Phalanx: An FHE-Friendly SNARK for Verifiable Computation on Encrypted Data

Xinxuan Zhang
zhangxinxuan@iie.ac.cn
State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS
School of Cybersecurity, UCAS
Beijing, China

Ruida Wang*
wangruida@iie.ac.cn
State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS
School of Cybersecurity, UCAS
Beijing, China

Zeyu Liu
zeyu.liu@yale.edu
Yale University
New Haven, United States

Binwu Xiang†
bwxiang@sc.ecnu.edu.cn
East China Normal University
Shanghai, China

Yi Deng*
deng@iie.ac.cn
State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS
School of Cybersecurity, UCAS
Beijing, China

Ben Fisch
ben.fisch@yale.edu
Yale University
New Haven, United States

Xianhui Lu*
luxianhui@iie.ac.cn
State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS
School of Cybersecurity, UCAS
Beijing, China

Abstract

Verifiable Computation over encrypted data (VCoed) has two popular paradigms: *SNARK-FHE* (applying SNARKs to prove FHE operations) and *FHE-SNARK* (homomorphically evaluating SNARK proofs). For the existing works, FHE-SNARK has a much better efficiency compared to SNARK-FHE.

In this work, we follow the line of FHE-SNARK and further improve its efficiency by designing Phalanx—an FHE-friendly SNARK that is: a) 3× lower multiplicative depth than FRI-based SNARKs; and b) Compatible with FHE SIMD operations.

Based on Phalanx, we construct an FHE-SNARK scheme that has: a) **7.3**×∼ **24.4**× speedup: 2.27-hour proof generation for 2^{20}-gate circuits on a single core CPU and 0.68-hour when the input ciphertexts are in iNTT form (vs. 16.57 hours in the state-of-the-art); and b) *Practical verification*: 61.4 MB proofs with 2.8 seconds verification (single core).

Keywords

Verifiable Computation on Encrypted Data, FHE, SNARK

1 Introduction

Fully homomorphic encryption (FHE), first realized by Gentry's breakthrough work in 2009 [35], enables arbitrary computations on encrypted data without decryption. After a decade of advancements [9–12, 18, 20, 25, 26, 36], FHE demonstrates practical efficiency for diverse applications including secure machine learning [24, 43, 46, 47, 55], private information retrieval [2, 30, 52, 57], private set intersection [15, 16, 22], and oblivious message retrieval [48, 53, 54].

Despite its power, a critical limitation of FHE is the lack of computation integrity guarantees: A malicious server can arbitrarily alter computation results without knowing any information about the data. To address this issue, Gennaro, Gentry, and Parno [34] introduced the concept of Verifiable Computation over encrypted data (which we denote as VCoed for short; without loss of generality, verifiable computation in this paper specifically denote the variant over encrypted data[1]). Essentially, VCoed allows a client to outsource its computation to an untrusted server while (1) protecting the data privacy and (2) enabling the client to verify computation correctness. While the concept was introduced, the subsequent realization of VCoed in the next decade has limited capability and efficiency [28, 34], making them unsuitable for verifying FHE computations.

Recently, with the rapid development of SNARKs (Succinct Non-interactive Arguments of Knowledge), a new method has been proposed to construct VCoed specifically for FHE [4, 6, 29, 32, 62]. In particular, the server, while performing the homomorphic computation, generates a SNARK proof demonstrating that the computation (over the ciphertexts) has been executed correctly, which we call

*Corresponding authors: Ruida Wang, Yi Deng, Xianhui Lu.
†Work carried out while at State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS

[1] Note that while the concept of verifiable computation has roots in older definitions [63], which focuses on integrity without privacy, thus does not align our objectives.

SNARK-FHE, as SNARK is used outside FHE. The client can then efficiently verify the proof and ascertain whether the computation was conducted honestly. While this approach marks a significant improvement over previous works, it is still not practical for real-world applications. For instance, a real-world deployment by ZAMA demonstrates that generating a proof for TFHE gate bootstrapping takes approximately 20 minutes.

More recently, a new line of work has emerged [3, 31, 33], introducing an alternative approach to building VCoed for FHE. Specifically, after the server obtains the homomorphically evaluated result (encrypted under FHE), it then homomorphically generates a SNARK proof against the encrypted input and output, which we call *FHE-SNARK*, since FHE is used outside SNARK. In other words, the proof is generated against the plaintext inside FHE (instead of generated directly over ciphertexts as above) and is therefore also computed homomorphically via FHE. This approach is considerably more efficient. For instance, the proof time for a 2^{20}-gate circuit costs 9.26 hours in [31] (0.03 second per gate compared to 20 minutes in the previous method).

Although the prior works in the second line (FHE-SNARK) seems to provide a stepping stone towards practical VCoed, there remains substantial room for further enhancement. For example, previous works [3, 31] have opted for holography-IOP-based SNARKs using the FRI protocol. This approach introduces a large multiplicative depth and is not conducive to FHE SIMD (Single Instruction, Multiple Data) operations, both of which significantly impair the efficiency of FHE evaluation. This leads to a common question:

Is there an approach to design an FHE-friendly SNARK? More generally, can we develop a more efficient FHE-SNARK for verifiable computation?

1.1 Our Result

We address this question by proposing Phalanx, an FHE-friendly SNARK based on Spartan, and then achieving practical verifiable computation under FHE-SNARK paradigm.

FHE-friendly SNARK. We first construct a new SNARK where the statement and the witness are encrypted, achieving two key improvements:

- Low multiplication depth: Our construction significantly reduces the multiplication depth to 4, compared to 12+ in existing schemes.
- Compatibility with FHE SIMD operations: The construction fully harnesses FHE packing capabilities, enabling batched homomorphic evaluations.

Practical FHE-SNARK for verifiable computation. Based on the aforementioned results, we construct a practical knowledge sound FHE-SNARK scheme with explicit resistance against input-dependent attack. Our FHE-SNARK demonstrate a **7.3×∼ 24.4×** acceleration in proof generation time compared to [31]. Specifically, for arithmetic circuits with 2^{20} gates, our implementation costs **2.27** hours (**8.2** ms/gate) to generate a computation proof on a single-core CPU. If we assume that the instance ciphertexts are already in iNTT form, the implementation costs only **0.68** hours (**2.4** ms/gate). Moreover, the scheme is well-suited for multi-core processors since all homomorphic operations are parallelizable.

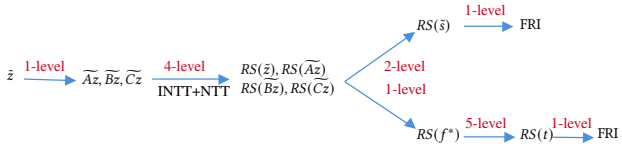

Figure 1: Multiplication-depth of prior constructions[3, 31]

1.2 Technique Overview

New Construction of SNARKs for encrypted instances. In modern SNARK constructions, the prover typically encodes the witness and statement as polynomials and transforms the problem of verifying instance correctness into checking or querying the encoded polynomials. The prover provides oracles for these polynomials and then instantiates them using concrete cryptographic primitives such as hash functions or polynomial commitments. Since in VCoed schemes, the client only sends the encryption of its input to server, the arithmetized R1CS instance will have an encrypted statement and a partially encrypted witness w. As a result, the prover (acted by the server) can only obtain the encoded polynomials in encrypted.

Prior approaches. Prior constructions[3, 31] build upon the Fractal scheme and replace some oracles/messages sent by the prover to the corresponding encrypted ones. Specifically, for encrypted $z = (w, x)$, they encode it to an encrypted univariate polynomial $\tilde{z}$ such that $\tilde{z}(h_i) = z_i$ for a specific set $H = \{h_0, \cdots, h_{n-1}\}$. Az, Bz, and Cz are encoded similarly as $\widetilde{Az}, \widetilde{Bz}, \widetilde{Cz}$. Verifying $Az \circ Bz = Cz$ ultimately reduces to checking, via FRI protocol homomorphically, the encrypted Reed-Solomon code of polynomials like $s = \frac{\widetilde{Az} \cdot \widetilde{Bz} - \widetilde{Cz}}{v_H}$ and $t = \frac{f^* - X \cdot g}{v_H}$, where $v_H = \Pi_{h \in H}(x - h)$, f^*, g are specially designed polynomials constructed from $\widetilde{Az}, \widetilde{Bz}, \widetilde{Cz}$. Their oracles for encrypted strings are directly instantiated using hash functions. They made significant progress towards practical verifiable computation, however, their construction still has following shortages:

- *High multiplication-depth.* Their constructions make extensive use of NTT and are highly sequential. Even equipped with 2-layer NTT, they still exhibit a substantial multiplication-depth (12 depths, as illustrated in Fig.1), significantly increasing the cost of single homomorphic evaluations in the underlying FHE scheme.
- *Limited Compatibility with FHE SIMD operations.* Applying a 2-layer NTT on large-size encrypted polynomials is not well-suited for FHE SIMD operations, as it leads to a substantial number of matrix-vector multiplications.

Our approaches. Unlike prior constructions, our scheme builds upon the "*multilinear* polynomial commitment + polynomial interactive oracle proof (PIOP)" paradigm. In our construction, the prover encodes encrypted z as an encrypted multilinear polynomial $\tilde{z}$ on $\{0,1\}^{\log |z|}$. Az, Bz, and Cz are encoded similarly as $\widetilde{Az}, \widetilde{Bz}, \widetilde{Cz}$. Follows Spartan's PIOP [60], verifying $Az \circ Bz = Cz$ reduces to checking: 1) $F(x) = \widetilde{Az} \cdot \widetilde{Bz} - \widetilde{Cz} = 0$ on $\{0,1\}^*$, and 2) $\widetilde{Mz}$ encodes Mz for $M = A, B, C$. They can be further reduced to sumcheck $\Sigma_{x \in \{0,1\}^*} eq(x, \tau) F(x)$ and $\Sigma_y \widetilde{M}(\tau', y)\tilde{z}(y) = \widetilde{Mz}(\tau')$. Through running the well-known sumcheck[56] homomorphically, these checks reduce to random queries to the encrypted polynomial z, avoiding NTT unlike univariate FRI-based approaches.

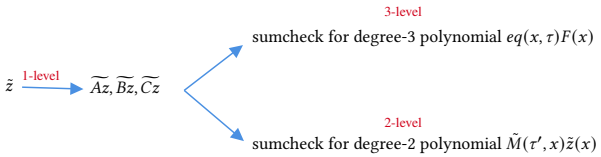

Figure 2: Multiplication-depth of our construction

However, running sumchecks homomorphically introduces significant multiplicative depths. For a n-round sumcheck proving $\Sigma_x f(x) = z$ (for simplicity, assume f is an n-variate multilinear polynomial), it computes, in round i, evaluations $f(x_1, \cdots, x_{n-i}, r_{n-i+1}, \cdots, r_n)$ for all $x_1, \cdots, x_{n-i} \in \{0,1\}$ recursively and thereby introduces n multiplication depth. We reduce depth by computing evaluations in each round via linear combinations directly on f's evaluations on $\{0,1\}^n$, costing $O(n \cdot 2^n)$ instead of $O(2^n)$. Furthermore, the modified sumcheck protocol exhibits excellent compatibility with FHE packing techniques because the linear combinations in each round share the same scalars for all $x_1, \cdots, x_{n-i} \in \{0,1\}$. The above construction can also be extended to low-degree polynomial f in the form of $f := h(g_1, \cdots, g_n)$ where $g_1, \cdots, g_n$ are all multilinear polynomials, which suffices for our construction.

With the modified sumcheck protocol, the proof system achieves a low multiplication-depth (only 4), as described in Figure.2.

However, there is still one caveat: we need to instantiate the polynomial oracle for encrypted polynomials. Since FRI-based polynomial commitment exhibits weak compatibility with packing[2], we build upon Ligero/Brakedown's polynomial commitment. In the commitment phase, the committer rewrites the encrypted evaluations on the hypercube as a matrix and homomorphically encodes each row using an error correcting code. The Merkle-hash of the resulting encoded matrix serves as the commitment. In the evaluation phase, the prover sends random linear combinations of the rows of the evaluation matrix and Merkle authentication paths for random columns of the encoded matrix to the verifier, who can then verify the "well-formedness" of the commitment and evaluations using the property of error correcting code.

This construction exhibits excellent compatibility with packing because each row is encoded using the same encoding scheme. By packing each column of the evaluation matrix into a single ciphertext, the commitment algorithm requires only scalar multiplications, while the evaluation consists solely of inner products on packed ciphertexts. Moreover, the packing method here is identical to that used in the sumcheck protocol, allowing us to reuse the packing. We apply the Reed-Solomon code as the underlying encoding scheme because it can be accelerated via a constant-layer NTT algorithm [3] while maintaining a small depth.

In our SNARK construction, the prover commits only to the encrypted polynomial $\tilde{z}$, so the overall depth is determined by the maximum of the polynomial IOP and the polynomial commitment depths. Using 3-layer NTTs, our polynomial commitments require 3 depths (3 depths for the commitment phase and 1 depth for the evaluation phase; since both are computed from $\tilde{z}$, the resulting depth is their maximum). Our polynomial IOP requires 4 depths. Therefore, the final SNARK achieves a multiplication depth of 4.

[2]FRI-based PCS uses a single RS-code of a large polynomial. Using a two-layer NTT to compute the codewords (as in prior constructions[31]) involves arranging the polynomial coefficients into a two-dimensional matrix and performing one-layer NTTs sequentially on the rows and columns via matrix-vector multiplications. When packing either the rows or the columns, only half of the computations can be reduced to scalar multiplications; the other half still requires costly plaintext–ciphertext matrix-vector multiplications.

[3]Unlike prior construction, we perform a large number of NTT operations only on *small-size* encrypted polynomials/vectors.

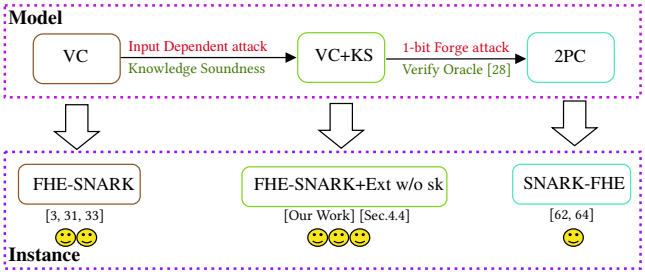

Figure 3: Our result, where "KS" denotes knowledge soundness, "Ext w/o sk" stands for extractability without secret key and smiles stand for practical efficiency.

Verifiable computation on encrypted data. In this paper, our VCoed scheme requires a stronger security called knowledge soundness property to avoid potential attack that the server modifies its input depending on clients in encrypted. This additional security requires that the server indeed knows its input in plaintexts to the computed circuit. However, since SNARKs operate on the encrypted witness, the original knowledge soundness of SNARKs does not work here because the extractor requires secret key to decrypt the prover's messages and extract the witness.

To achieve knowledge soundness, we require the server to send a plaintext polynomial commitment that commits its input. This commitment enables extraction of the input without needing the secret key. Since this input is also part of the committed value z, we can verify that the committed input is consistent with the corresponding portion of z through additional queries to the underlying polynomial commitments, using a process similar to the verification of public input consistency.

A note on the difference between the two paradigms of VCoed. While our main focus is on the new VCoed instantiation based on the FHE-friendly SNARK scheme, Phalanx, we also conducted a broader conceptual investigation into the modeling of VCoed. We formalize the functionality gap between the two paradigms: FHE-SNARK and SNARK-FHE(also informally discussed in [3, 28, 34, 58]). In particular, we identified subtle security issues in the FHE-SNARK setting when the server holds its own private input. To address this, we formalize a class of input-dependent attacks, and further enhance the Phalanx-based FHE-SNARK construction with a "knowledge soundness" property, which provably prevents such attacks. This refined scheme brings FHE-SNARK closer to the security guarantees of SNARK-FHE in the secure two-party computation (2PC) setting, albeit with an unavoidable 1-bit leakage gap (which is also identified in [3] but not formalized). While not the main contribution of our work, this analysis highlights a conceptual separation between existing approaches and motivates our corresponding definitions. We include the full discussion in Appendix A for interested readers and briefly summarize our full result as Figure. 3.

2 Preliminaries

Notions. Let λ denote the security parameter. For any positive integer m, we denote by $[m]$ the set $\{0, 1, \cdots, m-1\}$ and by $\mathsf{bin}_n(m) \in \{0,1\}^n$ its n-bit binary representation (padded with leading zeros when necessary). For a vector $r = (r_{n-1}, \cdots, r_0) \in \{0,1\}^n$, we denote by $\mathsf{int}(r)$ the integer $\sum 2^i \cdot r_i$. We use the standard abbreviation PPT to denote probabilistic polynomial time. For two random ensembles $\mathcal{X} := \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} := \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, we write $\mathcal{X} \overset{c}{\approx} \mathcal{Y}$ to mean $\mathcal{X}$ and $\mathcal{Y}$ are indistinguishable against all polynomial-size circuits. For any vector $v = (v_0, v_1, \cdots, v_{n-1})$ and any integer $i \in [n]$, we denote by $v[i] := v_i$.

2.1 Proof Systems

Proof systems serve as a fundamental building block in this paper, ensuring the correctness of computations. Some portions of these definitions are taken verbatim from [14].

Definition 2.1 (R1CS Indexed Relation). The indexed relation $\mathcal{R}_{R1CS}$ is the set of all triples

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ((\mathbb{F}, A, B, C, m, n), x, w)$$

where $\mathbb{F}$ is a finite field, $m, n \in \mathbb{N}$, A, B, C are $m \times m$ matrices over $\mathbb{F}$ with at most n non-zero entries, and $z = (w, x)$ is a vector in $\mathbb{F}^m$ such that $Az \circ Bz = Cz$. Here, $\circ$ denotes the Hadamard (entry-wise) product.

