FHE-SNARK vs. SNARK-FHE: From Analysis to Practical Verifiable Computation

Xinxuan Zhang[1,2], Ruida Wang[1,2], Zeyu Liu[3], Binwu Xiang[1,2], Yi Deng[1,2] and Xianhui Lu[1,2]

[1] State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS
[2] School of Cyber Security, University of Chinese Academy of Sciences
[3] Yale University
{zhangxinxuan, wangruida, xiangbinwu, deng, luxianhui}@iie.ac.cn
zeyu.liu@yale.edu

Abstract. Verifiable Computation over encrypted data (VC) faces a critical dilemma between two competing paradigms: *SNARK-FHE* (applying SNARKs to prove FHE operations) and *FHE-SNARK* (homomorphically evaluating SNARK proofs). There are two interesting questions remain open to solving such a dilemma: 1) Are they identical in terms of security? 2) How practically efficient can we get? This work answers these questions through the following results:
1) We establish a formal security analysis between VC and secure two-party computation (2PC). We investigate VC with server inputs and show the following: a) VC with server input has an exact 1-bit security loss compared to 2PC; b) SNARK-FHE aligns with 2PC while FHE-SNARK naturally falls in the VC category; c) Existing FHE-SNARK works is vulnerable in the VC with server input setting, for which we formalize a *input-dependent attack*.
2) We design an FHE-friendly SNARK that is: a) 3× lower multiplicative depth than FRI-based SNARKs; b) Compatible with FHE SIMD operations. Based on this novel SNARK, we construct an FHE-SNARK scheme that has: a) *Stronger security*: resistant against input-dependent attack; b) *8× speedup*: 3.6-hour proof generation for 2^{20}-gate circuits on a single core CPU (vs. 29 hours in the state-of-the-art); c) *Practical verification*: 65.3 MB proofs with 2.7 seconds verification (single core).

1 Introduction

Fully homomorphic encryption (FHE), first realized by Gentry's breakthrough work in 2009 [31], enables arbitrary computations on encrypted data without decryption. After a decade of advancements [10,9,7,22,8,32,21,18,16], FHE demonstrates practical efficiency for diverse applications including secure machine learning [40,48,41,38,20], private information retrieval [2,50,26,45], private set intersection [14,13,19], and oblivious message retrieval [46,47,42].

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 [30] introduced the concept of Verifiable Computation over encrypted data (which we denote as VC for short).[4] Essentially, VC 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 VC in the next decade has limited capability and efficiency [30,24], 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 VC specifically for FHE [25,6,4,52,28]. 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 *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 [3,27,29], introducing an alternative approach to building VC 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 29 hours in [27] (0.1 second per gate compared to 20 minutes in the previous method).

While these advances are fascinating, we believe that several aspects remain unexplained. First, it remains unclear why these two seemingly similar approaches—one employing SNARKs outside FHE and the other leveraging FHE outside SNARKs—result in such a significant performance gap. An intuition is that the more performant one is weaker than the less efficient one. Thus, this raises our first question:

(1) Do FHE-SNARK and SNARK-FHE offer equivalent security guarantees?

Furthermore, the prior works in the second line (FHE-SNARK) seems to provide a stepping stone towards practical VC, but there remains substantial room for further enhancement. For example, previous works [3,27] 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 us to our second question:

[4] Note that while the concept of verifiable computation has roots in older definitions [53], which focuses on integrity without privacy, thus does not align our objectives.

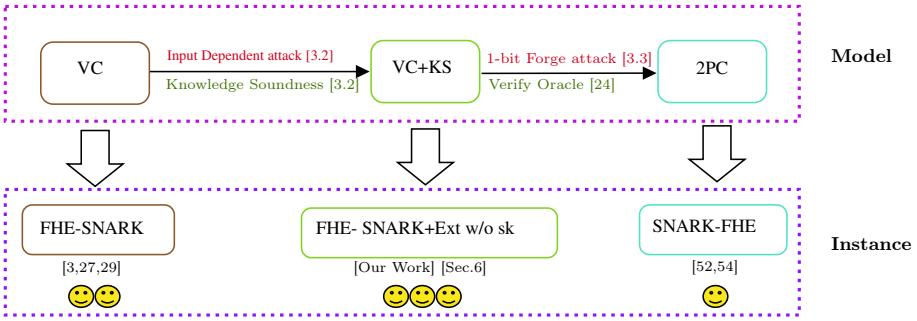

Fig. 1. Our result, where "KS" denotes knowledge soundness and "Ext w/o sk" stands for extractability without secret key.

