixed, its value must also have a unique interpretation across native moduli. The sign and native representation of a canonical coefficient d are defined in Eq. (3).

$$\sigma(d) = \begin{cases} 0, & 0 \leq d < \theta, \\ 1, & \theta \leq d < t, \end{cases} \quad \theta = (t + 1)/2, \quad (3)$$

$$\tilde{d} = d - t\sigma(d), \quad L_\ell(d) = d + (p_\ell - t)\sigma(d).$$

In Eq. (3), $\tilde{d}$ is the signed representative in $[-393216, 393216]$, and $L_\ell(d)$ is its residue in $[0, p_\ell]$. When $\sigma(d) = 0$, $L_\ell(d) = d$. When $\sigma(d) = 1$, the signed representative is $\tilde{d} = d - t < 0$, and $L_\ell(d) = d + (p_\ell - t)$ represents the same value modulo p_ℓ . Thus, $L_\ell(d) \equiv \tilde{d} \pmod{p_\ell}$ in both cases, and both native limbs use the same $\sigma(d)$.

The circuit enforces the canonical range using 20-bit decompositions of d and $t - 1 - d$. Since $t = 786433 < 2^{20}$, 20 bits are sufficient to represent the required

nonnegative range. The circuit also requires a Boolean sign and nonnegative values for $d - \theta\sigma$ and $\theta - 1 + (t - \theta)\sigma - d$. Hence, $\sigma = 0$ forces $d < \theta$, whereas $\sigma = 1$ forces $d \geq \theta$. The two intervals are disjoint, which fixes a unique sign for every canonical coefficient. In particular, $d = 393216$ and $d = 393217$ lie on opposite sides of the boundary.

3.3 Chunk Commitments to the Source and Masks

Domain-separated MiMC digests bind each chunk's source and two masks [1]. Source tag 1 is shared by preprocessing and the native bridge. Mask tags 22 and 23 distinguish the two limbs. The commitments are defined as follows.

$$\begin{aligned} C_{b,c}^D &= H_1(b, c, \rho_{b,c}^D, d_{b,c,0}, \dots, d_{b,c,h-1}), \\ C_{\ell,b,c}^\mu &= H_{22+\ell}(b, c, \rho_{\ell,b,c}^\mu, \mu_{\ell,b,ch}, \dots, \mu_{\ell,b,ch+h-1}). \end{aligned} \quad (4)$$

In Eq. (4), H_a serializes MiMC inputs with tag a first, and ρ is a private salt. Including positions binds an opening to its block and chunk. The verifier compares source digests with the ordered preprocessing list, while the chunk argument checks their openings.

Masks satisfy $0 \leq \mu_{\ell,b,i} < p_\ell$. Native computation uses the residue of $L_\ell(d) + \mu$. For the chunk relation, the circuit instead accumulates the congruent non-negative integer $d + (p_\ell - t)\sigma + \mu$. It then decomposes the weighted sum into a quotient and partial sum, as specified in the next subsection.

3.4 Reconstructing Public Targets from Authenticated Rows

The public verifier reconstructs weights and targets from accepted rows and positions, tying the chunk relation to authenticated HE artifacts. For native limb ℓ , write these data as $(x_{\ell,j}, y_{\ell,b,j})$, where $j = 0, \dots, m-1$ and $m = 340$. The evaluation points use the authenticated positions, shift, and transform root.

For block b , limb ℓ , and round k , domain-separated SHAKE256 derives the challenge [18]. Its input contains the native auxiliary statement, preprocessing and native master values, hashes of the chunk commitment list and native proof, and (b, ℓ, k) . The resulting $\alpha_{\ell,k,b}$ lies in $\{1, \dots, p_\ell - 1\}$, and its powers combine the rows. We define weights and targets as follows.

$$\begin{aligned} w_{\ell,k,b,i} &= \sum_{j=0}^{m-1} \alpha_{\ell,k,b}^j x_{\ell,j}^i \pmod{p_\ell}, \\ T_{\ell,k,b} &= \sum_{j=0}^{m-1} \alpha_{\ell,k,b}^j y_{\ell,b,j} \pmod{p_\ell}. \end{aligned} \quad (5)$$

In Eq. (5), i ranges over positions $0, \dots, 4095$ within a block, and $k = 0, \dots, 5$. Each block thus has 12 targets across two limbs. The verifier checks that each limb has 340 distinct positions. Challenges and weights follow the authenticated row order.