Definition 2.2 (Interactive Proof). Let $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ be an interactive protocol described as follows:

- $\mathsf{Setup}(1^\lambda)$: On input a security parameter 1^λ, Setup outputs a common reference string crs.
- $\mathcal{I}(\mathsf{crs}, \mathbb{i})$: On input crs and the index $\mathbb{i}$, the **deterministic** algorithm $\mathcal{I}$ outputs the verifier and prover parameters $(\mathsf{vp}, \mathsf{pp})$.
- $\langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle$: $\langle \mathcal{P}, \mathcal{V} \rangle$ is a protocol between the prover and verifier, where the prover wants to show that $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}$ and the verifier outputs a bit indicates acceptance or not. Here we denote the output of the verifier by $0/1 \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle$.

Such an interactive protocol is called a proof system for indexed relation $\mathcal{R}$ if the following properties hold:

- Completeness: for all $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}$,

$$\Pr\left[\langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle = 1 \;\middle|\; \begin{matrix} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{crs}, \mathbb{i}) \end{matrix} \right] = 1$$

- Soundness: For every pair of unbounded adversary algorithm $(\mathcal{A}_1, \mathcal{A}_2)$, it holds that:

$$\Pr\left[\langle \mathcal{A}_2(\mathsf{pp}, \mathbb{x}, st), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle = 1 \wedge (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}) \;\middle|\; \begin{matrix} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, st) \leftarrow \mathcal{A}_1(\mathsf{crs}) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{crs}, \mathbb{i}) \end{matrix} \right] \leq \mathsf{negl}(1^\lambda)$$

where $\mathcal{L}(\mathcal{R})$ is the set defined as $\{(\mathbb{i}, \mathbb{x}) \mid \exists\, \mathbb{w}\ s.t.\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}\}$.

Furthermore, an argument system is defined similarly to a proof system, except that it is only required to be sound against probabilistic polynomial-time adversary. An interactive protocol is called *public-coin* if all messages sent by the verifier are uniformly random elements from some subset of the plaintext space, independent of the current partial transcript. If a proof (argument) system Π involves only one message in its online phase $\langle \mathcal{P}, \mathcal{V} \rangle$, specifically, the prover sends a proof $\pi \leftarrow \mathcal{P}(\cdot)$ to the verifier, who then verifies the proof using $\mathcal{V}$, we call it a non-interactive proof (argument) system.

Definition 2.3 (Knowledge Soundness). A proof system (or argument system) is called a proof of knowledge (or argument of knowledge, respectively) if it satisfies the knowledge soundness defined as follows: There exists a probabilistic polynomial time oracle machine $\mathcal{E}$, called the extractor, such that for any pair of unbounded (or probabilistic polynomial-time, respectively) adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, the following probability is negligible:

$$\Pr\left[\begin{matrix} \langle \mathcal{A}_2(\mathsf{pp}, \mathbb{x}, st), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle = 1 \wedge \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \end{matrix} \;\middle|\; \begin{matrix} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, st) \leftarrow \mathcal{A}_1(\mathsf{crs}) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{crs}, \mathbb{i}) \\ \mathbb{w} \leftarrow \mathcal{E}^{\mathcal{A}_1, \mathcal{A}_2}(\mathsf{crs}, \mathbb{i}, \mathbb{x}) \end{matrix} \right]$$

Definition 2.4 (Succinct Non-interactive Arguments of Knowledge). A non-interactive argument of knowledge is considered succinct if both the verification time and communication size are sublinear in the size of the witness.

Succinct Non-interactive Arguments of Knowledge (SNARKs) can be constructed from various information-theoretic proof systems that give the verifier oracle access to prover messages. These information-theoretic proof systems can be compiled using various cryptographic tools, such as hash functions and polynomial commitments. A typical information-theoretic proof systems is polynomial interactive oracle proofs (PIOPs).

Polynomial Interactive Oracle Proofs (PIOPs). PIOPs allows the prover sends several polynomial oracles in each round. More precisely, a PIOP for an indexed relation $\mathcal{R}$ consists of a deterministic index algorithm $\mathcal{I}$ and a protocol $\langle \mathcal{P}, \mathcal{V} \rangle$ such that $\mathcal{I}(\mathbb{i})$ encodes the index $\mathbb{i}$ as several polynomial functions $\mathsf{Ind}(\mathbb{i})$ and $\langle \mathcal{P}(\mathsf{Ind}(\mathbb{i}), \mathbb{x}, \mathbb{w}), \mathcal{V}^{\mathsf{Ind}(\mathbb{i})}(\mathbb{x}) \rangle$ is a public-coin protocol where in each round the prover can send some polynomial function oracles to the verifier, and the verifier can query these oracles, as well as $\mathsf{Ind}(\mathbb{i})$, at arbitrary points of its choice.

A PIOP can be compiled into an interactive argument by using a polynomial commitment scheme to instantiate the polynomial oracles. Roughly, if both of the polynomial commitment scheme and the PIOP have knowledge soundness, then the resulting scheme has knowledge soundness. Additionally, if the polynomial commitment scheme is hiding, and the EVAL protocol of the polynomial commitment scheme and the PIOP are both HVZK, then the resulting interactive argument is also HVZK.

In this paper, we occasionally need to prove an instance of which statement and witness might be encrypted. This SNARK scheme can be formalized as **SNARKs for encrypted instances**[3, 31, 33], where only the verifier possesses the secret key for the encryptions and the prover only knows public key. This scheme satisfies similar completeness and computational binding properties with only trivial syntactic changes. Additionally, the knowledge soundness properties are modified to allow the extractor access to the secret key (or a decryption oracle).

2.2 Polynomial commitment over encrypted values

Polynomial commitments enable a committer to commit a polynomial and then open the value of the polynomial at arbitrary points.

In this paper, we need to prove statement encrypted using a FHE scheme, which necessitates committing encrypted polynomials. The concept of polynomial commitments over encrypted values was first introduced by Garg et al. [33] and subsequently appeared implicitly in [3, 31]. We provide the formal definition of polynomial commitments over encrypted values as follows:

Definition 2.5 (Polynomial Commitment on Encrypted Values). A polynomial commitment for encrypted multilinear polynomials with respect to an encryption scheme ENC = (KeyGen, Enc, Dec) consists of four algorithms COM = (Setup, Com, deCom, Eval) :

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter 1^λ, the Setup algorithm outputs a public parameter pp.
- $\mathsf{Com}(\mathsf{pp}, \mathsf{pk}, \mathsf{Enc}_{\mathsf{pk}}(f))$: On input the public parameter pp, an encryption public key pk, and an encrypted polynomials $\mathsf{Enc}_{\mathsf{pk}}(f)$ where $f \in \mathbb{F}[X_1, \cdots, X_d]$, the Com algorithm outputs a commitment C and a decommitment τ.
- $\mathsf{deCom}(\mathsf{pp}, C, \tau, \mathsf{sk})$: On input the public parameter pp, a commitment C, a decommitment τ, and the secret key sk, the deCom algorithm outputs the committed multilinear polynomial $f \in \mathbb{F}[X_1, \cdots, X_d]$ in plaintext, or $\perp$ if the decommitment is invalid.
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{pk}, C, z, y^*; \mathsf{sk}, \mathsf{Enc}_{\mathsf{pk}}(f), \tau)$: Eval is an interactive protocol

$$\langle P(\mathsf{Enc}_{\mathsf{pk}}(f), \tau), V(\mathsf{sk})\rangle(\mathsf{pp}, \mathsf{pk}, C, z, y^*)$$

with the setup algorithm Setup where P wants to convince V of the followings relation:

$$R_{eval} = \left\{ (\mathsf{pp}, \mathsf{pk}, C, z, y^*; \mathsf{sk}, \mathsf{Enc}_{\mathsf{pk}}(f), \tau) : \begin{array}{r} \mathsf{deCom}(\mathsf{pp}, C, \tau, \mathsf{sk}) = f \wedge \\ y = \mathsf{Dec}_{\mathsf{sk}}(y^*) \wedge f(z) = y \end{array} \right\}.$$

Such a polynomial commitment COM need to satisfy the following properties:

- **Completeness**: For all $\lambda \in \mathbb{N}, d \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KeyGen}(1^\lambda)$, multilinear polynomial $f \in \mathbb{F}[X_1, \cdots, X_d]$, and any $z \in \mathbb{F}^d$, the following probability equals 1.

$$\Pr\left[1 = \mathsf{Eval}(\mathsf{pp}, \mathsf{pk}, C, z, y^*; \mathsf{sk}, \mathsf{Enc}_{\mathsf{pk}}(f), \tau) \,\middle|\, \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ (C; \tau) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathsf{pk}, \mathsf{Enc}_{\mathsf{pk}}(f)) \\ y = f(z), y^* \in \mathsf{Enc}_{\mathsf{pk}}(y) \end{array} \right]$$

- **Computational Binding**: For all $\lambda \in \mathbb{N}, d \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KeyGen}(1^\lambda)$, and any polynomial-size adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} f_0 \neq f_1 \wedge \\ f_0, f_1 \neq \perp \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ (C, \tau_0, \tau_1) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}), \\ f_0 \leftarrow \mathsf{deCom}(\mathsf{pp}, C, \tau_0, \mathsf{sk}), \\ f_1 \leftarrow \mathsf{deCom}(\mathsf{pp}, C, \tau_1, \mathsf{sk}) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

- **Knowledge Soundness**: Let $\mathsf{Eval} = \langle P, V\rangle$. Then there exists a PPT oracle machine $\mathcal{E}$, called the extractor, such that for any pair of PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and any $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KeyGen}(1^\lambda)$, the following probability is negligible.

$$\Pr\left[\begin{array}{r} \langle \mathcal{A}_2(st), V(\mathsf{sk})\rangle(\mathsf{pp}, \mathsf{pk}, C, z, y^*) = 1 \wedge \\ f(z) \neq y \text{ or } \mathsf{deCom}(\mathsf{pp}, C, \tau, \mathsf{sk}) \neq f \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (C, z, y^*, st) \leftarrow \mathcal{A}_1(\mathsf{pp}, \mathsf{pk}) \\ (f, \tau) \leftarrow \mathcal{E}^{\mathcal{A}_1, \mathcal{A}_2}(\mathsf{pp}, \mathsf{sk}, C, z, y^*) \end{array} \right]$$

where y is the decryption of y^*, i.e., $y = \mathsf{Dec}_{\mathsf{sk}}(y^*)$.

$\underline{\mathsf{G}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda)}$:

1: $b \xleftarrow{\$} \{0, 1\}$
2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathsf{C})$
3: $(m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{PK}, \mathsf{C})$
4: $(c_{m_b}, \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, m_b)$
5: $\hat{b} \leftarrow \mathcal{A}(c_{m_b}, \mathsf{state})$
6: **return** $b \stackrel{?}{=} \hat{b}$

---

3 Verifiable Computation on Encrypted Data

Verifiable Computation on Encrypted Data (VCoed) allows a client to outsource its computation to an untrusted server while (1) protecting the data privacy and (2) enabling the client to verify computation correctness. We adapt the definition from [34] and extend the soundness property to knowledge soundness property to prevent potential attack in the cases that both client and server have their inputs.

Definition 3.1. **(Verifiable Computation).** A verifiable computation scheme VC consists of four polynomial-time algorithms:

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathsf{C})$: given a circuit C, the setup algorithm outputs a key pair (sk, pk).
- $(\mathsf{Enc}(x), \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, x)$: an encoding algorithm takes an input data x and public key pk, and outputs an encoded $\mathsf{Enc}(x)$ and a state st.
- $(\mathsf{Enc}(\mathsf{out}), \pi) \leftarrow \mathsf{VC.Compute}(\mathsf{pk}, \mathsf{Enc}(x), \mathsf{C}, y)$: the computation algorithm takes the public key pk, the encoded $\mathsf{Enc}(x)$, circuit C and server's input y, it outputs an encoded computing result $\mathsf{Enc}(\mathsf{out})$ and a proof π.
- $\mathsf{out}/\perp \leftarrow \mathsf{VC.Verify}(\mathsf{sk}, \mathsf{Enc}(\mathsf{out}), \mathsf{st}, \pi)$: given the secret key sk, the encoded result $\mathsf{Enc}(\mathsf{out})$, state st, and proof π, the verification algorithm returns out behind its encoding $\mathsf{Enc}(\mathsf{out})$ if $\mathsf{out} = \mathsf{C}(x, y)$ or outputs $\perp$ otherwise.

We then formally define the correctness, outsourceability, and security of the VCoed scheme.

Correctness. The correctness property ensures that the honest server will pass the verification of an honest client. Specifically, the following probability is negligible.

$$\Pr\left[\perp \leftarrow \mathsf{VC.Verify}(\mathsf{sk}, \mathsf{Enc}(\mathsf{out}), \mathsf{st}, \pi) \,\middle|\, \begin{array}{r} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathsf{C}) \\ (\mathsf{Enc}(x), \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, x) \\ (\mathsf{Enc}(\mathsf{out}), \pi) \leftarrow \mathsf{VC.Compute}(\mathsf{pk}, \mathsf{Enc}(x), \mathsf{C}, y) \end{array} \right]$$

Outsourceability. A VCoed scheme is outsourceable if the time required by the client to run VC.ProbGen and VC.Verify and the size of π are $o(|C|)$, where $|C|$ is the size of circuit C.

Security. In this paper, we mainly consider the security model when client and server both have input.

Client security: Previous work [33] captures the client security (including privacy and integrity guarantee) of the verifiable computation scheme by defining the following two properties.

- Client-privacy: meaning that a malicious server should not learn any information about the client's input. More formally, a VCoed scheme is verifier-private, if for any PPT adversary $\mathcal{A}$, its advantage in the following game $\mathsf{G}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda)$, defined as $\mathsf{Adv}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda) = \Pr[\mathsf{G}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda) \rightarrow 1] - 1/2 = \mathsf{negl}(\lambda)$.
- Soundness: meaning that a malicious server cannot produce an invalid computational result that passes verification. More formally, a VCoed scheme is sound, if for any PPT adversary $\mathcal{A}$, it

holds that its advantage in the following game $\mathsf{G}^{\mathsf{Soundness}}_{\mathsf{VC},\mathcal{A}}(\lambda)$, defined as $\mathsf{Adv}^{\mathsf{Soundness}}_{\mathsf{VC},\mathcal{A}}(\lambda) := \Pr[\mathsf{G}^{\mathsf{Soundness}}_{\mathsf{VC},\mathcal{A}}(\lambda) \to 1] = \mathsf{negl}(\lambda)$.

$\mathsf{G}^{\mathsf{Soundness}}_{\mathsf{VC},\mathcal{A}}(\lambda)$:

1: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathsf{C})$
2: $x \leftarrow \mathcal{A}(\mathsf{PK}, \mathsf{C})$
3: $(\mathsf{Enc}(x), \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, x)$
4: $(\mathsf{Enc}(\mathsf{out}'), \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{Enc}(x), \mathsf{C})$
5: **if** $\mathsf{VC.Verify}(\mathsf{sk}, \mathsf{Enc}(\mathsf{out}'), \mathsf{st}, \pi) = y' \wedge \nexists y \ s.t. \ \mathsf{C}(x, y) = \mathsf{out}'$
6: **return** 1
7: **return** 0

In this paper, we introduce **knowledge soundness** as a strengthened version of soundness. Informally, in scenarios where both the client and the server have inputs—while the client's input remains private—the server may still alter its input depending on the client's input without fully knowing the modified value. Such an input-dependent attacks are common in 2PC [51]. To prevent them, we adopt knowledge soundness as a new security requirement for VCoED. A detailed discussion can be found in Appendix A.

- Knowledge soundness: A verifiable computation system is knowledge sound if there exists a probabilistic polynomial time oracle machine $\mathcal{E}$ called the extractor such that for any PPT adversary $\mathcal{A}$, the following probability is negligible,

$$\Pr\left[\begin{array}{r|l} \mathsf{VC.Verify}(\mathsf{sk}, c, \mathsf{st}, \pi) = \mathsf{out} \neq \perp \ \wedge & (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathsf{C}) \\ \mathsf{out} \neq \mathsf{C}(x, y) & (\mathsf{Enc}(x), \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, x) \\ & (c, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{Enc}(x), \mathsf{C}) \\ & y \leftarrow \mathcal{E}^{\mathcal{A}}((\mathsf{pk}, \mathsf{Enc}(x), \mathsf{C}), (c, \pi)) \end{array}\right]$$

Server security: Following [3], we provide a simulation-based definition for server-privacy. This definition is analogous to that used in 2PC, requiring that the client's view can be simulated by a simulator in the ideal world, or equivalently, that the simulator has one-time oracle access to the function $\mathsf{C}(\cdot, w)$).

- Server-privacy: A VCoed protocol is server-private if there exists a PPT simulator $\mathcal{S} = (S_1, S_2)$ such that for every circuit C, every input w, and every adversary $\mathcal{A}$, the following distributions are computational indistinguishable:

$\mathcal{D}_1$: $\{\mathsf{pk}, \mathsf{sk}, r, \delta_y, \pi\}_\lambda$, where $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathsf{C})$, $\delta_x \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{sk}; r)$, r is the randomness of $\mathcal{A}$, and $(\delta_y, \pi) \leftarrow \mathsf{VC.Compute}(\mathsf{pk}, \delta_x, \mathsf{C}, w)$.

$\mathcal{D}_2$: $\{\mathsf{pk}, \mathsf{sk}, r, \mathcal{S}_2^{\mathcal{A}, O_w}(1^\lambda, \tau, \delta_x)\}_\lambda$, where $(\mathsf{sk}, \mathsf{pk}, \tau) \leftarrow \mathcal{S}_1(1^\lambda, \mathsf{C})$, $\delta_x \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{sk}; r)$, r is the randomness of $\mathcal{A}$, and O_w is a one-time oracle ($\mathcal{S}_2$ can query this oracle for only once) that on input x, it outputs $\mathsf{C}(x, w)$.

Furthermore, A VCoed protocol is called honest-client server-private if it satisfies the property above against a honest-client.

4 Construction: Knowledge soundness VCoed based on FHE-SNARK

In the construction of verifiable computation, the server not only needs to homomorphically compute the result but also must provide a proof for a verifiable computation instance, where the client input is encrypted. Additionally, this proof must ensure that the server indeed knows its private input to prevent dependent-input attacks.