(2) 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

This paper addresses aforementioned questions as described in Figure.1.

More comprehensive model for verifiable computation. First, we demonstrate that in VC 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 VC 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 VC and secure two-party computation (2PC). Our analysis reveals that VC with knowledge soundness still has a 1-bit security loss compared to 2PC. We clarify it by formalizing a 1-bit forge attack and propose Theorem.1 to establish that VC with strong client privacy (where the server can query the "verify" oracle once) is *equivalent* to 2PC.

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

FHE-friendly SNARK. We first design a novel SNARK variant where the statement and a portion of the witness are encrypted. Our construction achieves three key improvements:

- Extractablity without secret key: It allows for the extraction of the plaintext portion of witness without requiring the secret key, thereby preventing input-dependent attacks in the resulting VC constructions.
- 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.

During the construction of our SNARK, we develop a novel commitment scheme for polynomials whose point-value representations may contain both ciphertexts and plaintexts. This commitment scheme has the potential for independent applications in other areas, such as blind SNARKs [27].

Practical FHE-SNARK for verifiable computation. Based on the aforementioned results, we construct a practical knowledge soundness FHE-SNARK scheme with explicit resistance against input-dependent attack. Our FHE-SNARK demonstrate a $\mathbf{8}\times$ acceleration in proof generation time compared to [27]. Specifically, for arithmetic circuits with 2^{20} gates, our implementation costs **3.6** hours (**12.3** ms/gate) to generate a computation proof on a single-core CPU. Moreover, the scheme is well-suited for multi-core processors since all operations are parallelizable.

1.2 Technique Overview

Input dependent attack and knowledge soundness in VC. Traditional VC frameworks exhibit critical security gaps when extended to server-assisted dual-input scenarios. While computational soundness guarantees result correctness under the existence of valid witness w, it fails to enforce the server's explicit knowledge of w. This fundamental oversight enables *input-dependent attack* - a class of vulnerabilities where adversaries strategically adapt inputs based on inferred client information, violating the input independence principle inherent in secure 2PC.

The security inadequacy becomes evident when analyzing existing approaches for handling server inputs. In scenarios where both parties provide inputs, two mainstream solutions prove unsatisfactory:

- **Embedded Input Setup**: Embedding server's input w into the setup algorithm destroys the one-time initialization and and creates undesirable server inputs dependency.
- **Verification-Time Input Disclosure**: Treating w as verification algorithm input violates outsourceability ($o(|\mathsf{C}|)$ complexity requirement) by forcing linear-time verification.

Treating server inputs as private to client avoids these drawbacks but introduces new client privacy challenges, necessitating stronger security guarantees. To concretize the threat, consider a bilateral auction scenario where the client's bid m remains encoded via VC scheme. A malicious server can compute $\mathsf{Enc}(w') = \mathsf{Enc}(m+1)$ as its bid input, ensuring victory at a minimum cost while generating valid proofs.

This attack generalizes to any function computable through $\mathsf{VC.compute}$, enabling arbitrary input manipulation that preserves proof validity despite the server lacking knowledge of the actual w' value. The core vulnerability stems from the server's ability to programmatically derive its input from the client's encrypted data without semantic understanding. In other words, if the server has to know its own input, it will cannot set up such input dependent attacks since the client's input keep hidden from the server. Therefore we put forward a new security requirement called knowledge soundness, which requires the existence

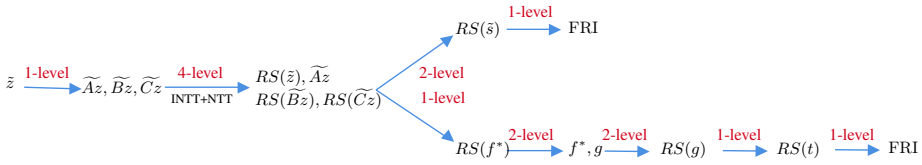

Fig. 2. Multiplication-depth of prior constructions[3,27]

of extractor such that, *without secret key* as input, it can extract the input of server when the server generates a valid proof. In fact, a VC with verifier-privacy and knowledge soundness is very close to 2PC. The only gap between them is the one bit from whether the proof is valid, when the proof is designed verifier.