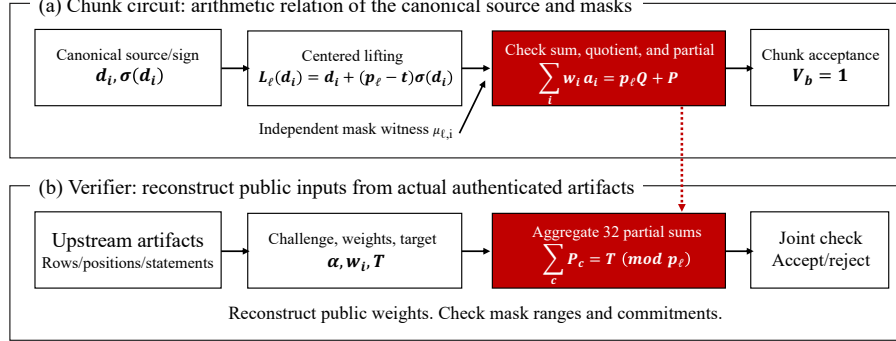


Fig. 4. Connection between chunk circuits and public-input reconstruction. Masks are witnesses independent of centered representatives, and partial sums are aggregated by modular congruence with the block target. The expression $a_i = L_\ell(d_i) + \mu_{\ell,i}$ is an integer representation before modular reduction. Block and round subscripts are omitted.

The chunk circuit uses the weights in Eq. (5) with a common source, signs, and two masks. The circuit equality and verifier-side block aggregation are defined as follows.

$$\begin{aligned}
 \sum_{i=0}^{h-1} w_{\ell,k,b, ch+i} (d_{b,c,i} + (p_\ell - t)\sigma(d_{b,c,i}) + \mu_{\ell,b, ch+i}) &= p_\ell Q_{\ell,k,b,c} + P_{\ell,k,b,c}, \\
 0 \leq P_{\ell,k,b,c} < p_\ell, \quad 0 \leq Q_{\ell,k,b,c} < 2^{48}, \\
 \sum_{c=0}^{C-1} P_{\ell,k,b,c} &\equiv T_{\ell,k,b} \pmod{p_\ell}.
 \end{aligned} \tag{6}$$

The circuit checks the first equality in Eq. (6) over $\mathbb{F}_r$, where P is a public partial sum and Q a private quotient. The external verifier reconstructs weights in $[0, p_\ell)$ and checks the block aggregate. The integer interpretation requires both circuit range checks and this validation of public inputs.

Lemma 2 (Integer interpretation of chunk arithmetic). *Under the range and sign constraints in Eq. (3), the mask and partial-sum ranges, the 48-bit quotient constraint, and verifier-reconstructed weights, the first equality in Eq. (6) is an integer equality without scalar-field wrap-around.*

Proof. We have $p_\ell < 2^{35}$, and each accumulated value is less than $2p_\ell$. Each weight is also less than p_ℓ , so the sum of 128 terms is bounded by $256p_\ell^2 < 2^{78}$. The right-hand side is bounded by $p_\ell 2^{48} + p_\ell < 2^{84}$. Both bounds are below the approximately 255-bit BLS12-381 scalar-field prime r . Equality in the field therefore implies equality over the integers. This conclusion does not cover replacing public weights with arbitrary scalar-field elements.

3.5 Joint Relation and Verification Procedure

The source-binding relation $\mathcal{R}_{\text{bind}}(I, W)$ combines four conditions. First, V_q , V_n , and V_f accept their specified statements and proofs. Second, positions follow Eqs. (1) and (2), with source and mask openings for each (b, c) satisfying Eq. (4). Third, Eqs. (3) and (6) hold for inputs reconstructed from V_n 's authenticated rows. Fourth, counts, order, digit coverage, and approved keys and formats agree with I .