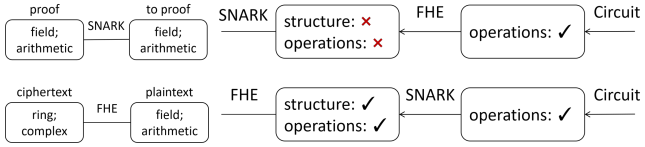

Figure 4: SNARK-FHE vs FHE-SNARK

This section is organized into four parts: first, we establish why FHE-SNARK outperforms SNARK-FHE in VCoed construction; second, we develop a polynomial commitment over encrypted values; third, we propose (zk-)SNARKs for encrypted instance; and finally, we synthesize these components to construct verifiable computations.

4.1 Why FHE-SNARK?

In this paper, we focus on FHE-SNARK. One immediate question one may ask is why FHE-SNARK and what are the difference between SNARK-FHE (applying SNARKs to prove FHE operations) and FHE-SNARK (homomorphically evaluating SNARK proofs). To start, we note that while both achieve client privacy and knowledge soundness, SNARK-FHE, interestingly, prevents 1-bit forge attack, thus directly realize a 2PC functionality. On the other hand, FHE-SNARK can only be used to realize VCoed. Despite its weaker functionality, FHE-SNARK in fact has a significant advantage over SNARK-FHE in terms of efficiency. This mainly thanks to the alignment of algebraic structures and supported operations between FHE and ZK, when FHE is "outside" of ZK as in FHE-SNARK. On the other hand, SNARK-FHE encounters challenges due to structural and operational mismatches. We summarize this in Figure 4, and present a detailed and comprehensive demonstration in Appendix C. This explains why FHE-SNARK can be orders of magnitude faster than SNARK-FHE, as discussed in Section 1, and why we focus on FHE-SNARK in our following construction.

4.2 Low Multiplication-Depth Polynomial Commitment over Encrypted Values

Polynomial commitments are a core primitive of SNARKs. Most modern SNARKs encode the statement and witness (as well as the circuit itself) as polynomials and reduce the verification to queries to these polynomials. However, since in the verifiable computation the client's input is encrypted and the server can only conclude the computation homomorphically, the server has to compute the polynomial commitment for encrypted polynomials (in other words, homomorphically compute the commitment).

In this section, we show how to construct such a polynomial commitment scheme for encrypted multilinear polynomials with flexible and low multiplication depth and fully harnesses FHE ciphertext packing capabilities. In such a scheme, the number of variables n in the polynomial and the public key pk of the FHE scheme is public, and the committer knows the encrypted point-value representation $[f]_{\mathsf{c}} := \{\mathsf{Enc}_{\mathsf{pk}}(f(x))\}_{x \in \{0,1\}^n}$ while the receiver knows the secret key of underlying FHE scheme.

In short, we use an HE version of Ligero [1] (Brakedown scheme in [38]), where the prover holds an HE-encrypted witness and evaluate the commit and evaluation procedures homomorphically.

We describe this process in detail below with some optimizations tailored to improve the concrete efficiency for the HE schemes.

Setup. The setup phase is transparent and the algorithm simply selects a random string k to serve as the index for the hash function h_k and outputs $\mathsf{pp} = k$.

Commitment. Roughly, the committer views $[f]_{\mathsf{c}}$ as a $2^d \times 2^{n-d}$ matrix M and then homomorphically encodes each row of M using an error-correcting code scheme Encode. Finally, apply a Merkle-hashing to all columns of encoded scheme. The resulting hash root serves as the commitment C.

The decommitment phase is straightforward: the committer sends $[f]_{\mathsf{c}}$, $\hat{M}$, and used randomness as the decommitment τ. The receiver outputs f by decryption if 1) C is indeed the Merkle-hash of $\hat{M}$ and 2) the decryption of $\hat{M}$ is equivalent(close) to the encoding of decryption of M.

Note that the original Brakedown scheme employs a specifically designed coding scheme that achieves linear encoding time; however, it incurs a substantial multiplication depth, rendering the aforementioned homomorphic encoding impractical. Here, we modify the commitment algorithm to achieve both *low multiplication depth* and *excellent compatibility with FHE SIMD operations*.

First, we choose the well-known Reed-Solomon code scheme as the underlying coding scheme and use the constant-layer version of fast Number Theoretic Transform (NTT) algorithm to accelerate the encoding phase[4]. One can flexibly choose the number of layers of underlying NTT to achieve a balance between the circuit depth and the number of scalar multiplications, e.g., setting the depth to k results in a computational complexity of $O(2^{n+n/k})$, where n is the length of message. (Note that this requires an NTT-friendly field; however, one may alternatively employ a foldable code scheme [66] along with a similar constant-layer encoding algorithm to eliminate this constraint.) Second, since each row of M is encoded deterministically using the same linear encoding algorithm, we can pack all ciphertexts in each column of M into a single ciphertext. The encoding algorithm can then be applied directly to these packed ciphertexts, producing the ciphertexts in $\hat{M}$ in packed form. A formal description of the commitment and decommitment phases is presented in Figure 5.

Evaluation. In the evaluation protocol, the prover aims to prove that $f(x) = y$ for a committed polynomial f, a point x, and an encrypted value y. The evaluation protocol consists of two phases: a testing phase and an evaluation phase. Testing phase is used to ensure that the commitment is "well-formed", meaning that each row in $\hat{M}$ is indeed (close to) an (encrypted) codeword. The evaluation phase then operates on "well-formed" commitments. Importantly, the testing phase only needs to be executed once and is independent of the specific evaluation point.

Testing phase. Our testing phase mirrors that of Brakedown's polynomial commitment scheme, with the exception that certain operations are performed homomorphically over FHE ciphertexts.

[4]Recall that a standard n-dimensional NTT algorithm consists of $\log(n)$ layers, where each element in a layer is a linear combination of two elements from the previous layer. To construct a k-layer NTT, one can compress $\frac{1}{k}\log n$ layers from the standard NTT into a single layer; that is, each element in a layer is now a linear combination of $n^{\frac{1}{k}}$ elements from the previous layer. And similar trick is also used in [33].

Commit: On input $[f]_c$, the committer proceeds as follows:

(1) Rewrite $[f]_c := \{\mathsf{Enc}_{\mathsf{pk}}(f(x))\}_{x\in\{0,1\}^n}$ as a $2^d \times 2^{n-d}$ matrix $M = \{u_{i,j}\}_{i\in[2^d], j\in[2^{n-d}]}$ where $u_{i,j} = \mathsf{Enc}_{\mathsf{pk}}(f(\mathsf{bin}_n(i\cdot 2^{n-d} + j)))$.
(2) Homomorphically encode each row u_i of M using the Reed-Solomon code algorithm Encode and obtain the encoded matrix $\hat{M}$.
  (a) Suppose that all ciphertexts in a column of M have been packed into a single ciphertext ct_j. Then the committer can compute all the packed ciphertexts $\hat{\mathsf{ct}}_j$ of the columns of $\hat{M}$ via $\{\hat{\mathsf{ct}}_j\} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\mathsf{Encode}, \{\mathsf{ct}_j\})$.
  (b) The committer can use constant-layer NTT algorithm to accelerate Encode algorithm.
(3) Use h to Merkle-hash the columns of $\hat{M}$. Output the hash root as the commitment *com* of f. All randomness (if exists) used during the commitment phase, along with $[f]_{\mathsf{c}}$ and $\hat{M}$, constitutes the decommitment τ.

Decommit: Rewrite $[f]_c$ as M. Recompute the Merkle-hash of $\hat{M}$ using provided randomness and check 1) whether the hash root is indeed *com*, 2) for each row, the distance between $\mathsf{Dec}(\hat{M})$ and $\mathsf{Encode}(\mathsf{Dec}(M))$ is less than $\gamma/3$, where γ is the minimal distance of the linear code. If both checks pass, output f by decrypting $[f]_c$.

Figure 5: Commitment and decommitment algorithm

In this phase, the receiver begins by sending a set of random values $r = \{r_i\}_{i\in[2^d]}$. The committer responds with the homomorphic linear combination of the rows $\{u_i\}$ of M with scalars $\{r_i\}$. Let $\{\hat{u}_i\}$ be the rows of $\hat{M}$ and $\{\hat{v}_i\}$ be the columns. Assuming that there exists a row $\hat{u}_i$ of $\hat{M}$ of which decryption is far from codewords, then, with overwhelming probability, the decryption $\mathsf{Dec}(\Sigma_{i\in[2^d]} r_i \cdot \hat{u}_i)$ will also be far from the encoding of $\mathsf{Dec}(\Sigma_{i\in[2^d]} r_i \cdot u_i)$. Consequently, the receiver selects a random set $I \subset \{0,1\}^{n-d}$ of appropriate size, and the committer retrieves $\{\hat{v}_i\}_{i\in I}$ by providing the Merkle-authentication path. The receiver checks that for each i, the inner production of their decryption $\mathsf{Dec}_{\mathsf{sk}}(\hat{v}_i)$ and vector r, is consistent with the i-th element of the encoding of $\Sigma_{i\in[2^d]} r_i \mathsf{Dec}_{\mathsf{sk}}(u_i)$. This ensures that the rows in $\hat{M}$ are close to (encrypted) codewords.

The testing phase also exhibits *excellent compatibility with packing technique*. When all ciphertexts in a single column of M have been pack into a single ciphertext, the committer can homomorphically compute the inner product between these packed ciphertexts and the random vector r, producing the encrypted linear combination as the result.

Evaluation phase. The evaluation phase closely resembles the testing phase. From the fact that for any $\alpha = (\alpha_0, \cdots, \alpha_{n-1}) \in \mathbb{F}^n$, $f(\alpha) = q_1 \cdot [M]_p \cdot q_2$, for $q_1 = ((1-\alpha_0, \alpha_0) \otimes \cdots \otimes (1-\alpha_{d-1}, \alpha_{d-1})$ and $q_2 = ((1-\alpha_d, \alpha_d) \otimes \cdots \otimes (1-\alpha_{n-1}, \alpha_{n-1})$ where $[M]_p$ represents the matrix of the plaintext point-value representation of f, in other words, the decryption of M. Consequently, the receiver can choose q_1 as the r used in the testing phase to obtain $q_1 \cdot M$ and, finally, compute $f(\alpha)$ itself.

Evaluation: The evaluation protocol consists of two phases:

Testing phase.

(1) The receiver chooses and sends a random vector $r \in \mathbb{F}^{2^d}$ to the committer.
(2) The committer computes and sends the homomorphic linear combination of the ciphertext rows $\{u_i\}$ of M with scalars $\{r_i\}$, i.e., $\Sigma_{i\in[2^d]} r_i \cdot u_i$.
   (a) Suppose that all ciphertexts in a column of M have been packed into a single ciphertext ct_j. The committer can homomorphically calculate the inner product of these packed ciphertexts and the random vector r to obtain the encrypted resulting linear combination.
(3) The receiver decrypts the receiving vector to obtain u^* and computes its encoding $\hat{u}^* \in \mathbb{F}^N$. It choose a random subset I of $[2^N]$ with size Q and sends it to the committer.
(4) For each $j \in I$, the committer sends the j-th column of $\hat{M}$ to the receiver, along with the corresponding Merkle-hash proof.
(5) The receiver verifies that all the Merkle-hash proofs are valid and checks that for each received column $\hat{v}_j$ of $\hat{M}$, the inner product $\langle r, \mathrm{Dec}(\hat{v}_j)\rangle$ equals the j-th element of $\hat{u}^*$.

Evaluation phase. Suppose the receiver wants to query $f(\alpha)$ with point $\alpha = (\alpha_0, \cdots, \alpha_{n-1}) \in \mathbb{F}^n$, the receiver proceeds as follows:

(1) The receiver computes $q_1 = ((1-\alpha_0, \alpha_0) \otimes (1-\alpha_1, \alpha_1) \otimes \cdots \otimes (1-\alpha_{d-1}, \alpha_{d-1})$ and $q_2 = ((1-\alpha_d, \alpha_d) \otimes \cdots \otimes (1-\alpha_{n-1}, \alpha_{n-1})$. The receiver runs a testing phase with the committer, except that r is set to q_1. If the testing phase fails, abort; otherwise, continue.
(2) Suppose u^* is the vector obtained in step 3 of above testing phase, the receiver computes $f(\alpha) = \langle q_2, u^*\rangle$.

Figure 6: Evaluation protocol

The formalized construction is shown in Figure 6.

Theorem 4.1. *The scheme described in Figure 5 and 6 is a polynomial commitment for encrypted multilinear polynomials.*

Proof. **Completeness and Binding properties.** Completeness follows directly from our construction. Computational binding relies on the collision-resistance of the hash function and the properties of the linear code. To open a commitment to two distinct polynomials, an adversary has to either provide two different encoded matrices, $\hat{M}$ and $\hat{M}'$, that hash to the same root, thereby breaking the collision-resistance of the hash function, or provide the same $\hat{M}$ but two encrypted polynomials $[f]_c$ and $[f']_c$ with $f \neq f'$. In the latter case, we write $[f]_c$ and $[f']_c$ as matrices M and M'. Then, for each row, the distance between $\mathsf{Encode}(\mathrm{Dec}(M))$ (resp. $\mathsf{Encode}(\mathrm{Dec}(M'))$) and $\mathrm{Dec}(\hat{M})$ is less than $\gamma/3$. This implies that, for each row, the distance between $\mathsf{Encode}(\mathrm{Dec}(M))$ and $\mathsf{Encode}(\mathrm{Dec}(M'))$ is less than $2\gamma/3$, which contradicts the fact that γ is the minimal distance of the linear code and $\mathrm{Dec}(M) \neq \mathrm{Dec}(M')$.

Knowledge Soundness. In the analysis of knowledge soundness, we treat the hash function as a random oracle, enabling us to view the evaluation protocol as an IOP system.

As discussed in [38] and existing IOP-to-succinct-argument transformation of [5], given a prover P that convinces the argument-system verifier to accept with non-negligible probability, there is an efficient straight-line extractor capable of outputting IOP proof string π that "opens" the Merkle commitment sent by the argument system prover in the commitment phase.

Recall that the encoding matrix $\hat{M}$ is Merkle hashed in our construction. Therefore, π contains both the encoding matrix $\hat{M}$ and the randomness used for hashing. Consequently, if each row of $\mathrm{Dec}(\hat{M})$ is sufficiently close to a codeword, the extractor can decode them to obtain a matrix $[M]_p$, which serves as the point-value representation of f, and thereby recover f. We will conclude the proof via showing the following three points:

(1) For each row of $\mathrm{Dec}(\hat{M})$, there exists a codeword such that the distance between them is less than $\gamma/3$ (in other words, the matrix $\hat{M}$, along with the encrypted polynomial $[f]_c$ and the randomness, constitutes a valid decommitment).
(2) $[M]_p$ represents a polynomial f that satisfies $f(z) = y$.
(3) For a suitable linear code with minimal distance γ, an efficient decoder algorithm exists.

For any vector u, we denote by $u[j]$ the j-th elements of u. To prove that the rows of $\mathrm{Dec}(\hat{M})$ are close to codewords, we introduce a lemma proved in Ligero:

Lemma 4.2 (Lemma 4.5 in [1]). *Let L be a Reed-Solomon code $[N, k, \gamma]$ with minimal distance $\gamma = N - k + 1$ and e be a positive integer such that $e < \gamma/3$. Then for a set of vectors $\{\hat{u}_i\}_{i\in[m]}, \hat{u}_i \in \mathbb{F}^N$ and supposing $\hat{u}_i'$ be the closest codeword in L to $\hat{u}_i$, if the size of $\Delta := \{j \in [n] | \exists i \in [m]\ s.t.\ \hat{u}_i'[j] \neq \hat{u}_i[j]\}$ is larger than e, then for a random $\hat{u}^*$ in the span of $\{\hat{u}_i\}_{i\in[m]}$, we have*

$$\Pr[d(\hat{u}^*, L) \leq e] \leq \gamma/|\mathbb{F}|$$

Using this lemma, we have that:

Lemma 4.3. *If the prover passes all the checks in the texting phase with probability at least $\gamma/|\mathbb{F}| + (1 - \frac{\gamma}{3\cdot N})^Q$, then there exists a sequence of m codewords $c_0, \cdots, c_{m-1}$ in L such that the size of $\Delta := \{j \in [n] | \exists i \in [m]\ s.t. c_i[j] \neq \hat{u}_i[j]\}$ is less than $\gamma/3$, where $\hat{u}_i$ is the i-th decrypted row of the extracted encoding metrix $\mathrm{Dec}(\hat{M})$.*

Proof. Assume that there exists a row $\hat{u}'$ of the extracted encoding matrix $\mathrm{Dec}(\hat{M})$ such that $d(c', \hat{u}') \geq \gamma/3$ where c' is its closest codeword in L. Then at least one of the following two cases happens:

- **Case I.** Given randomness $\{r_i\}_{i\in[2^d]}$ chosen by the receiver, $d(\Sigma_{i\in 2^d} r_i \cdot \hat{u}_i, L) < \gamma/3$.
- **Case II.** Given randomness $\{r_i\}_{i\in[2^d]}$ chosen by the receiver, $d(\Sigma_{i\in 2^d} r_i \cdot \hat{u}_i, L) \geq \gamma/3$.

Setting e to be the maximal integer less than $\gamma/3$, then from the assumption we have that the size of $\Delta := \{j \in [n] | \exists i \in [m]\ \text{s.t. } u_i[j] \neq \hat{u}_i[j]\}$ is lager than e. From Lemma.4.2, the probability that Case I happens is no more than $\gamma/|\mathbb{F}|$.

In Case II, for a random $j \in [N]$, the probability that the j-th position of $\Sigma_{i\in 2^d} r_i \cdot \hat{u}_i$ is consistent with $\mathsf{Encode}(u^*)$ is no more

than $(1-\frac{\gamma}{3N})$. Since the receiver choose the subset I with size Q, the probability that the prove passes the testing phase is less than $(1-\frac{\gamma}{3\cdot N})^Q$ in this case.