SNARKs for VC 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 about these polynomials and then uses concrete cryptographic primitives such as hash functions or polynomial commitments to instantiate these oracles. Since in VC 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 (since w contains some intermediate values that arise during the computation). As a result, the prover (acted by the server) can only obtain the encoded polynomials in encrypted.

Prior approaches. Prior constructions[3,27] build upon the Fractal scheme and replace some oracles/messages sent by the prover to the corresponding encrypted ones. Specifically, they encoding z as a univariate encrypted 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^* is a specially designed polynomial constructed from $\widetilde{Az}, \widetilde{Bz}, \widetilde{Cz}$, and g is a polynomial computable from f^*'s coefficients. The introduction of Reed-Solomon code and polynomial multiplications brings extensive use of NTT. Even equiped with 2-layer NTT, they still exhibit a substantial multiplication depth (12 depths, as illustrated in Fig.2), significantly increaseing the cost of single homomorphic evaluations in the underlying FHE scheme.

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 z as an encrypted multilinear polynomial $\tilde{z}$ on $\{0,1\}^{\log |z|}$ (instead of a univariate polynomial). Az, Bz, and Cz are encoded similarly as $\widetilde{Az}, \widetilde{Bz}, \widetilde{Cz}$. Follows Spartan's PIOP [51], 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 $M = A, B, C$. The former one, multivariate $F(x)$'s zero-check, reduces to a sumcheck $\Sigma_{x \in \{0,1\}^*} eq(x, \tau) F(x)$, while the latter one, the encoding check, reduces to sumchecks $\Sigma_y \widetilde{M}(\tau', y)\tilde{z}(y) = \widetilde{Mz}(\tau')$. Through

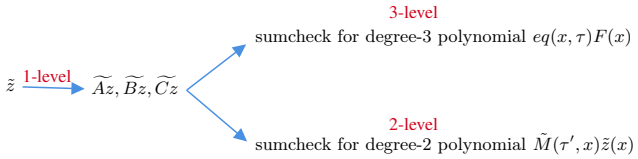

Fig. 3. Multiplication-depth of our construction

running the well-known sumcheck[49] homomorphically, these checks reduces to random queries to the encrypted polynomial z, avoiding NTT unlike univariate FRI-based approaches.

However, running sumchecks homomorphically introduces significant multiplicative depth. 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 $f(x_0, \cdots, x_{n-i}, r_{n-i+1}, \cdots, r_{n-1})$ for all $x_0, \cdots, x_{n-i+1} \in \{0,1\}$ 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 $O(n2^n)$ 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.

With the modified sumcheck protocol, the proof system achieves a low multiplication depth (only 4), as described in Figure.3.

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 (it requires one matrix-vector multiplication for each packed ciphertext), 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 a 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, committing algorithm involves only scalar multiplications, while the evaluation contains only inner-products, on packed ciphertext. And the way to pack here is identical to the one in sumcheck; therefore, we can further 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 while maintaining a small depth. Since in the constructions, the prover only commits to the encrypted polynomial $\tilde{z}$, thereby the levels of polynomial commitment are independent from the levels of PIOP. We choose a 3-layer NTT so that the resulting SNARK still has a multiplication-depth of 4.

Prevent input dependent attack. To prevent input-dependent attacks, we need to ensure that the prover indeed knows its input to the computed circuit. However, since SNARKs are run on the (partially) encrypted witness, the traditional knowledge soundness of SNARKs does not work here because the extractor requires secret key to decrypt the prover's messages and extract the witness. In fact, once plaintexts and ciphertexts are mixed through homomorphic evaluations, it is difficult to retrieve the plaintexts again without the secret key.