Proposition 1 (Conditional Canonical-Source Uniqueness). *Assume that V_q, V_n, V_f are sound for the interfaces above, that H_a is binding for messages containing positions, and that V_b is knowledge-sound under a valid Groth16 CRS. Also assume that statements, roots, the ordered commitment list, and verification keys are fixed before challenge derivation. If Algorithm 1 accepts, all chunks uniquely assemble into one $D^* \in \{0, \dots, t-1\}^N$, and the accepted q0 commitment list and native-bridge chunks refer to the same D^* .*

Proof. Knowledge soundness of V_b allows extraction of the source opening for each (b, c) . Binding of H_1 , which includes (b, c) , excludes distinct canonical openings of the same $C_{b,c}^D$. Complete $B \times C$ coverage and the bijective indices of Lemma 1 fix these chunks as one D^* without duplication. V_q links the same ordered $C_{b,c}^D$ to the q0 context, while V_b links them to targets reconstructed from V_n . V_f binds the same native statement to the forward relation. Accepting a different canonical source linked to the same public statement would therefore require one of the assumptions to fail.

The proposition establishes conditional uniqueness of the canonical vector assembled through the accepted interfaces, not merely equality of digest identifiers. Component soundness remains an assumption. Concrete failure bounds and setup and Fiat–Shamir conditions are addressed in Sect. 3.7.

In Algorithm 1, $\mathcal{K}$ contains approved verification keys and format information for each relation. Final acceptance requires checking all chunk counts, their order, and every verification result.

Rows authenticated under one root are parsed once and reused across blocks. Indices and (b, ℓ, k) challenges remain block-specific, so each block still requires weight reconstruction and chunk verification. Changes to the root, auxiliary statement, proof, or format change the authenticated context. The C++ helper only computes public modular arithmetic and does not replace upstream authentication.

3.6 Direct Joint Circuit and Redundancy Elimination

G is the full-input shared path, whereas D and F are representative-block circuits for a separate same-relation comparison. G checks that arguments for the same (b, c) open the same source digest. D directly shares source variables between preprocessing and both native relations. It checks source and two centered-representative commitments, native source and mask commitments, and preprocessing and native partial sums. Its targets come from the masked preprocessing root referenced by the native statement and the existing native root.

Algorithm 1 Verifier-aligned public joint verification

 Input: Public statement–proof bundle I , approved key–format list $\mathcal{K}$

 Output: **accept** or **reject**

- 1: Check formats, parameters, key identifiers, and the ordered commitment list.
 - 2: **if** keys or formats differ from $\mathcal{K}$, or the list structure is invalid **then**
 - 3: **return reject**
 - 4: **end if**
 - 5: Check actual upstream proofs and statements with V_q , V_n , and V_f .
 - 6: **if** any upstream verification rejects **then**
 - 7: **return reject**
 - 8: **end if**
 - 9: Extract authenticated rows and positions, and fix the context of each root.
 - 10: **for** $b = 0, \dots, B - 1$ **do**
 - 11: Reconstruct w, T in Eq. (5) from authenticated column $\pi(b)$.
 - 12: **for** $c = 0, \dots, C - 1$ **do**
 - 13: Compare source and mask commitments and (b, c) .
 - 14: Verify the chunk proof with the approved key and output **reject** on failure.
 - 15: **end for**
 - 16: Check the 12 block targets in Eq. (6).
 - 17: **end for**
 - 18: Check both digits’ global targets, chunk counts, and complete range coverage.
 - 19: **return accept**
-

F removes D’s duplicate raw hashing, range and sign checks, and centered lifting by performing them once with shared source variables and salts. Both mask digests and the distinct weighted-sum relations remain. Thus the optimization shares computation while retaining the verification targets.

Proposition 2 (Sharing canonical-conversion constraints). *If the same d and $\sigma(d)$ are already enforced and $p_\ell > t$, then $L_\ell(d) = d + (p_\ell - t)\sigma(d)$ is uniquely determined in $[0, p_\ell)$. This expression can therefore be shared instead of treating L_ℓ as an independent witness and checking its range again.*

Proof. For $\sigma = 0$, $L_\ell = d < \theta < p_\ell$. For $\sigma = 1$, $\theta \leq d < t$, so $p_\ell - t + \theta \leq L_\ell \leq p_\ell - 1$. In both cases, the range holds and d determines the value. This substitution does not remove checks on other digest inputs or weighted-sum targets.

This proposition justifies the local sharing of range and sign conversions. It is not an equivalence proof for the complete circuits. Section 4.6 evaluates D and F under the same roots and public relation.