Therefore, the total probability that the prover passes the testing phase is less than $\gamma/|\mathbb{F}|+(1-\frac{\gamma}{3\cdot N})^Q$, arriving at a contradiction. □

Let $c_0, \cdots, c_{m-1}$ be the codewords in L described in above lemma, and $\{u_i\}$ are their decodings, let $[M]_p$ be the matrix of which rows are $\{u_i\}$. Then, from above lemma, the encryption of f represented by $[M]_p$, the extracted matrix $\hat{M}$ and randomness consist a valid opening of commitment.

Next, we show that the extracted polynomial satisfies $f(z)=y$. In the evaluation phase, if the prover can convince the verifier with probability at least $(1-\frac{2\gamma}{3\cdot N})^Q$, then the extracted polynomial f satisfies $f(z)=y$. Otherwise, suppose that $f(z)\neq y$, it must be that $\Sigma_i q_1[i]\cdot u_i \neq u^*$, where q_1 is the vector sent by the receiver in evaluation phase, and u^* is vector receiver obtained in step(2). From the fact that γ is the minimal distance of the code and $d(\Sigma_i q_1[i]\cdot\hat{u}_i, \Sigma_i q_1[i]\cdot c_i) < \gamma/3$(from Lemma.4.3), we have that $d(\Sigma_i q_1[i]\cdot\hat{u}_i, \mathsf{Encode}(u^*)) \geq 2\gamma/3$. Therefore, the probability that the committer passes the evaluation phase is less than $(1-\frac{2\gamma}{3\cdot N})^Q$, arriving at a contradiction.

Finally, we need to ensure that, given a vector $\hat{u}$ satisfying $d(\hat{u}, L) < \gamma/3$, there exists an efficient decoder that decodes it and obtains u with the closest encoding to $\hat{u}$. According to the well-known list decoding algorithm put forward by Guruswami and Sudan[39], for a Reed-Solomon code $L[N,k,\gamma]$, there exists an efficient algorithm that can list all messages m such that $d(\mathsf{Encode}(m),\hat{u}) \leq e$ as long as $e \leq N-\sqrt{kN}$. In other words, to achieve our decoding, we only require that $\frac{N-k+1}{3} \leq N-\sqrt{kN}$. It means $N \geq k$, which is already satisfied by Reed-Solomon code.

In summary, there exists an efficient extractor that, giving the secret key of encryption, can extract the committed polynomial f. Specifically, this is achieved by extracting $\hat{M}$ from the RO oracle, decrypting it to obtain $\mathsf{Dec}(\hat{M})$, and then decoding it to obtain $[M]_p$, the plaintext point-value representation of f. The knowledge soundness error is $\gamma/|\mathbb{F}| + (1-\frac{\gamma}{3N})^Q + (1-\frac{2\gamma}{3N})^Q$. □

4.3 SNARKs for Encrypted Instances

In VCoed, the proved instances are encrypted. This requires us to design a SNARK for encrypted instances (or in other words, runs the SNARK scheme homomorphically) with low multiplication-depth and excellent compatibility with FHE packing technique. In this subsection, we construct a SNARK for R1CS instance (Def.2.1) where $z=(w,x)$ is under encrypted.

PIOPs are information-theoretic proof systems where the prover can optionally send polynomial oracles to the verifier, who can then finalize verification using these oracles. In essence, PIOPs enable provers and verifiers to reduce statement verification to queries on these polynomial oracles, which are subsequently instantiated using polynomial commitments. To construct SNARKs for encrypted instances/relations, we need to demonstrate how to execute PIOPs homomorphically to reduce the verification of these relations to queries on encrypted polynomials while achieving low multiplication-depth and great compatibility with packing.

Sumcheck on encrypted polynomials. The sumcheck protocol is the core of PIOP, which efficiently proves the sum of polynomial evaluations over the hypercube. For polynomial $f:\mathbb{F}^n\to\mathbb{F}$ and $y=\Sigma_{x\in\{0,1\}^n} f(x)$, the sumcheck protocol provides a way for the verifier who has oracle access to f to verify the above claim via *a single query* to $O(f)$.

The sumcheck protocol is an n-round interactive protocol. In the first round, the prover sends a t-degree polynomial $f_1^*(X) = \Sigma_{x'\in\{0,1\}^{n-1}} f(x', X)$ via sending $t+1$ evaluations. Now, if $f_1^*(0)+f_1^*(1)=y$, the verifier only needs to check that the sent f_1^* indeed equals $\Sigma_{x'\in\{0,1\}^{n-1}} f(x',X)$, which is reduced to checking

$$y' = f_1^*(r_{n-1}) = \Sigma_{x'\in\{0,1\}^{n-1}} f(x', r_{n-1}) = \Sigma_{x'\in\{0,1\}^{n-1}} f_1(x')$$

for a random $r_{n-1}\in\mathbb{F}$. Now $f_1(x') := f(x', r_{n-1})$ is a $n-1$-variate polynomial. The prover and verifier recursively execute the above steps on this new sumcheck claim. At the end of the protocol, the verifier only needs to check a statement like $y^* = f(r_0,\ldots,r_{n-1})$, which can be done via a single query to $O(f)$.

Multiplication-depth. When dealing with encrypted polynomials, the prover can only homomorphically compute all evaluations of f in ciphertext form. In the concrete construction, $f=h(g_0,g_1,\cdots,g_{m-1})$ for some encrypted multilinear polynomials $\{g_i\}_{i\in[m]}$ and public polynomial h. Executing the sumcheck protocol homomorphically means that the prover will compute all evaluations of $\{g_i\}$ used in each round via simple linear combinations on the evaluations obtained in previous round. However, this approach increases the multiplicative depth, consequently raising the cost of each homomorphic evaluation. Therefore, we compute the evaluations in each round *directly from* encrypted polynomials $\{g_i\}$. This approach results in a constant multiplicative depth for the protocol, albeit at a computational cost of $O(n\cdot 2^n)$ instead of $O(2^n)$. In practice, this trade-off is often preferable.

Compatibility with packing. Suppose that $\{g_i\}$ are encrypted multilinear polynomials, we view their encrypted representation $[g_i]_c$ as $2^d\times 2^{n-d}$ matrix M and pack each column j into a single ciphertext ct_j, as done in the polynomial commitment construction. Thereby ct_j encrypts $\{g_i(X,\mathsf{bin}_{n-d}(j-1))\}_{X\in\{0,1\}^d}$. From the fact that for all $X\in\mathbb{F}^t, r\in\mathbb{F}^{n-t}$, $g_i(X,r)$ can be expressed as linear combination of $\{g_j(X,x_{n-t},\cdots,x_{n-1})\}_{x_{n-t},\cdots,x_{n-1}\in\{0,1\}}$, where the scalars depend only on r. Therefore, for the first $n-d$ round, the prover can compute packed ciphertexts of $\{g_i(X,r)\}_{X\in\{0,1\}^d}$ by performing scalar multiplications directly on the packed ciphertexts $\{\mathrm{ct}_j\}$, which allows the prover to compute f_i efficiently. For better efficiency, we let the sumcheck protocol terminate in the $n-d$-th layer, allowing the verifiers to obtain complete descriptions of f_{n-d} and verify its correctness through a single query to $O(f)$, adding an acceptable additional cost to the verifier. The formalized construction is defered to Appendix E.

From encrypted relations to queries on polynomials. In what follows, for any vector x, we denote by $\tilde{x}$ the multilinear polynomial whose representation satisfies $\tilde{x}(i)=x[\mathrm{int}(i)]$ for $i\in\{0,1\}^{\log|x|}$. Furthermore, for any $n\times n$ matrix $A=\{a_{i,j}\}$, denote by $\tilde{A}$ the $2\log n$-variate multilinear polynomial whose representation satisfies $\tilde{A}(i,j) := a_{\mathrm{int}(i),\mathrm{int}(j)}$ for all $i,j\in\{0,1\}^{\log n}$.

To conclude the construction and reduce the verification of encrypted instance to queries on encrypted polynomials, we can homomorphically execute the well-known Spartan's PIOP protocol.

Overview of Spartan's PIOP. In the Spartan's PIOP, the prover first provides the oracle of $\tilde{z}$ to the verifier. Then, up to a negligible soundness error, checking whether $Az \circ Bz = Cz$ is equivalent to checking whether the following identity holds with the oracle access to $\tilde{z}$:

$$0 = \Sigma_{x\in\{0,1\}^n} eq(x,\tau)\cdot F(x)$$

where $F(x) = \Sigma_{y\in\{0,1\}^n}\tilde{A}(x,y)\cdot\tilde{z}(y) \times \Sigma_{y\in\{0,1\}^n}\tilde{B}(x,y)\cdot\tilde{z}(y) - \Sigma_{y\in\{0,1\}^n}\tilde{C}(x,y)\cdot\tilde{z}(y)$. We write $F(x) = \Sigma_{y\in\{0,1\}^n} f(x,y)$.

Both parties run the sumcheck protocol twice. The first one reduces above verification to the queries of $eq(r,\tau)$, which can be computed directly, and $F(r)$. The second one reduces the checking of $F(r) = \Sigma_{y\in\{0,1\}^n} f(r,y)$ to the queries of encrypted polynomial $\tilde{z}$ (and the plaintext polynomials $\tilde{A},\tilde{B},\tilde{C}$, of which oracles are provided in preprocession). And finally, the verifier checks the public input is consistent with $\tilde{z}$ through an additional query to $\tilde{z}$. As a result, the checking is finally reduced to the queries to the oracle of encrypted polynomial $\tilde{z}$ (and the oracles of plaintext polynomials $\tilde{A},\tilde{B},\tilde{C}$, which are provided by the algorithm $\mathcal{I}$).

The *homomorphic encrypted version* of Spartan's PIOP can be constructed via a straightforward strategy: the prover attempts to return all elements in plaintext unless it fails to do so and must sent encrypted elements. In addition, we use the sumcheck protocol constructed above for encrypted polynomials as underlying sumcheck protocol for better efficiency and communications. The formalized construction is deferred to Appendix E.

SNARKs for Encrypted Instances. Now we can combine the homomorphic encrypted version of Spartan's PIOP protocol and polynomial commitment with (encrypted) polynomial commitments to obtain a public-coin argument of knowledge protocol for encrypted instance. To be more detailed, the polynomial oracle of $\tilde{A},\tilde{B},\tilde{C}$ is achieved through standard polynomial commitment and the polynomial oracle of $\tilde{z}$ is instantiated via polynomial commitment on encrypted polynomials. The construction is deferred to Appendix E.

Theorem 4.4. *The protocol constructed above is an argument of knowledge for encrypted instances.*

proof sketch. Our construction fits within the paradigm of "PIOP + polynomial commitment" and its homomorphic encrypted version[33]. Therefore, the completeness is straightforward and the knowledge soundness security follows from the origin PIOP and polynomial commitment scheme because any adversary against our scheme can be easily transformed an adversary against its plaintext version construction via decrypting all sent messages. □

Non-interactive, Fiat-Shamir heuristic and zero-knowledge. Our construction is public coin, enabling the use of the Fiat-Shamir heuristic [27] to transform it into a non-interactive protocol by employing a hash function to generate challenges. In the random oracle model, honest verifier zero-knowledge can be transformed into standard zero-knowledge, and knowledge soundness is preserved after the transformation (as is partial knowledge soundness). The (knowledge) soundness error of the resulting protocol can be analyzed using the (HE version of the) BCS transform[5, 31]. Furthermore, since our construction can be viewed as an encrypted Spartan scheme (with Ligero/Shockwave polynomial commitment), it exhibits the same knowledge error. Similar to Brakedown (and Spartan), our construction can be made zero-knowledge using standard techniques with minimal overhead[1, 19, 38] (except that we additionally require that the HE scheme can be re-randomized using noise flooding to avoid the leakage in the noise of encryption).

Multiplication-Depth and Compatibility with FHE Packing Techniques. The primary multiplication depth costs arise from computing Az, Bz, Cz, and the subsequent two sumcheck protocols. By setting the depth of underlying polynomial commitment on encrypted polynomials as 3 (meaning using a 3-layer NTT), the entire multiplication-depth of our construction is 4. Remind that both the underlying polynomial commitment scheme and the sumcheck protocol are compatible with FHE packing. The resulting SNARK scheme also exhibits excellent compatibility with FHE packing technique. Specifically, given the unpacked ciphertexts contained in the statement and witness, the prover first computes Az, Bz, Cz (since A, B, C are sparse matrices, the computation of Az, Bz, Cz only involves $O(n)$ scalar multiplications). Then, the prover represents z and Az, Bz, Cz as matrices, packs the ciphertexts in each column into a single ciphertext, and runs both the underlying polynomial commitment scheme and the sumcheck protocol with these packed ciphertexts. Since packing requires additional depth the resulting scheme now has a depth of 5.

4.4 Construction of VCoed

In this section, we demonstrate how to construct verifiable computation using the primitives constructed in previous sections. Our construction is relatively straightforward. The client uses FHE to encode its input, and the server leverages the homomorphic evaluation capabilities of FHE to perform the computation and generates a proof using our newly constructed SNARK for encrypted relations. Since the encrypted version of SNARKs can only achieve the extractibility of witness with secret key, we require the prover to send a plaintext polynomial commitment of its input and prove that the committed input is consistent with the corresponding part of the committed encrypted polynomials $\tilde{z}$ through additional queries to underlying polynomial commitments (in a similar way as the checking of consistence of public input). The formal construction is shown in Fig.7.

Theorem 4.5. *The protocol puts forward in Figure.7 is a VCoed with client-privacy and knowledge soundness.*

proof sketch. Completeness follows directly from our construction and the client-privacy follows directly from the CPA security of the FHE scheme. As for the proof of knowledge soundness, the extractor $\mathcal{E}$ can be constructed using the extractor $\mathcal{E}_{pcs}$ of underlying polynomial commitment cmt_y. Since cmt_y is a commitment in plaintext, the extraction does not need secret key. Now we show that the extracted y is valid when server convinces client. First, the knowledge soundness of underlying SNARK ensure that the committed polynomial $\tilde{z}$ satisfying that $Az \cdot Bz = Cz$. Write $z := ((w^*, y'), \mathbb{x})$, then we have that $\tilde{z}(0,\cdots,0,1,r)) = \tilde{y'}(r)$. From Schwartz-Zippel lemma and FS-heuristic, $\tilde{y'}(r) = \tilde{y}(r)$ for a random r ensures $y' = y$, therefore we have that $C(x,y) = \text{out}$. □

Verifiable Computation

VC.Setup$(1^\lambda, \mathsf{C})$: Arithmetize the circuit satisfiability of C as an R1CS relation $(\mathbb{i}, \mathbb{w} := (w^*, y), \mathbb{x} = (x, \mathrm{out}))$ where $\mathrm{out} = C(x, y)$ and w^* denotes intermediate values appeared during the computation of C. W.l.o.g., assume that $|y| = |\mathbb{x}| = 2^k$, $|z = (\mathbb{w}, \mathbb{x})| = 2^n$ for some $k, n \in \mathbb{N}$ (this can always be achieved by padding with an appropriate number of zeros). Run the setup algorithm and the deterministic algorithm $\mathcal{I}(1^\lambda, \mathbb{i})$ of our SNARKs to generate crs and the proving/verifying parameters $(\mathsf{pp}, \mathsf{vp})$. Run the key generation algorithm of FHE encryption and get $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Key.Gen}(1^\lambda)$. Output $\mathsf{PK} = (crs, \mathsf{pk}, \mathsf{pp})$, $\mathsf{SK} = (crs, \mathsf{sk}, \mathsf{vp})$.

VC.ProbGen(PK, x): Encrypt its input $c_1 = \mathsf{Enc}_{\mathsf{pk}}(x)$ and output $\mathsf{Enc}(x) = c_1$ and $\mathsf{st} = x$.

VC.Compute$(\mathsf{PK}, \mathsf{Enc}(m), \mathsf{C}, y)$: Compute the output of the circuit $\mathsf{Enc}_{\mathsf{pk}}(\mathrm{out}) \leftarrow \mathsf{Eval}_{\mathsf{pk}}(C(\cdot, y), \mathsf{Enc}_{\mathsf{pk}}(x))$. Retrieve encrypted witness $\mathbb{w}$ directly from above computation. Generate the proof π through a proof generation algorithm constructed as follows:

(1) Compute the proof π_1 for $(\mathbb{i}, (\mathsf{Enc}(\mathbb{w}), \mathsf{Enc}(\mathbb{x}))$ using our SNARK scheme for encrypted relations.
(2) Compute the polynomial commitment cmt_y for $\tilde{y}$ in plaintext.
(3) Let cmt_x be the polynomial commitment for encrypted polynomial $\tilde{z}$ where $z := ((w^*, y), (x, \mathrm{out}))$ contained in π_1. Choose challenge $r \in \mathbb{F}^k$ through hashing the transcript and evaluate (encrypted) polynomials $\tilde{y}, \tilde{z}$ on $r \in \mathbb{F}^p$ and $(0, \cdots, 0, 1, r) \in \mathbb{F}^n$ to show that $\tilde{z}(0, \cdots, 0, 1, r) = \tilde{y}(r)$. Write the proofs of evaluation as π_2, π_3.
(4) Output $\mathsf{Enc}_{\mathsf{pk}}(\mathrm{out})$ and $\pi = (\pi_1, \pi_2, \pi_3, \mathsf{cmt}_y)$

VC.Verify$(\mathsf{SK}, \mathsf{Enc}_{\mathsf{pk}}(\mathrm{out}), \mathsf{st}, \pi)$: Check the validness of all proofs in π. If all the proofs are valid and $\tilde{z}(0, \cdots, 0, 1, r)) = \tilde{y}(r)$, output out by decrypting $\mathsf{Enc}_{\mathsf{pk}}(\mathrm{out})$. Otherwise, output $\perp$.

Figure 7: Verifiable Computation on Encrypted Data

Optimization. Note that the proof of VCoed consists of three polynomial evaluations to the encrypted polynomial $\tilde{z}$, resulting in significant communication overhead. However, by applying standard aggregation techniques (see [61], Section 4.5.2 for details), these can be compressed into a single evaluation without increasing the multiplicative depth. Furthermore, we offer an advanced solution in the Appendix D that modifies the polynomial commitment construction to commit to $\tilde{z}$ while enabling extractability of y without requiring the secret key. (When using this modified polynomial commitment for $\tilde{z}$, we can eliminate π_2, π_3, and cmt_y from the proof π.) This approach further reduces both computational and communication costs, albeit slightly.

Server-privacy. Our construction can be modified easily to achieve server-privacy against an honest client by using standard blind technique for SNARK[1, 19, 38] to avoid leakage in the polynomial commitment and PIOP and re-randomizing the noise of the ciphertexts in output and proof to avoid leakage in the noise of encryption. To achieve server-privacy against a malicious client, we require the client to prove the correctness of its encoding of the input, and the key pairs of the underlying FHE scheme can also be generated by the client if they prove the correctness of the key generation process as well.

5 Implementation and Comparison

We then demonstrate the practicality of Phalanx, with a primary focus on proof generation time. We conduct comparative analyses against existing FHE-SNARK implementations (or partial implementations) [3, 31]. The evaluation begins with our parameter settings and experimental micro-benchmarks.

BGV/BFV Parameter Setting. In our evaluation, we use $t \approx 2^{50}$ being a prime as our FHE plaintext modulus. Using it, each multiplication (both plaintext and ciphertext) incurs roughly 60 bits of noise, and the scalar multiplication incurs about 50 bits of noise. Since we need at most 5 levels of multiplications (detailed below in Section 5.2), we choose $Q \approx 2^{420}$ as our ciphertext modulus (one additional level for the evaluation key and one level for plaintext itself) with ring dimension $N = 2^{14}$. This guarantees > 128-bit of computational security.

Micro-benchmarks. We test the following micro-benchmarks in SEAL [59] library for the parameters we use (on a GCP instance N4 with CPU Intel Emerald Rapids with 16 GB RAM), which are later used to estimate our overall performance. The following assume $Q \approx 2^{60}$ and for $Q \approx 2^{60 \cdot x}$, the runtime is approximately x times slower.

- Addition: 0.0136ms
- Scalar-by-ciphertext multiplications: 0.0548ms
- Plaintext-by-ciphertext multiplications: 0.61ms
- Ciphertext-by-ciphertext multiplications: 5.0ms
- Rotation/automorphism: 0.89ms
- Decryption: 0.15ms
- SHA256 hash: 131MB/sec

5.1 Polynomial commitment over encrypted values.

For an n-variate multilinear polynomial with its encrypted point-value representation satisfying our requirement, the committing algorithm requires $O(t \cdot 2^{n+\frac{n}{2t}})$ scalar multiplications when using a t-level NTT, and the evaluation requires $O(2^n)$ scalar multiplications and additions for proving.

When using the packing technique of FHE and packing 2^d ciphertexts into a single packed ciphertext (assumes that the input has been packed already), then the committing algorithm requires $O(t \cdot 2^{n-d+\frac{n}{2t}})$ scalar multiplications when using a t-level NTT, and the evaluation requires $O(2^{n-d})$ plaintext-ciphertext inner-product. As a comparison, the FRI-based polynomial commitment used in HELIOPOLIS [3] requires at least $O(2^{n-d})$ plaintext-ciphertext matrix-vector multiplications during the RS encoding of committed polynomials.

Now, we discuss the concrete performance of our polynomial commitment. Suppose that the variate number of committed polynomials is $n = 20$, the rate of underlying Reed-Solomon code is $1/4$ and we adapt 3-layer NTTs. In this case, the multiplication-depth of our commitment is 3. To achieve 80-bit security, we set the number of columns opened in testing phase and evaluation phase to be $\frac{80}{\log(1-\frac{\lambda}{3N})} \approx 193$. To achieve 100-bit security, we set the number to be 240. We run our polynomial commitment with the BFV plaintext modulus, a 50-bit prime. To ensure the 80-bit security, our evaluation protocol is run over the Galois degree-2 extension

security param.	commit	prove[5]	verify	communication
$\lambda = 80$	2^{11} HE_{sca}/level+hash	2^7 HE_{inn}	388 HE_{dec}+V_{hash}+V_{remain}	388(Enc+π_{merkle})
$\lambda = 100$	2^{11} HE_{sca}/level+hash	2^7 HE_{inn}	482 HE_{dec}+V_{hash}+V_{remain}	482(Enc+π_{merkle})

Table 1: Performance of our polynomial commitment on hybrid values. HE_{sca}, HE_{inn} and HE_{dec} mean the cost of single scalar multiplication, plaintext-ciphertext inner-product, and decryption on packed ciphertexts. ·/level means the homomorphic evaluation costs for each level (3 levels in total). V_{hash} denotes the cost for verifying the Merkle-hash proofs and V_{remain} denotes the remaining minimal cost only consisting several multiplications on plaintexts. Enc and π_{merkle} means the length of single ciphertext and Merkle-hash proofs.

field of this field. For 100-bit security, our evaluation protocol is run over the Galois degree-3 extension field of this field. Note that our commitment algorithm still runs over the original field instead of the extension field. This is because a Reed-Solomon code over $\mathbb{F}_p$ (with encoding matrix $\{e_{i,j}\}_{i,j}$) can be naturally viewed as a Reed-Solomon code over $\mathbb{F}_{p^2}$ (with encoding matrix $\{(0, e_{i,j})\}_{i,j}$, where (a, b) denotes the elements $aX + b \in \mathbb{F}_{p^2}$), and for any encoding $\mathsf{Enc}(m)$ in $\mathbb{F}_p$ for $m \in \mathbb{F}_p$, $(0, \mathsf{Enc}(m)) \in \mathbb{F}_{p^2}$ is exactly the encoding in $\mathbb{F}_{p^2}$ for $(0, m) \in \mathbb{F}_{p^2}$.

The computation and communication costs are shown in Table 1 and the concrete performance is shown in Table 2.

security param.	commit	prove	verify	communication
$\lambda = 80$	1.33s	6.78s	461ms	46.8MB
$\lambda = 100$	1.33s	10.18s	573ms	58.1MB

Table 2: Single core performance of our polynomial commitment on hybrid values.

Comparison. Since HELIOPOLIS [3] only implements homomorphic evaluation for FRI, we then provide a comparison of polynomial commitment part. This comparison is non-trivial due to HELIOPOLIS's limitation to polynomial sizes up to 2^{15}, achieving a 207 second runtime on 32 cores hardware. In addition, Gama et al. [31] evaluated FRI at a 2^{14}-size circuit with a 71-second single-core runtime. In contrast, our approach achieves a commitment time of 1.33 + 10.18 =11.51 seconds at a significantly larger 2^{20}-size circuit. Even when accounting for hardware discrepancies (notably, our test environment uses slower hardware than Gama et al.'s setup, as quantified in Section 5.2), our method demonstrates substantial performance advantages.

5.2 Phalanx: Overall Performance

We then provide the overall performance of our Phalanx instance with 2^{20}-size R1CS instance (A, B, C are $2^{20} \times 2^{20}$ matrixs and for each matrix, there is 2^{20} non-zero entries). Suppose that all the ciphertexts of the prover's input are all unpacked, which is the most general case in applications. The VCoed instance is over a FHE-friend field, $\mathbb{F}_p$, with a 50-bit prime p, and some steps of our construction are need to run over the Galois degree-3 extension field of $\mathbb{F}_p$ to achieve 100-bit security. The prover cost is shown as follows (we only list the operations of FHE since the remaining parts are minimal. We require 5 levels of homomorphic multiplication depth here since the packing consumes one addition level):

(1) For Unpacked $z = (w, x)$, compute Az, Bz, Cz, which costs $3 \cdot 2^{20}$ scalar multiplications over unpacked ciphertexts (level-6 → level-5). The plaintext is over $\mathbb{F}_p$ and costs 1034s.
(2) Packing: for z, Az, Bz, Cz (containing $4 \cdot 2^{20}$ ciphertexts in total), pack 2^{14} ciphertexts into a single ciphertext resulting 2^8 packed ciphertexts, using $4 \cdot 2^{20}$ plaintext-ciphertext multiplications[6] (Level-5 → level-4). The plaintext is over $\mathbb{F}_p$ and costs 7107s.
(3) Sumcheck for $\Sigma_x eq(x, \tau)F(x)$: 1). [level-4 → level-3]: 9×2^9 scalar multiplications and additions. 2). [level-3 → level-2]: 2^8 ciphertext-ciphertext multiplications and additions. 3). [level-2 → level-1]: 2^8 plaintext-ciphertext vector multiplications and additions. The plaintext is over the Galois degree-3 extension field and cost 5.67, 34.56, 2.81 seconds respectively.
(4) Sumcheck for $\Sigma_x M(r, x) \cdot z(x)$: 1). [level-3 → level-2]: 9×2^7 scalar multiplications and additions. 2). [level-2 → level-1]: 3×2^6 plaintext-ciphertext vector multiplications and additions. These step run over the Galois degree-3 extension field and cost 2.12, 2.76 seconds respectively.
(5) Polynomial commitment for $\tilde{z}$ with 1 query (level-4 → level-1).

The communication mainly consists of 504 packed ciphertexts for 100-bit security. The communication also consists several Merkle-hash proof, which is minimal compares to the size of ciphertexts.

The verification consists of the decryptions of all received packed ciphertexts, the verification of underlying polynomial commitment on hybrid values and some other verification contained in the standard SNARKs for plaintext instance.

Phalanx*. The latency in Phalanx primarily stems from the iNTT (inverse Number Theoretic Transform) operation executed in step (2). Notably, since the input of Phalanx comes from homomorphic circuit evaluation results, these values could inherently reside in iNTT format. This could substantially reduce our scheme's runtime. We denote the algorithm in such scenario as Phalanx*, indicating the best-case performance under the iNTT-formatted input assumption. Conversely, the baseline Phalanx represents worst-case performance.

In total, the estimated performance is shown in Table 3.

	security param.	prove	verify	communication
Blind-SNARK [31]	$\lambda = 100$	16.57hrs	–	–
Our Phalanx	$\lambda = 100$	2.27hrs (7.3×)	2.8s	61.4MB
Our Phalanx*	$\lambda = 100$	0.68 hrs (24.4×)	2.8s	61.4MB

Table 3: Single core performance of our polynomial commitment on hybrid values.

[5]Note that here we also pack the results after the inner-products. For clarify, we do not include them in the table, but esssentially, it is one rotation plus one plaintext multiplication, which is much cheaper than a single inner-product, and we include them in our runtime estimation.

[6]We perform one additional optimization to this step: before doing multiplications, we transform all the ciphertexts into iNTT form, and then perform the packing process (multiply by a plaintext to extract the slots and sum them up). Lastly, we transform the summed up ciphertexts back to its NTT form.

Remark. To eliminate experimental discrepancies caused by hardware differences, we replicated Blind-SNARK's macro-benchmarks on our hardware to derive a hardware scaling factor of 1.79[7]. By applying this scaling factor to their proposed results, we obtained revised performance for equitable comparison. Without revision, their raw result is 9.26hrs.

Multi-threading. Our estimated performance uses only a single CPU core while the scheme is highly parallelizable: our analysis in this section implicitly indicates that most FHE operations can be parallelized by a factor up to 2^8 (256). On a 256-core server, the FHE operations in Phalanx take only 32sec, bringing the total runtime to <1min; for Phalanx*, the FHE operations take 10sec, making the total runtime to <30sec. Even on a 64-core server, Phalanx* finishes in <1min. In comparison, Blind-SNARK[29] requires >15min with 64 cores.

References

[1] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 2087–2104. ACM (2017). https://doi.org/10.1145/3133956.3134104, https://doi.org/10.1145/3133956.3134104

[2] Angel, S., Chen, H., Laine, K., Setty, S.T.V.: PIR with compressed queries and amortized query processing. In: 2018 IEEE Symposium on Security and Privacy. pp. 962–979. IEEE Computer Society Press, San Francisco, CA, USA (May 21–23, 2018)

[3] Aranha, D.F., Costache, A., Guimarães, A., Soria-Vazquez, E.: HELIOPOLIS: verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. In: Chung, K., Sasaki, Y. (eds.) Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part V. Lecture Notes in Computer Science, vol. 15488, pp. 302–334. Springer (2024). https://doi.org/10.1007/978-981-96-0935-2_10, https://doi.org/10.1007/978-981-96-0935-2_10

[4] Atapoor, S., Baghery, K., Pereira, H.V.L., Spiessens, J.: Verifiable FHE via lattice-based snarks. IACR Commun. Cryptol. **1**(1), 24 (2024). https://doi.org/10.62056/A6KSDKP10, https://doi.org/10.62056/a6ksdkp10

[5] Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A.D. (eds.) Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9986, pp. 31–60 (2016). https://doi.org/10.1007/978-3-662-53644-5_2, https://doi.org/10.1007/978-3-662-53644-5_2

[6] Bois, A., Cascudo, I., Fiore, D., Kim, D.: Flexible and efficient verifiable computation on encrypted data. In: Garay, J.A. (ed.) Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12711, pp. 528–558. Springer (2021). https://doi.org/10.1007/978-3-030-75248-4_19, https://doi.org/10.1007/978-3-030-75248-4_19

[7] Bourse, F., Izabachène, M.: Plug-and-play sanitization for TFHE. Cryptology ePrint Archive, Paper 2022/1438 (2022), https://eprint.iacr.org/2022/1438

[8] Bourse, F., Pino, R.D., Minelli, M., Wee, H.: FHE circuit privacy almost for free. Cryptp 2016, https://eprint.iacr.org/2016/381

[9] Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417. p. 868–886. Springer-Verlag (2012)

[10] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)

[11] Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd FOCS. pp. 97–106. IEEE Computer Society Press (Oct 22–25, 2011)

[12] Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg, Germany (Aug 14–18, 2011)

[13] Checri, M., Sirdey, R., Boudguiga, A., Bultel, J.P.: On the practical cpad security of "exact" and threshold fhe schemes and libraries. In: Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part III. p. 3–33. Springer-Verlag, Berlin, Heidelberg (2024)

[14] Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14005, pp. 499–530. Springer (2023). https://doi.org/10.1007/978-3-031-30617-4_17, https://doi.org/10.1007/978-3-031-30617-4_17

[15] Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 1223–1237. ACM Press, Toronto, ON, Canada (Oct 15–19, 2018). https://doi.org/10.1145/3243734.3243836

[16] Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1243–1255. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017). https://doi.org/10.1145/3133956.3134061

[17] Cheon, J.H., Choe, H., Passelègue, A., Stehlé, D., Suvanto, E.: Attacks against the ind-cpad security of exact fhe schemes. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 2505–2519 (2024)

[18] Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017)

[19] Chiesa, A., Forbes, M.A., Spooner, N.: A zero knowledge sumcheck and its applications. Electron. Colloquium Comput. Complex. **TR17-057** (2017), https://eccc.weizmann.ac.il/report/2017/057

[20] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016. pp. 3–33. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)

[21] Chillotti, I., Gama, N., Goubin, L.: Attacking fhe-based applications by software fault injections. Cryptology ePrint Archive (2016)

[22] Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled PSI from homomorphic encryption with reduced computation and communication. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 1135–1150. ACM Press, Virtual Event, Republic of Korea (Nov 15–19, 2021). https://doi.org/10.1145/3460120.3484760

[23] Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled PSI from homomorphic encryption with reduced computation and communication. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021. pp. 1135–1150. ACM (2021). https://doi.org/10.1145/3460120.3484760, https://doi.org/10.1145/3460120.3484760

[24] Dalvi, A., Jain, A., Moradiya, S., Nirmal, R., Sanghavi, J., Siddavatam, I.: Securing neural networks using homomorphic encryption. In: 2021 International Conference on Intelligent Technologies (CONIT). pp. 1–7 (2021). https://doi.org/10.1109/CONIT51480.2021.9498376

[25] Ducas, L., Micciancio, D.: FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. pp. 617–640. Springer, Berlin, Heidelberg (2015)

[26] Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), https://ia.cr/2012/144

[27] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986). https://doi.org/10.1007/3-540-47721-7_12, https://doi.org/10.1007/3-540-47721-7_12

[28] Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: Ahn, G., Yung, M., Li, N. (eds.) Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014. pp. 844–855. ACM (2014). https://doi.org/10.1145/2660267.2660366, https://doi.org/10.1145/2660267.2660366

[29] Fiore, D., Nitulescu, A., Pointcheval, D.: Boosting verifiable computation on encrypted data. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12111, pp. 124–154. Springer (2020). https://doi.org/10.1007/978-3-030-45388-6_5, https://doi.org/10.1007/978-3-030-45388-6_5

[7] This factor was calculated as the ratio of ciphertext multiplication runtime under identical parameters in the SEAL library (our runtime / their reported time).