3.7 Composed Soundness and Scope of Public Information

A concrete failure bound for the conditional source-uniqueness proposition in Sect. 3.5 requires a common adversarial model. The terms are upstream relation

failures $\epsilon_q, \epsilon_n, \epsilon_f$, commitment-binding failure ϵ_H , chunk-argument knowledge-soundness failure ϵ_b , and non-interactive challenge-fixing failure ϵ_{FS} . The required union bound has the following form.

$$\epsilon_{\text{bind}} \leq \epsilon_q + \epsilon_n + \epsilon_f + \epsilon_H + 4096\epsilon_b + \epsilon_{FS}. \quad (7)$$

Equation (7) specifies failure terms, not a 128-bit security estimate. In this interface-level accounting, the 4,096 chunk terms concern the native bridge, while the q_0 term must cover the complete V_q contract, including both digit chains. All terms require bounds in the same adaptive model, with FRI/IOP extraction and Fiat–Shamir commitment-order conditions [4,5,15].

Groth16 requires a valid CRS for the intended circuit. Checking VK bytes establishes key agreement, not an independent setup ceremony or disposal of toxic waste.

HE statements and proofs, authenticated rows, commitment lists, and native partial sums remain public. We therefore do not claim whole-transcript zero knowledge or witness hiding. The result is conditional canonical-source uniqueness and an implementation of its public checks on authenticated artifacts.

4 Implementation and Evaluation

4.1 Configurations and Measurement Boundaries

The evaluation covers full-input execution, public-input reconstruction, specified mutation tests, and computation and storage costs. The paths in Sect. 3.6 have different evaluation scopes. G covers all 128 blocks, while D/F compare constraint sharing on block 0 under the same two authenticated roots and public relation. Table 3 summarizes the evaluation configurations, execution scope, and measurement environment.

Table 3. Evaluated configurations and implementation conditions. Dependency versions and runtime resource settings are listed separately.

Item	Configuration or confirmed condition
Config. G	128 blocks, 524,288 coeff., 4,096 bridge chunks
Config. D/F	4,096 coeff. and 32 chunks from block 0 each
Upstream	Masked q_0 FRI, native CMC, strict-forward
Chunk proof	Groth16 over BLS12-381 with MiMC
Implementation	Go/gnark circuits, Python orchestration, C++ public arithmetic
Dependencies	Go 1.25.7, gnark 0.16.3, gnark-crypto 0.21.0
Runtime	G/D/F: GOMAXPROCS=2, GOMEMLIMIT=2GiB
Measurement	macOS /usr/bin/time -l, file-access sandbox
Statistics	Recorded runs only, no repeated-run distribution or CIs

Chunk arguments are implemented using gnark [12]. Measurements were performed on an Apple M2 MacBook Air with 8 GiB of memory and macOS 15.7.1. `GOMAXPROCS=2` limits concurrent Go execution to two units. `GOMEMLIMIT=2GiB` is a memory-management hint rather than an RSS ceiling. Blockwise wall-clock time and peak RSS were recorded using `/usr/bin/time -l` and runner logs.

For G, blockwise time includes generation followed by verification. For D/F, proving time is the sum of `groth16.Prove` calls over 32 distinct chunks, not 32 repeated executions of the same input. Because repeated-run distributions were not preserved, means, variances, and confidence intervals are not reported. D/F use the same hardware, input size, authenticated roots, and public relation, and differ in whether duplicate circuit constraints are shared. Setup cost is measured separately. Chunk-verification time excludes upstream FRI verification and challenge reconstruction. MB and GB denote 10^6 B and 10^9 B, respectively.

4.2 Full 128-Block Results and Generation Cost

The G experiment evaluates whether the proposed source-binding procedure operates correctly at full-input scale and where the main execution cost occurs. It processes 524,288 canonical coefficients across 128 blocks and generates and verifies 32 native-bridge chunks per block, for a total of 4,096 chunks. Table 4 reports the acceptance results together with execution time and memory cost for the full run.

Table 4. Full-input results for G. The generation path excludes initial generation of upstream HE proofs and the cost of separate output proofs.