[30] Fisch, B., Lazzaretti, A., Liu, Z., Papamanthou, C.: Thorpir: Single server pir via homomorphic thorp shuffles. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. p. 1448–1462. CCS '24, Association for Computing Machinery, New York, NY, USA (2024)

[31] Gama, M., Beni, E.H., Kang, J., Spiessens, J., Vercauteren, F.: Blind zksnarks for private proof delegation and verifiable computation over encrypted data. IACR Cryptol. ePrint Arch. p. 1684 (2024), https://eprint.iacr.org/2024/1684

[32] Ganesh, C., Nitulescu, A., Soria-Vazquez, E.: Rinocchio: Snarks for ring arithmetic. J. Cryptol. **36**(4), 41 (2023). https://doi.org/10.1007/S00145-023-09481-3, https://doi.org/10.1007/s00145-023-09481-3

[33] Garg, S., Goel, A., Wang, M.: How to prove statements obliviously? In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X. Lecture Notes in Computer Science, vol. 14929, pp. 449–487. Springer (2024). https://doi.org/10.1007/978-3-031-68403-6_14, https://doi.org/10.1007/978-3-031-68403-6_14

[34] Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6223, pp. 465–482. Springer (2010). https://doi.org/10.1007/978-3-642-14623-7_25, https://doi.org/10.1007/978-3-642-14623-7_25

[35] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009. pp. 169–178. ACM (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440

[36] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg, Germany (Aug 18–22, 2013)

[37] Goldreich, O.: The Foundations of Cryptography - Volume 2: Basic Applications. Cambridge University Press (2004). https://doi.org/10.1017/CBO9780511721656, http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol2.html

[38] Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic snarks for R1CS. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14082, pp. 193–226. Springer (2023). https://doi.org/10.1007/978-3-031-38545-2_7, https://doi.org/10.1007/978-3-031-38545-2_7

[39] Guruswami, V., Sudan, M.: Improved decoding of reed-solomon and algebraic-geometry codes. IEEE Trans. Inf. Theory **45**(6), 1757–1767 (1999). https://doi.org/10.1109/18.782097, https://doi.org/10.1109/18.782097

[40] Halevi, S., Shoup, V.: Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481 (2020), https://eprint.iacr.org/2020/1481

[41] Halevi, S., Shoup, V.: Bootstrapping for HElib. Journal of Cryptology **34**(1), 7 (Jan 2021)

[42] Hwang, I., Min, S., Song, Y.: Practical circuit privacy/sanitization for TFHE. Cryptology ePrint Archive, Paper 2025/216 (2025), https://eprint.iacr.org/2025/216

[43] Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A low latency framework for secure neural network inference. In: Enck, W., Felt, A.P. (eds.) USENIX Security 2018. pp. 1651–1669. USENIX Association (Aug 15–17, 2018)

[44] Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. In: ASIACRYPT 2021. p. 608–639. Springer (2021)

[45] Kluczniak, K.: Circuit privacy for fhew/tfhe-style fully homomorphic encryption in practice. cic **1, Issue 4** (2025). https://doi.org/10.62056/av11c3w9p, https://cic.iacr.org/p/1/4/33

[46] Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., No, J.S.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. IEEE Access **10**, 30039–30054 (2022). https://doi.org/10.1109/ACCESS.2022.3159694

[47] Lee, J.W., Lee, E., Kim, Y.S., No, J.S.: Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology – ASIACRYPT 2023. pp. 36–68. Springer Nature Singapore, Singapore (2023)

[48] Lee, K., Yeo, Y.: SophOMR: Improved oblivious message retrieval from SIMD-aware homomorphic compression. Cryptology ePrint Archive, Paper 2024/1814 (2024), https://eprint.iacr.org/2024/1814

[49] Li, B., Micciancio, D.: On the security of homomorphic encryption on approximate numbers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 648–677. Springer (2021)

[50] Li, B., Micciancio, D., Schultz-Wu, M., Sorrell, J.: Securing approximate homomorphic encryption using differential privacy. In: Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part I. p. 560–589. Springer-Verlag, Berlin, Heidelberg (2022)

[51] Lindell, Y. (ed.): Tutorials on the Foundations of Cryptography. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-57048-8, https://doi.org/10.1007/978-3-319-57048-8

[52] Liu, J., Li, J., Wu, D., Ren, K.: Pirana: Faster multi-query pir via constant-weight codes. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 4315–4330. IEEE (2024)

[53] Liu, Z., Tromer, E.: Oblivious message retrieval. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part I. LNCS, vol. 13507, pp. 753–783. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–18, 2022)

[54] Liu, Z., Tromer, E., Wang, Y.: Group oblivious message retrieval. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 4367–4385 (2024)

[55] jie Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: 2021 IEEE Symposium on Security and Privacy. pp. 1057–1073. IEEE Computer Society Press (May 24–27, 2021)

[56] Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. In: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I. pp. 2–10. IEEE Computer Society (1990). https://doi.org/10.1109/FSCS.1990.89518, https://doi.org/10.1109/FSCS.1990.89518

[57] Menon, S.J., Wu, D.J.: SPIRAL: Fast, high-rate single-server PIR via FHE composition. In: 2022 IEEE Symposium on Security and Privacy. pp. 930–947. IEEE Computer Society Press, San Francisco, CA, USA (May 22–26, 2022)

[58] Parno, B., Raykova, M., Vaikuntanathan, V.: How to delegate and verify in public: Verifiable computation from attribute-based encryption. In: Cramer, R. (ed.) Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7194, pp. 422–439. Springer (2012). https://doi.org/10.1007/978-3-642-28914-9_24, https://doi.org/10.1007/978-3-642-28914-9_24

[59] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL (Jan 2023), microsoft Research, Redmond, WA.

[60] Setty, S.T.V.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12172, pp. 704–737. Springer (2020). https://doi.org/10.1007/978-3-030-56877-1_25, https://doi.org/10.1007/978-3-030-56877-1_25

[61] Thaler, J.: Proofs, Arguments, and Zero-Knowledge (2022)

[62] Thibault, L.T., Walter, M.: Towards verifiable FHE in practice: Proving correct execution of tfhe's bootstrapping using plonky2. IACR Cryptol. ePrint Arch. p. 451 (2024), https://eprint.iacr.org/2024/451

[63] Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Lecture Notes in Computer Science, vol. 4948, pp. 1–18. Springer (2008). https://doi.org/10.1007/978-3-540-78524-8_1, https://doi.org/10.1007/978-3-540-78524-8_1

[64] Viand, A.: Useable Fully Homomorphic Encryption. Ph.D. thesis, ETH Zurich, Zürich, Switzerland (2023). https://doi.org/10.3929/ETHZ-B-000613734, https://hdl.handle.net/20.500.11850/613734

[65] Viand, A., Knabenhans, C., Hithnawi, A.: Verifiable fully homomorphic encryption. arXiv preprint arXiv:2301.07041 (2023)

[66] Zeilberger, H., Chen, B., Fisch, B.: Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X. Lecture Notes in Computer Science, vol. 14929, pp. 138–169. Springer (2024). https://doi.org/10.1007/978-3-031-68403-6_5, https://doi.org/10.1007/978-3-031-68403-6_5

[67] Zhang, Z., Plantard, T., Susilo, W.: Reaction attack on outsourced computing with fully homomorphic encryption schemes. In: International Conference on Information Security and Cryptology. pp. 419–436. Springer (2011)

A The Gap Between VCoed and 2PC

While VCoed developed rapidly, there are several aspects remain insufficiently discussed. First, early VCoed addresses scenarios where server performs computation over client's input. But server may have its own input in real-world applications, where prior work [3] hinted the potential disparities. Such a case appears to resemble secure two-party computation (2PC), so the first question arise:

(1) Is there any new challenges for VCoed to deal with server inputs? Furthermore, is there a deep relation between VCoed (with server inputs) and 2PC?

Furthermore, it remains unclear why the two seemingly similar VCoed approaches—one employing SNARKs outside FHE and the other leveraging FHE outside SNARKs—result in a significant performance gap. An intuition is that the more performant one is weaker than the less efficient one. Thus, this raises our second question:

(2) Do FHE-SNARK and SNARK-FHE offer equivalent VCoed Functionality?

Despite numerous existing works[3, 28, 34, 58] and community discussions (implicitly) address these issues and offering partial answers, a more formal analysis and a more comprehensive discussion remain valuable. This paper addresses aforementioned questions by proposing a more comprehensive model for verifiable computation as described in Figure.3. First, we demonstrate that in VCoed with server inputs, the server can launch additional attacks (we formalize it as input-dependent attack, where malicious servers adaptively forge inputs based on encrypted client data) on clients despite the adherence to VCoed security properties in prior works. To mitigate these potential threats, we introduce a security concept (adapted from SNARK) called knowledge soundness, which aims to ensure that the server does not abuse the clients' inputs.

Next, through formal security reduction analysis, we establish an intrinsic connection between VCoed and secure two-party computation (2PC). Our analysis reveals that VCoed with knowledge soundness still has a 1-bit information loss compared to 2PC. We clarify it by formalizing a 1-bit forge attack and propose Theorem.A.1 to establish that VCoed with strong client privacy (where the server can query the "verify" oracle once) is *equivalent* to 2PC.

Having formalized these concepts, we answer the second question as follows: FHE-SNARK is applicable for achieving VCoed, whereas SNARK-FHE fits naturally within the 2PC model, rendering it comparatively stronger.

A.1 VCoed Needs Knowledge Soundness

When both the client and server have (private) inputs—a scenario common to many applications [7, 8, 42, 45], including PSI, OPRF, PPML, and others—new challenges arise regarding client security. Our discussion begins with the *inadequacy* of soundness in such cases. While soundness ensures the correctness of computation, it only ensures the existence of y rather than the knowledge of y. In other words, the malicious server might complete the computation and proof without actually knowing y. This allows the server to make harmful attacks and change the output of the computation according to client's input.

To illustrate this point more concretely, we first present a construction that satisfies the previously defined soundness. Then, we introduce an attack method we refer to as "input dependent attack".

Strawman construction. We can easily construct a soundness verifiable computation scheme using a fully homomorphic encryption scheme and a proof system. Specifically, the client takes its input, encrypts it using the FHE key, and sends the encrypted data to the server. The server performs the desired homomorphic computation on the encrypted input according to the given circuit. It also evaluates the corresponding proof homomorphically which verifies the correctness of the circuit computation. The client decrypts the outputs using its FHE secret key and verifies the provided proof to ensure the computation was performed correctly.

Input-dependent attack. In this section, we introduce our attack to illustrate why the strawman construction fails when the server has a private input. The core idea is straightforward: since the client is unaware of the server's input, the server can use an arbitrary input to the circuit, which may depend on the client's input. Consider a bilateral auction scenario as a simple yet catastrophic example (also a classic example in 2PC illustrating input independence[51]). The client has its bid, x, and the server possesses a bid, y. The server computes a circuit as follows: $\mathsf{C}(x, y) := x > y ? 1 : 0$. This means the circuit outputs 1 if the client's bid is greater and 0 otherwise. Although the honest server can compute the output via $\mathsf{Enc}(\text{out}) \leftarrow \mathsf{Eval}(\mathsf{C}(\cdot, y), \mathsf{Enc}(x))$, a malicious server can exploit this by using a ciphertext $y' = \mathsf{Enc}(x{+}1)$ instead of the honest plaintext y, and then computing $\mathsf{Enc}(\text{out}) \leftarrow \mathsf{Eval}(\mathsf{C}(\cdot, \cdot), (\mathsf{Enc}(x), y'))$. This manipulation ensures that the server always wins. Additionally, since the proof is generated over $x, \mathsf{Dec}(y')$ homomorphically, the proof is always valid, even though the server does not know x.

Another advanced example is unbalanced PSI[23]. In such a scenario, a malicious server could inject fake ciphertexts (e.g., client input ciphertexts) to increase the intersection size or steal client data via results.

More generally, the server can craft a function f, and compute $y' \leftarrow f(\mathsf{Enc}(x))$ homomorphically to generate the malicious input y'. The server then computes $\mathsf{Eval}(\mathsf{C}(\cdot, \cdot), (\mathsf{Enc}(x), f(\mathsf{Enc}(x))))$, and returns the result to the client. Because y' is a valid encryption, the Verify process returns 1. Clearly, this undermines the integrity that we aim to provide, thus making the model unsuitable for such cases. Therefore, we define *knowledge soundness* for VCoed as follows.

- Knowledge soundness: A verifiable computation system is knowledge sound if there exists a probabilistic polynomial time oracle machine $\mathcal{E}$ called the extractor such that for any PPT adversary $\mathcal{A}$, the following probability is negligible,

$$\Pr\left[\begin{array}{l|r} \mathsf{VC.Verify}(\mathsf{sk}, c, \mathsf{st}, \pi) = \text{out} \neq \perp \ \wedge & (\mathsf{SK},\mathsf{PK}) \leftarrow \mathsf{VC.Setup}(1^\lambda, C) \\ \text{out} \neq C(x, y) & (\mathsf{Enc}(x), \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, x) \\ & (c, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{Enc}(x), \mathsf{C}) \\ & y \leftarrow \mathcal{E}^{\mathcal{A}}((\mathsf{pk}, \mathsf{Enc}(x), C), (c, \pi)) \end{array}\right]$$

This knowledge soundness definition is mostly standard, adapted from SNARK, except that the interfaces are replaced by the interfaces in VCoed. Essentially, it simply requires that the server know its input in plaintext such that it cannot perform the attack described above.

A.2 VCoed+Knowledge Soundness $\stackrel{?}{=}$ 2PC

Interestingly, VCoed with private server input closely resembles Secure Two-Party Computation (2PC). A secure 2PC protocol involves two entities jointly computing a deterministic circuit C. In this paper, we focus on the scenario where the computationally limited party, referred to as the client (C), learns the output. The other party, known as the server (S), undertakes the majority of the computational burden. In this setting, the client C has its private input m_1, and the server S has its input m_2. Together, they execute a protocol to compute the output $\mathsf{C}(m_1, m_2)$, with C ultimately obtaining $\mathsf{C}(m_1, m_2)$ upon completion of the execution.

Relation between verifiable computation and 2PC. With the definition clear, we are ready to discuss the relationship between verifiable computation and 2PC. First, it is straightforward to observe that server-privacy is equivalent to simulation security for the server. Therefore, we primarily focus on the security of the client. We start with the following observation:

Claim I: "Client-Privacy + Knowledge Soundness" is weaker than "Simulation security for client".

Construction & Security Analysis. We present a knowledge-soundness VCoed construction, and demonstrate that it is strictly weaker than 2PC simulation security through the identification of a concrete 1-bit forgery attack. While this claim reveals interesting security implications, we note that the vulnerability of FHE to malicious evaluators capable of observing client reactions has been previously documented [21, 65, 67].

In the toy construction with client-privacy and knowledge soundness, the verifier uses a FHE scheme to encrypt its input. The server then uses the ciphertext of its own input, homomorphically computes the intended circuit to obtain the output ciphertext, and homomorphically generates a SNARK proof demonstrating the correctness of the computation. Furthermore, the server computes a proof of knowledge in plaintext, demonstrating the knowledge of the witness used in the generation of the previous proof. For simplicity, consider that the computation is over the binary field. The client outputs $\perp$ if the verification does not pass. This toy construction possesses client privacy and knowledge soundness in a straightforward way.

Attack I (1-bit forge attack): Now, we demonstrate the attack showing that it does not satisfy the simulation security of 2PC.

After honestly computing the result of circuit, the malicious server encrypts a bit 0 and uses it to replace one bit of the client's input. The server then uses this modified input (potentially incorrect) together with the output from the circuit (supposedly correct, since the circuit is evaluated honestly) to generate the remaining proofs. Consequently:

- If the modified bit of was indeed 0, then the proof will pass the verification.
- If the modified bit of was indeed 1, then the proof will not pass the verification.

Therefore, by observing whether the client aborts, the server can learn this bit of the client's input. More generally, the server is able to test the input of the client against a certain value.

Then a further question arises: how much information can the server learn when it knows whether client aborts the protocol? The following observation shows that, the server can learn at most one additional bit about the client's input through knowledge of whether the client accepts the proof.

Claim II: Compared to 2PC, VCoed leakage is bounded by 1 bit.

In the following, we introduce a stronger notion of client-privacy for VCoed, which guarantees the privacy of the client's input even when the prover has access to a one-time verification oracle, which provides one bit of information. This definition is also presented in [28]. We then demonstrate that "client-privacy with a verification oracle" combined with knowledge soundness is as strong as the simulation security for the client in 2PC.

- Client-privacy with verification oracle: In the following game $\mathsf{G}^{\mathsf{S-client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda)$, the advantage of any PPT adversary $\mathcal{A}$, defined as $\mathsf{Adv}^{\mathsf{S-client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda) = \Pr[\mathsf{G}^{\mathsf{S-verifier-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda) \to 1] - 1/2 = \mathsf{negl}(\lambda)$. Here, O_{verify} represents a one-time oracle to the Verify procedure.

$\underline{\mathsf{G}^{\mathsf{S-verifier-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda)}$:

1: $b \xleftarrow{\$} \{0,1\}$
2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathsf{C})$
3: $(m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{PK}, \mathsf{C})$
4: $(c_{m_b}, \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, m_b)$
5: $\hat{b} \leftarrow \mathcal{A}^{O_{verify}}(c_{m_b}, \mathsf{state})$
6: **return** $b \stackrel{?}{=} \hat{b}$

Theorem A.1. *A VCoed scheme satisfying client-privacy with verification oracle and knowledge-soundness is also a 2PC scheme with security for client.*

Proof. We consider a VCoed scheme as a 2PC scheme where the client executes VC.ProbGen and VC.Verify, and the server executes VC.Compute. We now construct the simulator for security as follows:

$\mathcal{S}^{\mathcal{A}}(1^\lambda)$:

(1) After given m_2 as input, run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{VC.Setup}(1^\lambda)$ and encode a zero string $c_1 = \mathsf{Enc}_{\mathsf{pk}}(0^{|m_1|})$.
(2) Invoke the adversary $\mathcal{A}$ with input pk, c_1, m_2 and auxiliary input τ, and obtain its output (c_2, π).
(3) Run the extractor of knowledge soundness, $E^{\mathcal{A}}(\pi)$, to extract the input m_2^* of the adversary.
(4) Send m_2^* to the functionality F.
(5) Verify the proof π using sk, st. If the proof is invalid, the simulator sends abort to the functionality; otherwise, it continues to the next step.
(6) Output the view of adversary $\mathcal{A}$.

Now we need to show that for any m_1, m_2, and τ,

$$\{\mathsf{Real}_{\Pi,\mathcal{A}}(m_1, m_2, \tau)\} \stackrel{c}{\approx} \{\mathsf{Ideal}_{\mathcal{F}_{2pc},S}(m_1, m_2, \tau)\}.$$

We now prove the security by hybrid arguments. In the following, all hybrid experiments are further parameterized by the sender's input m_1, the receiver's input m_2, and the auxiliary input τ.

$\mathsf{HYD}_1(1^\lambda)$ is identical to Ideal except that the simulator $\mathcal{S}$, in step 1, computes $c_1 = \mathsf{Enc}_{pk}(m_1)$ using m_1 instead of $0^{|m_1|}$.

Since the only difference between HYD_1 and Ideal is the message encoded in c_1, we have that $\mathsf{HYD}_1(1^\lambda) \stackrel{c}{\approx} \mathsf{Ideal}_{\mathcal{F}_{2pc},S}(m_1, m_2, \tau)$

due to the client-privacy with verification oracle of the VCoed scheme. It is important to note that, to reduce the client-privacy with verification oracle to the indistinguishability between HYD_1 and Ideal, neither HYD_1 nor Ideal can use the secret key sk during their execution. Therefore, the verification in step 5 is replaced by invoking the verification oracle.

Now, we demonstrate that $\mathsf{HYD}_1(1^\lambda) \overset{c}{\approx} \mathsf{Real}_{\Pi,\mathcal{A}}(m_1, m_2, \tau)$. It is easy to find that, the only difference between these two distributions lies in the output of the client's output. If the verification of π fails, the client will abort in both experiments. If the verification of π passes, the client in $\mathsf{HYD}_1(1^\lambda)$ will output $C(m_1, m_2^*)$ with extracted m_2^* and the client in real experiment will output the decoding of received encoding. From the knowledge soundness of the VCoed scheme, we know that both outputs are identical, which concludes the proof. □

B Cryptographic Primitives

In this section, we provide the formal definitions of the cryptographic primitives used in the main body.

B.1 Secure Two-Party Computation

We follow the the real/ideal world paradigm as outlined in [37] to formalize the definition of 2PC. *Ideal model execution.* Ideal model execution is defined as follows.

- *Input*: Each party receives an input. C receives m_1 and S receives m_2, respectively.
- *Send to trusted party*: C and S send their inputs to a trusted party. An honest party always sends the received input. A malicious party may send a different input.
- *Aborting Adversaries*: An adversarial party can send a message $\perp$ to the trusted party to abort the execution. Otherwise, the following steps are executed.
- *Trusted party answers client C*: Upon receiving inputs m_1, m_2 from C and S respectively, the trusted party sends the output $\mathsf{out} = \mathsf{C}(m_1, m_2)$ to the client.
- *Outputs*: If the client C is honest, then it outputs out. The adversarial party (C or S) outputs its entire view.

We denote the adversary participating in the above protocol to be $\mathcal{B}$ and the auxiliary input to $\mathcal{B}$ is denoted by τ. We define $\mathsf{Ideal}_{\mathcal{F}_{2pc},\mathcal{B}}(m_1, m_2, \tau)$ to be the joint distribution over the outputs of the adversary and the honest party from the ideal execution described above.

Real model execution. We next consider the real model, in which the protocol Π to compute C is executed without trusted third party. In this case, at most one of the two parties (client and server) is controlled by an adversary $\mathcal{A}$. A semi-honest adversary will execute the protocol honestly but attempt to learn the other party's private input. A malicious party may follow an arbitrary feasible strategy, and in particular, may abort the execution at any time. We denote the auxiliary input to $\mathcal{A}$ as τ, and define $\mathsf{Real}_{\Pi,\mathcal{A}}(m_1, m_2, \tau)$ to be the joint distribution over the outputs of the adversary and the honest party from the real execution.

Definition B.1 (2PC Security against Semi-honest/Malicious Adversary). We say Π securely computes C if for every polynomial-size semi-honest/malicious adversary $\mathcal{A}$ in the real world, there exists a polynomial-size adversary $\mathcal{B}$ for the ideal model, such that for any auxiliary input $\tau \in \{0,1\}^*$.

$$\{\mathsf{Real}_{\Pi,\mathcal{A}}(m_1, m_2, \tau)\} \overset{c}{\approx} \{\mathsf{Ideal}_{\mathcal{F}_{2pc},\mathcal{B}}(m_1, m_2, \tau)\}.$$

B.2 B/FV Leveled Homomorphic Encryption

The BFV leveled homomorphic encryption scheme is first introduced in [9] using standard LWE assumption, and later adapted to ring LWE assumption by [26].

Given a polynomial $\in \mathcal{R}_t = \mathbb{Z}_t[X]/(X^N+1)$, the BFV scheme encrypts it into a ciphertext consisting of two polynomials, where each polynomial is from a larger cyclotomic ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N+1)$ for some $Q > t$. We refer t as the plaintext modulus, Q as the ciphertext modulus, and N as the ring dimension. t satisfies that $t \equiv 1 \mod 2N$, where N is a power of two. [8]

Plaintext encoding. In practice, instead of having a polynomial in $\mathcal{R}_t = \mathbb{Z}_t[X]/(X^N+1)$ to encrypt, applications usually have a vector of messages $\vec{m} = (m_1, \ldots, m_N) \in \mathbb{Z}_t^N$. Thus, to encrypt such input messages, BFV first encodes it by constructing another polynomial $y(X) = \sum_{i\in[N]} y_i X^{i-1}$ where $m_i = y(\zeta_j)$, $\zeta_j := \zeta^{3^j} \mod t$, and ζ is the $2N$-th primitive root of unity of t. Such encoding can be done using an Inverse Number Theoretic Transformation (INTT), which is a linear transformation represented as matrix multiplication.

Encryption and decryption. The BFV ciphertext encrypting $\vec{m}$ under $\mathsf{sk} \leftarrow \mathcal{D}$ has the following format: $\mathsf{ct} = (a, b) \in \mathcal{R}_Q^2$, which satisfies $b - a \cdot \mathsf{sk} = \lfloor Q/t \rfloor \cdot y + e$ where $\lfloor Q/t \rfloor \cdot y \in \mathcal{R}_Q$ and y is the polynomial encoded in the way above, and e is a small error term sampled from a Gaussian distribution over $\mathcal{R}_Q$ with some constant standard deviation.

Symmetric key encryption can be done by simply sampling a random a and constructing b accordingly using sk. Public key encryption can also be achieved easily but it is not relevant to our paper so we refer the readers to prior works (e.g., [9, 26, 44]) for details.

Decryption is thus calculating $y' \leftarrow \lceil (t/Q) \cdot (b - a \cdot \mathsf{sk}) \rfloor \in \mathcal{R}_t$ (note that $(b - a \cdot \mathsf{sk})$ is done over $\mathcal{R}_Q$), and then decodes it by applying a procedure to revert the encoding process (which is also a linear transformation). For simplicity, we assume BFV.Dec also embeds the decoding procedure and thus outputs plaintext $y' \in \mathbb{Z}_t^N$ in the decoded form directly (instead of a polynomial $y \in \mathcal{R}_t$). Similarly, we assume BFV.Enc contains the encoding process, thus taking a plaintext $y' \in \mathbb{Z}_t^N$. In addition, define $\mathsf{PartialDec}(\mathsf{sk}, \mathsf{ct} = (a, b) \in \mathcal{R}_Q^2) := b - a \cdot \mathsf{sk} \in \mathcal{R}_Q$ (i.e., decryption without performing the rounding to $\mathcal{R}_t$).

BFV operations. BFV essentially supports addition, multiplication, rotation, and polynomial function evaluation, satisfying the following property:

- (Addition) $\mathsf{BFV.Dec}(\mathsf{ct}_1 + \mathsf{ct}_2) = \mathsf{BFV.Dec}(\mathsf{ct}_1) + \mathsf{BFV.Dec}(\mathsf{ct}_2)$
- (Multiplication) $\mathsf{BFV.Dec}(\mathsf{ct}_1 \times \mathsf{ct}_2) = \mathsf{BFV.Dec}(\mathsf{ct}_1) \times \mathsf{BFV.Dec}(\mathsf{ct}_2)$
- (Rotation) $\mathsf{BFV.Dec}(\mathsf{rot}(\mathsf{ct}, j))[i] = \mathsf{BFV.Dec}(\mathsf{ct})[i+j \pmod N]$, $\forall i, j \in [N]$

[8] Note that this is the relationship between t, N does not need to be satisfied in general (e.g., see [40, 41] for the general encoding). However, throughout our paper, we suppose it holds to maximize the concrete efficiency and thus introduce it this way for simplicity.

- (Polynomial evaluation) $\mathsf{BFV.Dec}(\mathsf{BFV.Eval}(\mathsf{ct}, f)) = f(\mathsf{BFV.Dec}(\mathsf{ct}))$, where $f : \mathbb{Z}_t \to \mathbb{Z}_t$ is a polynomial function. Note that this is implied by addition and multiplication.
- (Vector-matrix multiplication) $\mathsf{BFV.Dec}(\mathsf{ct} \times A) = \mathsf{BFV.Dec}(\mathsf{ct}) \times A$, where $A \in \mathbb{Z}_t^{N \times D}$ for any $D > 0$.

All operations are operated over the entire plaintext vector $m \in \mathbb{Z}_t^N$ (element-wise). Thus, all messages need to be evaluated using the same polynomial f by default. This is also known as the Single Instruction Multiple Data (SIMD) property of BFV. Note that vector-matrix multiplication can be realized using scalar multiplication (implied by addition) and rotation. All of these BFV operations are used as blackboxes in our main constructions and we refer the readers to [9, 26, 40, 41, 44] to see how these operations are accomplished in detail. In this paper, we sometimes directly refer to the interfaces (e.g., Dec) for short without the BFV prefix (e.g., BFV.Dec).

C FHE-SNARK and SNARK-FHE

First, we revisit the two paradigms for building Verifiable Computation (VCoed) using Fully Homomorphic Encryption (FHE):

- SNARK-FHE (SNARK outside FHE): Use FHE to evaluate the intended circuit and then employ a SNARK to prove that the FHE computation over the ciphertexts is performed honestly.
- FHE-SNARK (FHE outside SNARK): Use FHE to evaluate the intended circuit and generate a homomorphic proof that the plaintext computations within the FHE environment were performed honestly.

Difference in functionalities. A natural question is how these two approaches differ functionally. Our discussion on the relationship between VCoed and succinct two-party computation (2PC) clarifies this. SNARK-FHE can be viewed as a form of 2PC, making it immune to attacks we previously mentioned. The knowledge soundness of underlying SNARKs prevents input dependent attack, and the one-bit leakage is irrelevant in SNARK-FHE: the adversarial server always knows if a proof passes, simulating the client's reaction accurately except with negligible probability. This is because all proof generation inputs are known to it. Thus, SNARK-FHE exhibits no inherent one-bit leakage, unlike FHE-SNARK. In other words, FHE-SNARK realizes VCoed without strong verifier privacy, while SNARK-FHE realizes 2PC.

Difference in efficiency. With functional differences clarified, we now briefly compare their efficiencies. SNARK-FHE achieves a stronger primitive than FHE-SNARK but is consequently less efficient. Such efficiency degradation due to model differences (i.e., giving verify oracle or not) is relatively common, especially for FHE schemes (or FHE-based constructions). For example, for FHE schemes, to allow IND-CPA-D [49] security instead of IND-CPA security (i.e., the adversary is given oracle access to a decryption oracle that allows decryption of only honestly-generated ciphertexts, independent of the challenging ciphertext), recent works [13, 17] have shown that we need to perform regular bootstrapping procedures after a certain number of additions which can indeed cause orders of magnitude efficiency loss. Some other attacks need to be fixed by using differential privacy [50] or using much more conservative parameters [17], which also cause significant efficiency degradation.

Such situations greatly resemble our case: our strong verifier privacy indeed provides an additional verify oracle to the adversary, and this difference, as discussed above, is a core distinction between FHE-SNARK and SNARK-FHE. Thus, we believe this difference is one major reason that FHE-SNARK and SNARK-FHE may have a relatively significant gap in terms of efficiency.

Putting model aside, if we focus on the realization itself, the efficiency is fundamentally influenced by the alignment between algebraic structures and operations. SNARK-FHE faces a challenge in matching these components:

Algebraic Structures: The plaintext space in FHE can adopt algebraic structures such as finite fields or integer rings. Conversely, the ciphertext space typically utilizes power-of-2 cyclotomic rings, e.g., $\mathcal{R}_q$ with ring dimension 2048 and q being 32 or 64 bits (akin to those used in FHEW-like schemes) or larger ones with ring dimension 32768 and $q = 800$ (as seen in BFV-like schemes). In contrast, SNARK operations generally occur within a small sized prime field such as 128-bit. So that it poses challenges for SNARK-FHE, where the use of small prime fields to handle different ciphertext structures. However, the FHE plaintext space can directly accommodate the prime finite fields used in SNARK systems, enabling a more seamless and efficient alignment in FHE-SNARK.

Operations: The plaintext operations supported by FHE contain both logical and arithmetic operations. However, ciphertext manipulations are more complex, encompassing arithmetic operations alongside specialized procedures such as modulus switching (an error management technique that discards least significant bits to control noise growth), gadget decomposition (splitting ciphertexts into smaller components using radix decomposition), and RNS decomposition (residue number system-based ciphertext decomposition). On the other hand, the circuits to be proved by SNARK and its own operations are generally arithmetic. Therefore, SNARK-FHE has to pay extra efficiency cost to prove rounding, gadget decomposition, etc. on FHE ciphertexts with SNARK in the outer layer. In contract, FHE-SNARK leverages the native support of plaintext arithmetic operations within FHE to effectively cover the required SNARK computations.

D Advanced way to committing encrypted polynomials $\tilde{z}$

Note that in our construction of VCoed, a main question is that for $z := ((w^*, y), \mathbb{x})$, the prover only knows its input y in plaintext, therefore he can only homomorphically commit the multilinear extension polynomial $\tilde{z}$. Such a committing only provides an extractibility of z with secret key.

However, here we show that, by slightly modifying our polynomial commitment construction, we can achieve extractability without secret key almost for free. This is based on two key observations.

First, the prover directly encodes z as the evaluations of $\tilde{z}$ on the hypercube. Suppose that the extractor can obtain some evaluations on the hypercube in plaintext; then, it can extract the corresponding part of witness in plaintext.

Second, the Ligero/Brakedown polynomial commitment exhibits a form of separability. Recall that, in the commitment phase, the

prover rewrites the evaluations on hypercube as a matrix and encodes its rows. This means that the evaluations in different rows will not be mixed during encoding. If the plaintext portion consists of some rows of the matrix (the structure of R1CS allows us to adjust the order of the witness and therefore achieve this), then their encoding will also be in plaintext. Furthermore, during the evaluation phase, the prover discloses random columns of the encoded matrix, which also prevents the prover from using ciphertexts to replace plaintexts (if so, the resulting codeword will consist of a significant proportion of ciphertexts due to its error-correcting properties). We use the list-decoding algorithm of the underlying Reed-Solomon code and the straight-line extractor mentioned in Brakedown to achieve extraction by directly decoding the hashed codewords extracted via the random oracle.

Assume that $|z| = 2^n$ and $|w^*| = k_1 \cdot 2^{n-d}$, $|y| = k_2 \cdot 2^{n-d}$ and $|\mathbb{x}| = k_3 \cdot 2^{n-d}$. So that, when we write $[z]_{\mathsf{mix}} = ((\mathsf{Enc}(w^*), y), \mathsf{Enc}(\mathbb{x}))$ as a $2^d \times 2^{n-d}$ matrix M, each row of M consists either all plaintext or all ciphertexts. The formalized construction of our polynomial commitment is shown in Fig.8.

Above construction shares same multiplication-depth and similar compatibility with FHE packing technique.

Optimization. In the Spartan's PIOP construction, one needs to check that the statement encoded in the polynomial $\hat{z}$ is indeed x. We can accomplish this check using the testing phase of the polynomial commitment. Recall that our polynomial commitment first rewrites z as a matrix M, with x composing several rows of M. It then uses the hash of the encoding matrix $\hat{M}$ as the commitment. In the testing phase, the prover sends a random linear combination of rows of M and demonstrates that its encoding is close to the same linear combination of the encoding of the rows of $\hat{M}$. Since the verifier knows the statement x itself, it can compute the combination of statement rows itself, and check that its encoding is close to the combination of the corresponding rows encoding x, demonstrating that these encodings are indeed (close to) the codewords of x.

Theorem D.1. *If we replace the original polynomial commitment scheme in our SNARK for VCoed instances with the commitment scheme described above, then the resulting proof alone is sufficient to ensure the knowledge soundness of the constructed VCoed protocol.*

Firstly, above construction already satisfies the Completeness, Binding and Knowledge Soundness of polynomial commitment for encrypted polynomials.

Therefore, we only need to show how to achieve the extractibility of y without secret key.

We treat the hash function as a random oracle, enabling us to view the evaluation protocol as an IOP system. We first recall the existence of extractor with secret key:

As discussed in [38] and existing IOP-to-succinct-argument transformation of [5], given a prover P that convinces the argument-system verifier to accept $\tilde{z}(\alpha) = \beta$ with non-negligible probability, there is an efficient straight-line extractor capable of outputting IOP proof string π that "opens" the Merkle commitment sent by the argument system prover in the commitment phase.

Recall that the encoding matrix $\hat{M}$ is Merkle hashed in our construction. Therefore, π contains both the encoding matrix $\hat{M}$ and the randomness used for hashing. We use $\mathsf{Dec}(\hat{M})$ to represent

---

Polynomial commitment for encrypted $\tilde{z}$

Input for the committer: $[z]_{\mathsf{mix}} = ((\mathsf{Enc}(w^*), y), \mathsf{Enc}(\mathbb{x}))$ where $z = ((w^*, y), \mathbb{x})$;

Setup: Choose a random hash function h, and output $\mathsf{pp} = h$.

Commit: The committer proceeds as follows:

(1) Rewrite $[z]_{\mathsf{mix}}$, a vector with 2^n elements, as a $2^d \times 2^{n-d}$ matrix M.
(2) Encode each row of M using the Reed-Solomon code algorithm Encode and obtain the encoded matrix $\hat{M}$. Specifically, rows $\hat{u}_i$ composed of plaintexts are encoded via $\hat{u}_i \leftarrow \mathsf{Encode}(u_{i,1}, \cdots, u_{i,2^{n-d}})$; rows $\hat{u}_i$ composed of ciphertexts are encoded via $\hat{u}_i \leftarrow \mathsf{HE.Encode}(u_{i,1}, \cdots, u_{i,2^{n-d}})$.
(3) Use h to Merkle-hash the columns of $\hat{M}$. Output the hash root as the commitment *com* of $\tilde{z}$. All randomness (if exists) used during the commitment phase, along with $[z]_{\mathsf{mix}}$ and $\hat{M}$, constitutes the decommitment τ.