Category	Metric	Recorded value
Coverage	Canonical coefficients	524,288
	Native-bridge blocks / chunks	128/128 / 4,096/4,096
	Preprocessing chunk/endpoint proofs	4,098
Bridge data	Native-bridge proof bodies	999,424 B
	Native partial sums	1,101,841 B
Execution	Blockwise generation + verification	22,982.05 s
	Runner wall-clock	23,723.31 s
	Max. blockwise peak RSS	998,014,976 B
Verification	Joint verification with reconstruction	184.77 s

All 128 blocks and 4,096 chunks were accepted in the full execution. The total proof-body size was 999,424 B, the runner took approximately 6.59 hours, and the maximum blockwise peak RSS was approximately 0.998 GB.

Blockwise generation and verification account for 96.88% of the total runner time, while processing outside the blocks accounts for 3.12%. This indicates that repeated block-level computation, rather than orchestration alone, is the main

target for reducing execution time. The subsequent analysis therefore focuses on circuit-level redundancy and block-processing cost. This timing includes both proof generation and verification and excludes the initial generation of upstream proofs, so it should not be interpreted as an end-to-end prover-to-verifier cost ratio.

4.3 Public Reconstruction and Reuse of Verification Work

Public reconstruction was evaluated under conditions that exclude stored challenge caches and duplicate commitment files. The verifier reconstructs the inputs and weights for all 128 blocks from the preprocessing bundle, native first message, and authenticated rows, while also checking the approved verification keys. Reads of challenge caches, duplicate commitments, and unused forward proofs were prohibited, and a quotient-only proof for the same native statement was used instead. Common authenticated artifacts were reused, while per-block weight reconstruction and all 4,096 chunk checks were retained.

Under these conditions, public verification of preprocessing, native CMC, the forward relation, and all bridge chunks completed in 184.77 seconds. In addition, 74,713,951 B of reconstructible duplicate files were excluded while preserving every per-block weight reconstruction and chunk check. This confirms that these duplicate files are not required for verification acceptance and that the required public inputs can instead be reconstructed from authenticated upstream artifacts.

The result separates stored intermediate data from the evidence required for public verification. Rather than trusting cached challenges or duplicate input representations, the verifier derives the required weights and targets from accepted roots and authenticated positions and then performs the same chunk checks. The excluded 74,713,951 B therefore represents removable duplication in public-input storage rather than a reduction in proof-body size. This distinction shows that storage redundancy can be reduced without removing the provenance and context checks required by the source-binding procedure. Because no repeated measurements of the cached path were preserved, the result does not establish a verification-speed improvement or whole-transcript hiding.

4.4 Dominant Components of Public-Data Size

This evaluation analyzes which components dominate the total public-data size and how much the overall size can be reduced by optimizing only the native-bridge proof bodies. Table 5 reports the explicit data used by G’s public reconstruction path, including the required partial sums, commitments, and verification keys while excluding reconstructible caches. Figure 5 visualizes the corresponding size distribution and highlights the components that dominate the public-data footprint.

The total explicit public-data size is 27,345,153 B. The native CMC proof accounts for 39.44%, the two-digit preprocessing bundle for 26.91%, and the quotient-only forward proof for 22.09%. Together, these three artifacts occupy

Table 5. Explicit public verification data for G. Percentages use the total of 27,345,153 B as the denominator.

Category	Component	Size (B)	Share (%)
Upstream artifacts	Two-digit preprocessing bundle	7,359,112	26.91
	Native CMC proof	10,784,832	39.44
	Native statement	216	< 0.01
	Quotient-only forward proof	6,041,176	22.09
Verification context	Native first aggregate	982,732	3.59
	Native first message	784	< 0.01
	Shared verification key	75,036	0.27
Bridge artifacts	Native-bridge Groth16 proof bodies	999,424	3.65
	Native public partial sums	1,101,841	4.03
Total	Explicit public components	27,345,153	100.00

Public verification data by component

Total: 27,345,153 B

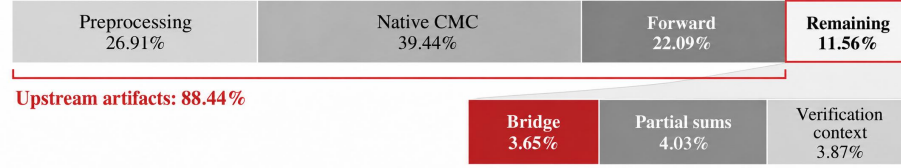


Fig. 5. Composition of G’s public verification data. The three upstream proof artifacts account for 88.44%, and native-bridge proof bodies account for 3.65%. Values for short segments are shown in the legend outside the bar.

24,185,120 B, or 88.44% of the total. In contrast, native-bridge proof bodies account for 3.65%, while public partial sums account for 4.03%. These results show that the dominant public-data cost lies in the upstream proof artifacts rather than in the chunk proof bodies.

To quantify the effect of reducing only the native-bridge proof bodies, let S denote the total public-data size and let $f = 0.0365485$ denote the fraction occupied by these proof bodies. Holding all other components fixed, reducing the proof-body size by a factor of κ gives the total size in Eq. (8).

$$S'(\kappa) = S \left((1 - f) + \frac{f}{\kappa} \right), \quad \kappa \geq 1. \quad (8)$$

Equation (8) shows that halving the native-bridge proof bodies reduces the total public-data size by only approximately 1.83%. Even eliminating these proof bodies entirely would limit the reduction to 3.65%. This indicates that optimizing circuit or chunk-proof size alone has limited impact on the overall public-data footprint. Larger reductions therefore require optimization of the upstream au-

thentication artifacts, particularly the native CMC, preprocessing, and forward proofs.

4.5 Specified Mutations and Cross-Linking Checks

To evaluate whether the verification path checks the full input range and preserves the accepted upstream context, specified mutations were applied to each configuration. In G, the final block was modified to test whether the verifier could accept the full input after checking only a prefix. In D/F, roots and challenges were substituted to test whether chunk public inputs remained bound to the accepted upstream artifacts. Table 6 summarizes the mutation types and rejection results for each configuration.

Table 6. Specified mutation tests by configuration. Counts report rejected cases among the specified test cases, not samples from a general attack distribution.

Config.	Main test targets	Rejected cases
G	Final-block proof, source and both mask commitments, chunk order, partial sums, targets, and key	8 of 8
D	Proofs, keys, commitments, targets, and order, plus challenge substitution for both actual roots	14 of 14
F	Public proofs, keys, digests, targets, and order, plus fused-circuit challenge changes	13 of 13
D / F	Local constraints on representative-chunk source, signs, masks, partial sums, and related values	7 of 7 each

G rejected all specified final-block mutations, confirming that the tested verification path includes the end of the full input range. D/F also rejected the specified local mutations involving roots and challenges both before and after redundancy elimination. These results show that the tested linkage between chunk inputs and the upstream authenticated context was preserved after constraint sharing. Thus, the evaluated source-binding checks remained active while duplicate circuit computations were removed.

Supplementary Actual-Root Alignment Audit. A supplementary interface audit evaluated whether public inputs reconstructed from actual authenticated roots were transferred consistently to chunk verification. For block 0, 74,240 root-determined fields were reconstructed from public openings without the source, salts, or masks. All 32/32 Groth16 proofs, 31/31 boundary relations, and 12/12 native targets were accepted, while mutations to proof order, boundaries, an old verification key, ranges, signs, and indices were rejected. For block 1 in the other digit, all 32/32 Groth16 proofs and 12/12 native targets were accepted, while all 10/10 global mutations and 7/7 joint mutations were rejected.

These observations show that inputs reconstructed from authenticated roots reach chunk verification consistently and that the specified context and index mismatches are detected. The results are limited to the defined mutation sets and implementation paths and do not establish full-circuit equivalence or a composed soundness probability against general adversaries.

4.6 Circuit and Setup Costs for the Same Relation

The D/F comparison evaluates the cost effect of sharing duplicate circuit constraints while fixing the same two authenticated roots, 4,096 canonical coefficients, public-input format, and 32 chunks. This isolates the effect of removing duplicate source-hash, range, and sign computations from changes in input size or verification targets. Table 7 reports the constraint count, setup cost, proving time, key size, memory use, and verification cost for D and F.

Table 7. Direct joint circuit D and fused circuit F for block 0. Setup runs separately from the proving runner. The chunk-verification runner excludes upstream FRI verification and challenge reconstruction.

Category	Metric	D	F	Reduction
Circuit	R1CS constraints/chunk	380,895	253,178	33.53%
Setup	Setup time (s)	102.015	64.605	36.67%
	Proving key (MB)	111.917	70.083	37.38%
	Verification key (B)	112,572	112,572	—
Proving	32-proof proving time (s)	244.695	168.606	31.10%
	Proving runner (s)	348.227	235.791	32.29%
	Peak RSS (MB)	1,272.398	1,070.105	15.90%
Proof size	32 proof bodies (B)	7,808	7,808	—
Verification	Chunk-verification runner (s)	0.807	0.830	—

—: no reduction reported.