Decommit: Same as the origin construction.

Evaluation: The evaluation protocol consists of two phases, as follows:

Testing phase.

(1) The receiver chooses and sends a random vector $r \in \mathbb{F}^{2^d}$ to the committer.
(2) The committer computes and sends the linear combination of the row of M with scalars $\{r_i\}$, i.e., $\Sigma_{i \in [2^d]} r_i \cdot u_i$ (In this step, we always regard u_i in encrypted, in order words, if u_i is plaintexts, encrypt it first. And now $r_i \cdot u_i$ means scalar multiplication of ciphertexts.).
(3) The receiver decrypts the receiving vector to obtain u^* and computes its encoding $\hat{u}^* \in \mathbb{F}^N$. It choose a random subset I of $[2^N]$ with size Q and sends it to the committer.
(4) For each $j \in I$, the committer sends the j-th column of $\hat{M}$ to the receiver, along with the corresponding Merkle-hash proof.
(5) The receiver verifies that all the Merkle-hash proofs are valid and checks that a) for each column $\hat{v}_j$ of $\hat{M}$, the inner product $\langle r, \mathsf{Dec}(\hat{v}_j) \rangle$ equals the j-th element of $\hat{u}^*$, and b) for each $k \in S$, the k-th element in $\hat{v}_j$ is in plaintext.

Evaluation phase. Suppose the receiver wants to query $\tilde{z}(\alpha)$ with point $\alpha = (\alpha_0, \cdots, \alpha_{n-1}) \in \mathbb{F}^n$, the receiver proceeds as follows:

(1) The receiver computes $p_1 = ((1-\alpha_0, \alpha_0) \otimes (1-\alpha_1, \alpha_1) \otimes \cdots \otimes (1-\alpha_{d-1}, \alpha_{d-1})$ and $p_2 = ((1-\alpha_d, \alpha_d) \otimes \cdots \otimes (1-\alpha_{n-1}, \alpha_{n-1})$. The receiver runs a testing phase with the committer, except that r is set to p_1. If the testing phase fails, abort; otherwise, continue.
(2) Suppose u^* is the vector obtained in step 3 of above testing phase, the receiver computes $\tilde{z}(\alpha) = \langle p_2, u^* \rangle$.

Figure 8: Polynomial commitment for encrypted $\tilde{z}$

the matrix obtained by keeping the plaintext in $\hat{M}$ unchanged, and replacing the ciphertext with the plaintext obtained by decrypting it. If each row of $\mathrm{Dec}(\hat{M})$ is sufficiently close to a codeword, the extractor can decode them to obtain $[M]_p$, and subsequently recover $\tilde{z}$. We will demonstrate the following: 1) for each row of $\mathrm{Dec}(\hat{M})$, there exists a codeword such that the distance between them is less than $\gamma/3$ (in other words, the matrix $\hat{M}$, along with the encrypted polynomial $[\tilde{z}]_c$ and the randomness, constitutes a valid decommitment), 2) $[M]_p$ represents a polynomial $\tilde{z}$ that satisfies $\tilde{z}(\alpha) = \beta$, and 3) for a suitable linear code with relative distance γ, an efficient decoder algorithm exists.

For any vector u, we denote by $u[j]$ the j-th elements of u. Using Lemma.4.2, we have that:

Lemma D.2. *If the prover passes all the checks in the texting phase with probability at least $\gamma/|\mathbb{F}| + (1 - \frac{\gamma}{3\cdot N})^Q$, then there exists a sequence of m codewords $c_0, \cdots, c_{m-1}$ in L such that the size of $\Delta := \{j \in [n] | \exists i \in [m]\ s.t. c_i[j] \neq \hat{u}_i[j]\}$ is less than $\gamma/3$, where $\hat{u}_i$ is the i-th decrypted row of extracted encoding matrix $\mathrm{Dec}(\hat{M})$.*

Proof. Assume that there exists a row $\hat{u}'$ of the extracted encoding matrix $\mathrm{Dec}(\hat{M})$ such that $d(c', \hat{u}') \geq \gamma/3$ where c' is its closest codeword in L. Then at least one of the following two cases happens:

- **Case I.** Given randomness $\{r_i\}_{i\in[2^d]}$ chosen by the receiver, $d(\Sigma_{i\in 2^d} r_i \cdot \hat{u}_i, L) < \gamma/3$.
- **Case II.**Given randomness $\{r_i\}_{i\in[2^d]}$ chosen by the receiver, $d(\Sigma_{i\in 2^d} r_i \cdot \hat{u}_i, L) \geq \gamma/3$.

Setting e to be the maximal integer less than $\gamma/3$, then from the assumption we have that the size of $\Delta := \{j \in [n] | \exists i \in [m]$ s.t. $u_i[j] \neq \hat{u}_i[j]\}$ is lager than e. From Lemma.4.2, the probability that Case I happens is no more than $\gamma/|\mathbb{F}|$.

In Case II, for a random $j \in [N]$, the probability that the j-th position of $\Sigma_{i\in 2^d} r_i \cdot \hat{u}_i$ is consistent with $\mathrm{Encode}(u^*)$ is no more than $(1 - \frac{\gamma}{3N})$. Since the receiver choose the subset I with size Q, the probability that the prove passes the testing phase is less than $(1 - \frac{\gamma}{3\cdot N})^Q$ in this case.

Therefore, the total probability that the prover passes the testing phase is less than $\gamma/|\mathbb{F}| + (1 - \frac{\gamma}{3\cdot N})^Q$, arriving at a contradiction. □

Now, suppose there exists such a sequence of m codewords $c_0, \cdots, c_{m-1}$ in L described in above lemma, and $\{u_i\}$ are their decodings. In the evaluation phase, if the prover can convince the verifier with at least $(1 - \frac{2\gamma}{3\cdot N})^Q$, then the extracted polynomial $\tilde{z}$ satisfies $\tilde{z}(\alpha) = \beta$. Otherwise, suppose that $\tilde{z}(\alpha) = \beta$, it must be that $\Sigma_i q_1[i] \cdot u_i \neq u^*$, where q_1 is the vector sent by the receiver, and u^* is the vector obtained by the receiver in step(2). From the properties of linear code and the fact that $d(\Sigma_i q_1[i] \cdot \hat{u}_i, \Sigma_i q_1[i] c_i) < \gamma/3$, we have that $d(\Sigma_i q_1[i] \cdot \hat{u}_i, \mathrm{Encode}(u^*)) \geq 2\gamma/3$. Therefore, the probability that the committer passes the evaluation phase is less than $(1 - \frac{2\gamma}{3\cdot N})^Q$, arriving at a contradiction.

Finally, we need to ensure that, given a vector $\hat{u}$ satisfying $d(\hat{u}, L) < \gamma/3$, there exists an efficient decoder that decodes it and obtains u with the closest encoding to $\hat{u}$. According to the well-known list decoding algorithm put forward by Guruswami and Sudan[39], for a Reed-Solomon code $L[N, k, \gamma]$, there exists an efficient algorithm that can list all messages m s.t. $d(\mathrm{Encode}(m), \hat{u}) \leq e$ as long as $e \leq N - \sqrt{kN}$. In other words, to achieve our decoding, we only require that $\frac{N-k+1}{3} \leq N - \sqrt{kN}$. It means $N \geq k$, which is already satisfied by Reed-Solomon code.

In summary, there exists an efficient extractor that, giving the secret key of encryption, can extract the committed polynomial f. Specifically, this is achieved by extracting $\hat{M}$ from the RO oracle, decrypting it to obtain $\mathrm{Dec}(\hat{M})$, and then decoding it to obtain $\mathrm{Dec}(M)$, the plaintext representation of $\tilde{z}$. The knowledge soundness error is $\gamma/|\mathbb{F}| + (1 - \frac{\gamma}{3N})^Q + (1 - \frac{2\gamma}{3N})^Q$.

The extractability without the secret key is proved as follows: The construction of this extractor without secret key is essentially the same as the above extractor. The only difference is that this extractor does not have the secret key and therefore cannot decrypt $\hat{M}$. Fortunately, in our construction, rows $\hat{u}_i$ of $\hat{M}$, where $\mathrm{bin}_d(i) \in S$, should be composed of solely plaintexts. Therefore, the extractor without secret key can still decode them and obtain the evaluations of f on the set S_p.

Although an adversary might insert ciphertexts into $\{\hat{u}_i\}_{\mathrm{bin}_d(i)\in S}$ to break the decoding, the receiver checks in step 5 of the testing phase that the corresponding elements of $\{\hat{u}_i\}_{\mathrm{bin}_d(i)\in S}$ are indeed plaintexts. Therefore, we can modify above analysis by replacing any ciphertexts in $\{\hat{u}_i\}_{\mathrm{bin}_d(i)\in S}$ with a special plaintext symbols $\perp$, and the analysis remains valid. Consequently, the extractor can decode the modified $\hat{u}_i$ (which are now all in plaintexts) to obtain the evaluations of f on the set S_p without the secret key. In other words, it means that there are enough valid plaintexts in $\{\hat{u}_i\}_{\mathrm{bin}_d(i)\in S}$ for decoding.

Homomorphic encrypted version of Spartan's PIOP

Index: $\mathbb{i} = (\mathbb{F}, A, B, C, m, n)$,
Statement: $\mathsf{Enc}_{\mathsf{pk}}(x)$. Witness: $\mathsf{Enc}_{\mathsf{pk}}(w)$.

$\mathcal{I}(1^\lambda, \mathbb{i})$: Run the $\mathcal{I}$ algorithm of Spartan to encode A, B, C as polynomials $\tilde{A}, \tilde{B}, \tilde{C}$ and output these polynomial oracles. (We need to use the same techniques in Spartan or Brakedown to handle these three sparse polynomials. Due to that all A, B, C are in plaintext, we skip it here.)

$\langle P, V \rangle$:

(1) Let $z = (w, x)$. The prover sends the encrypted polynomial oracle of $\tilde{z}$ to the verifier.
(2) Denote by
$$F(x) = \Sigma_{y\in\{0,1\}^n}\tilde{A}(x,y)\cdot\tilde{z}(y) \times \Sigma_{y\in\{0,1\}^n}\tilde{B}(x,y)\cdot\tilde{z}(y) - \Sigma_{y\in\{0,1\}^n}\tilde{C}(x,y)\cdot\tilde{z}(y),$$
the prover homomorphically computes the encryptions of Az, Bz, Cz to obtain the encrypted representation of polynomials $\Sigma_{y\in\{0,1\}^n}\tilde{A}(x,y)\cdot\tilde{z}(y)$ and $\Sigma_{y\in\{0,1\}^n}\tilde{B}(x,y)\cdot\tilde{z}(y), \Sigma_{y\in\{0,1\}^n}\tilde{C}(x,y)\cdot\tilde{z}(y)$. We denote by these three polynomials as $\tilde{z}_A, \tilde{z}_B, \tilde{z}_C$. Now we have that $F(x) = \tilde{z}_A(x) \times \tilde{z}_B(x) - \tilde{z}_C(x)$. The verifier chooses randomness τ and sends it to the prover.
(3) The prover and verifier runs the sumcheck protocol on encrypted polynomials constructed before for the following identity:
$$0 = \Sigma_{x\in\{0,1\}^n} eq(x,\tau)F(x)$$
(4) Above sumcheck will reduce the check to the following three equalities:
$$\Sigma_{y\in\{0,1\}^n}\tilde{A}(r_x,y)\cdot\tilde{z}(y) = z_1,$$
$$\Sigma_{y\in\{0,1\}^n}\tilde{B}(r_x,y)\cdot\tilde{z}(y) = z_2,$$
$$\Sigma_{y\in\{0,1\}^n}\tilde{C}(r_x,y)\cdot\tilde{z}(y) = z_3$$
The verifier chooses and sends three randomness r_1, r_2, r_3 and both prover and verifier runs the following sumcheck on encrypted polynomials
$$\Sigma_{y\in\{0,1\}^n}(r_1\tilde{A}(r_x,y) + r_2\tilde{B}(r_x,y) + r_3\tilde{C}(r_x,y))\cdot\tilde{z}(y) = r_1z_1 + r_2z_2 + r_3z_3$$
(5) Above sumcheck will reduce the checking to the following queries:
$$\tilde{A}(r_x,r_y) = z_A, \tilde{B}(r_x,r_y) = z_B, \tilde{C}(r_x,r_y) = z_C, \tilde{z}(r_y) = z'$$
all of them can be achieved by verifier through the (encrypted) polynomial oracle.
(6) The verifier checks that the public input is consistent with the witness, that is, $\tilde{z}(0^{n-\log|io|}, r^*) = \tilde{io}(r^*)$ for a random $r^* \in \mathbb{F}^{\log|x|}$.

Figure 10: Homomorphic encrypted version of Spartan's PIOP

Argument of knowledge for encrypted relations

Index: $\mathbb{i} = (\mathbb{F}, A, B, C, M, N)$,
Statement: $\mathsf{Enc}_{\mathsf{pk}}(x)$. Witness: $\mathsf{Enc}_{\mathsf{pk}}(w)$. FHE key pairs: $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Setup}(1^\lambda)$: Run the setup algorithm of the polynomial commitment and output crs.

$\mathcal{I}(crs, \mathbb{i})$: Run the $\mathcal{I}(crs, \mathbb{i})$ algorithm of homomorphic encrypted version of Spartan's PIOP constructed before, except it sends the polynomial commitment of $\tilde{A}, \tilde{B}, \tilde{C}$ instead of provides the oracles. (We need to use the techniques provided in Brakedown[38], or Spartan[60], to issue these sparse polynomials for efficiency. Since these polynomials are all in plaintext we skip details here).

$\langle P, V \rangle$: The prover sends the commitment of encrypted polynomial $\tilde{z}$, where $z = (w, z)$. Both parties run the homomorphic encrypted version of Spartan's PIOP for the encrypted R1CS instance $(\mathbb{i}, \mathsf{Enc}_{\mathsf{pk}}(x), \mathsf{Enc}_{\mathsf{pk}}(w))$, and when the verifier want to query $\tilde{z}$ and $\tilde{A}$,$\tilde{B}$,$\tilde{C}$, it runs the evaluation protocols of underlying polynomial commitment respectively.

Figure 11: Argument of knowledge for hybrid relation

Sumcheck on encrypted polynomials.

Suppose $\mathsf{ct}_y = \mathsf{Enc}_{\mathsf{pk}}(y)$ and $f(x) := h(g_0(x), \cdots, g_{m-1}(x))$ where each g_i are multilinear polynomial over $x \in \mathbb{F}^n$ of which encrypted representation is known by the prover.
The prover wants to prove that $y := \Sigma_{x\in\{0,1\}^n} f(x)$. Both party execute as follows:

(1) Let $f_1(x) := f(x)$, $y_1 = y$, r be an empty vector, for i from 1 to $n-d$, run as follows:
   (a) Let $f_i^*(X) := \Sigma_{x'\in\{0,1\}^{n-i}} f(x', X, r)$ be a t-degree univariate polynomial. For each $k \in [t+1]$ and $x' \in \{0,1\}^{n-i}$, the prover computes $\mathsf{Enc}_{\mathsf{pk}}(g_j(x', k, r)) = \Sigma_{x_d,\cdots,x_{n-1}\in\{0,1\}}(\mathsf{Enc}_{\mathsf{pk}}(g_j(x'_1, \cdots, x'_{d-1}, x_d, \cdots, x_{n-1})) \cdot \Pi_{t\in[d,n-1]}(x_t r'_t + (1-x_t)(1-r'_t)))$ where r'_t is the $(t-d+1)$-th elements of vector $(x'_d, \cdots, x'_{n-i-1}, k, r)$.
   (b) The prover computes $\mathsf{Enc}_{\mathsf{pk}}(f(x', k, r)) = h(g_0(x', k, r), \cdots, g_{m-1}(x', k, r))$ from $\mathsf{Enc}_{\mathsf{pk}}(g_j(x', k, r))$ homomorphically.
   (c) The prover computes and sends $\mathsf{ct}_k^* = \mathsf{Enc}_{\mathsf{pk}}(f_i^*(k)) = \Sigma_{x'\in\{0,1\}^{n-i}}\mathsf{Enc}_{\mathsf{pk}}(f(x', k, r))$ to the verifier.
   (d) The verifier decrypts ct_k^* and checks whether $f_i^*(0) + f_i^*(1) = y_i$. If so, reconstruct $f_i^*(X)$ using $f_i^*(k)$. Sample and send $r' \leftarrow \mathbb{F}$ to the prover. Set $y_{i+1} := f_i^*(r')$ and $r := (r', r)$.
   (e) Suppose that the encryptions of $\{g_j(x', x'')\}_{x'\in\{0,1\}^d}$ are packed into one ciphertexts, above three steps can be easily executed over packed ciphertexts and Step (a) consists only scalar multiplications and Step (b) consists both scalar multiplications and ciphertext/plaintext-ciphertext vector multiplications and Step (c) consists only additions.
(2) Now, the prover only needs to prove $y_{n-d+1} = \Sigma_{x'\in\{0,1\}^d} f(x', r)$ for encrypted y_{n-d+1}. For all $x' \in \{0,1\}^d$, the prover computes $\mathsf{Enc}_{\mathsf{pk}}(g_j(x', r))$ same as above and sends all these ciphertexts to the verifier.
(3) The verifier decrypts and retrieves $f(x', r)$. The verifier checks whether $y_{n-d+1} = \Sigma_{x'\in\{0,1\}^d} f(x', r)$, if so, chooses a random $r' \leftarrow \mathbb{F}^d$, queries the oracle of f and checks whether $O(f)(r', r) = f(r', r)$.

Figure 9: Sumcheck protocol for encrypted polynomials

E Some constructions for SNARKs

Construction of the sumcheck protocol on encrypted polynomials, homomorphic encrypted version of Spartan's PIOP and Argument of knowledge for encrypted relations is shown in Figure 9, 10 and 11 resp.