F reduces the number of constraints per chunk from 380,895 to 253,178, a reduction of 33.53%. The proving time for 32 chunks decreases from 244.695 to 168.606 seconds, or 31.10%, while the full proving-runner time decreases from 348.227 to 235.791 seconds, or 32.29%. Setup time also decreases from 102.015 to 64.605 seconds, or 36.67%. These results show that sharing duplicate circuit computations can directly reduce prover-side computation while preserving the same public relation.

When an approved key for the same circuit is reused, setup cost can be amortized across repeated workloads. For L repetitions of the 32-chunk workload, the per-workload cost is expressed in Eq. (9).

$$\bar{\tau}_X(L) = \frac{\tau_{\text{setup},X}}{L} + \tau_{\text{run},X}, \quad X \in \{D, F\}. \quad (9)$$

As L increases in Eq. (9), the contribution of setup cost decreases, and the D/F reduction approaches the runner-cost reduction of approximately 32.29%. This indicates that the measured benefit is not limited to setup reuse and also arises from lower computation in the repeated proving path.

Storage and memory costs show a similar but not identical trend. The proving key decreases from approximately 111.92 MB to 70.08 MB, a reduction of 37.38%, while peak RSS decreases from approximately 1,272.40 MB to 1,070.10 MB, or 15.90%. In contrast, the total proof-body size for 32 chunks remains 7,808 B, and the verification key remains 112,572 B. These results indicate that constraint sharing primarily benefits prover computation and proving-key storage rather than proof-body size or verification cost. Because repeated-run distributions were not preserved, the 0.807- and 0.830-second verification times are not used to claim a verification-speed improvement.

5 Conclusion

This paper proposed a verifier-aligned public-input transfer procedure to determine whether separately accepted preprocessing and native HE verification results refer to the same canonical source. The public verifier reconstructs the required inputs while preserving the accepted roots, positions, source and mask commitments, authenticated targets, and verification-key context. Chunk arguments then verify canonical ranges, signs, commitment openings, and partial-sum relations. This separates individual proof validity from cross-path source consistency and gives a conditional uniqueness relation for the canonical source under the stated soundness, commitment-binding, knowledge-soundness, and challenge-order assumptions.

The full-input evaluation processed 524,288 coefficients across 128 blocks and 4,096 native-bridge chunks, with public reconstruction and verification taking 184.77 seconds. In the same-relation comparison on a representative block, sharing duplicate hash, range, and sign computations reduced the constraint count by 33.53% and proving time by 31.10%, while the proof-body size remained unchanged. Among the 27,345,153 B of explicit public data, the native CMC, preprocessing, and forward proofs together accounted for 88.44%. These results show that circuit-level redundancy elimination directly reduces prover computation, whereas the dominant public-data cost remains in the upstream proof artifacts. Prover computation and public-data size should therefore be treated as distinct optimization targets.

However, the evaluation is limited to recorded full-input execution in a single environment and a same-relation comparison on a representative block, without repeated-run distributions, whole-transcript hiding analysis, or a concrete composed-security bound. Future work will extend the evaluation to composed-security analysis and transcript leakage while reducing the dominant upstream proof artifacts.

References

1. Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. *Cryptology ePrint Archive*, Paper 2016/492 (2016), <https://eprint.iacr.org/2016/492>
2. Aranha, D.F., Costache, A., Guimarães, A., Soria-Vazquez, E.: HELIOPOLIS: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. In: *Advances in Cryptology – ASIACRYPT 2024. Lecture Notes in Computer Science*, vol. 15488, pp. 302–334. Springer (2024). https://doi.org/10.1007/978-981-96-0935-2_10
3. Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A full RNS variant of FV like somewhat homomorphic encryption schemes. In: *Selected Areas in Cryptography – SAC 2016. Lecture Notes in Computer Science*, vol. 10532, pp. 423–442. Springer (2017). https://doi.org/10.1007/978-3-319-69453-5_23
4. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast Reed–Solomon interactive oracle proofs of proximity. In: *45th International Colloquium on Automata, Languages, and Programming. LIPIcs*, vol. 107, pp. 14:1–14:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2018). <https://doi.org/10.4230/LIPIcs.ICALP.2018.14>
5. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: *Theory of Cryptography – TCC 2016-B, Part II*. pp. 31–60. Springer (2016). https://doi.org/10.1007/978-3-662-53644-5_2
6. Ben-Sasson, E., Goldberg, L., Kopparty, S., Saraf, S.: DEEP-FRI: Sampling outside the box improves soundness. In: *11th Innovations in Theoretical Computer Science Conference. LIPIcs*, vol. 151, pp. 5:1–5:32. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2020). <https://doi.org/10.4230/LIPIcs.ITCS.2020.5>
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. *Cryptology ePrint Archive*, Paper 2011/277 (2011), <https://eprint.iacr.org/2011/277>
8. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In: *2019 ACM SIGSAC Conference on Computer and Communications Security. ACM* (2019). <https://doi.org/10.1145/3319535.3339820>
9. Cascudo, I., Costache, A., Cozzo, D., Fiore, D., Guimarães, A., Soria-Vazquez, E.: Verifiable computation for approximate homomorphic encryption schemes. In: *Advances in Cryptology – CRYPTO 2025*. pp. 643–677. Springer (2025). https://doi.org/10.1007/978-3-032-01907-3_21
10. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: *Selected Areas in Cryptography – SAC 2018. Lecture Notes in Computer Science*, vol. 11349, pp. 347–368. Springer (2019). https://doi.org/10.1007/978-3-030-10970-7_16
11. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. *Cryptology ePrint Archive*, Paper 2016/421 (2016), <https://eprint.iacr.org/2016/421>
12. Consensys: gnark: A fast zk-SNARK library. Software documentation (2026), <https://docs.gnark.consensys.io/>, accessed 25 September 2026
13. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, Paper 2012/144 (2012), <https://eprint.iacr.org/2012/144>

14. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: *Advances in Cryptology – CRYPTO ’86. Lecture Notes in Computer Science*, vol. 263, pp. 186–194. Springer (1987). https://doi.org/10.1007/3-540-47721-7_12
15. Groth, J.: On the size of pairing-based non-interactive arguments. In: *Advances in Cryptology – EUROCRYPT 2016. Lecture Notes in Computer Science*, vol. 9666, pp. 305–326. Springer (2016). https://doi.org/10.1007/978-3-662-49896-5_11
16. Jivanyan, A., Hovhannisyan, G., Hovhannisyan, H., Asatryan, N.: zkFFT: Extending Halo2 with vector commitments & more. *Cryptology ePrint Archive*, Paper 2024/1661 (2024), <https://eprint.iacr.org/2024/1661>
17. Liu, F., Liang, H., Xie, X., Yu, Y., Zheng, W., Hu, Y.: HasteBoots: Proving TFHE programmable bootstrapping in seconds. In: *35th USENIX Security Symposium*. pp. 41–60. USENIX Association (2026), <https://www.usenix.org/conference/usenixsecurity26/presentation/liu-fengrun>
18. National Institute of Standards and Technology: SHA-3 standard: Permutation-based hash and extendable-output functions. Tech. Rep. FIPS PUB 202, NIST (2015). <https://doi.org/10.6028/NIST.FIPS.202>
19. Wei, Y., Zhou, Z., Zhang, X., Wu, S., Xiang, B., Hong, C., Deng, Y.: GlueLUT: Generalized lookup table arguments over residue rings via auxiliary fields. *Cryptology ePrint Archive*, Paper 2026/494 (2026), <https://eprint.iacr.org/2026/494>
20. Zhang, X., Wang, R., Niu, Q., Liu, P., Lu, X., Zhao, L., Hou, R., Deng, Y.: Lasagne: Practical verifiable computation over encrypted data. *Cryptology ePrint Archive*, Paper 2026/857 (2026), <https://eprint.iacr.org/2026/857>
21. Zhou, Z., Li, Y., Yang, Z., Hong, C., Wei, T.: PRISM: Efficient zkSNARKs for RNS-based homomorphic encryption. *Cryptology ePrint Archive*, Paper 2026/1956 (2026). <https://doi.org/10.1145/3830454.3846611>