A straightforward approach is to prove knowledge of the server's input used in the proof as discussed in [3]. However, this is computationally expensive as it means to prove the computation of SNARK proof. Another simple solution is to require the server to public a commitment of its input and view it as parts of the statement. This can be seen as a commit-and-prove paradigm. To make this solution practical, one has to use a commitment scheme that fitting the structure of FHE. An optional commitment scheme is hash functions. In this case, the prover needs to add the hash circuit into the SNARK circuit. Even when using a SNARK-friendly hash like Poseidon, it increases by about tens of constraints per hashed elements. It is a significant cost when the number of the plaintext elements of the witness is relatively large compared to the circuit size. Another option is to use a separate polynomial commitment scheme specifically for the plaintext portion of the witness. Besides the additional cost coming from the additional commitment, since we use a multilinear polynomial commitment, we need to pad enough "0"s to the plaintext and ciphertext parts of the witness to ensure they satisfy specific structures (e.g., that their lengths are the same) for verification in most cases. This will increase the size of instance and bring noticeable cost especially for the verification. However, we demonstrate that, by slightly modifying our construction, we can achieve the extractability without the secret key at virtually no additional cost. 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 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 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.

The combination of PIOP and the modified polynomial commitment results in a SNARK for VC instances that satisfies a strong knowledge soundness, ad-

ditionally allowing for the extraction of the plaintext portion of the witness. To complete the security proof of the resulting SNARK, we introduce several new definitions, such as polynomials for cryptomix polynomials (where a cryptomix polynomial is defined as one whose point-value representation consists of both ciphertexts and plaintexts) and a new variant of knowledge soundness allowing extraction without a secret key. These definitions not only help us to conclude the security proof follows the "polynomial commitment + PIOP" paradigm, but also have the potential for independent applications in other areas.

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. We will use the terms (non-uniform) PPT algorithm and polynomial-size circuits interchangeably. 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 $v[i] := v_i$.

In this paper, we will frequently refer to the plaintext and ciphertext representations of elements. To avoid ambiguity, we use $[\cdot]_c$ to represent ciphertext and $[\cdot]_p$ to represent plaintext in the following way:

$$[\alpha]_c := \begin{cases} \alpha, & \text{if } \alpha \text{ is a ciphertext} \\ \mathsf{Enc}_{\mathsf{pk}}(\alpha), & \text{if } \alpha \text{ is a plaintext} \end{cases} \qquad [\alpha]_p := \begin{cases} \mathsf{Dec}_{\mathsf{sk}}(\alpha), & \text{if } \alpha \text{ is a ciphertext} \\ \alpha, & \text{if } \alpha \text{ is a plaintext} \end{cases}$$

For anyone owning α and the public key pk (resp. the secret key sk), it can always generate $[\alpha]_c$ (resp. $[\alpha]_p$) efficiently. For any matrix $M := (u_{i,j})$, we denote by $[M]_c$ the matrix $([u_{i,j}]_c)$, and by $[M]_p$ the matrix $([u_{i,j}]_p)$.

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 [12].

Definition 1 (R1CS Indexed Relation). *The indexed relation $\mathcal{R}_{R1CS}$ is the set of all triples*

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

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 (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.

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

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

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

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

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

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

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

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 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 holds:*

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

Definition 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. We now introduce two specific types of information-theoretic proof systems: interactive oracle proofs (IOPs) and polynomial interactive oracle proofs (PIOPs).

Interactive Oracle Proofs (IOPs). An (holographic) IOP 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:

- $\mathcal{I}(\mathtt{i})$: On input the index $\mathtt{i}$, the **deterministic** algorithm $\mathcal{I}$ outputs an encoding, $\mathsf{Ind}(\mathtt{i})$.
- $\langle \mathcal{P}(\mathsf{Ind}(\mathtt{i}), \mathtt{x}, \mathtt{w}), \mathcal{V}^{\mathsf{Ind}(\mathtt{i})}(\mathtt{x}) \rangle$: $\langle \mathcal{P}, \mathcal{V} \rangle$ is a public-coin protocol between the prover and verifier, where the prover sends the first and last message, and the verifier has only oracle access to the prover's messages.

Furthermore, an (holographic) IOP has to satisfy the corresponding **Correctness** and **Soundness** properties, both of which are defined similarly to those of standard proof systems, except that the verifier is given only oracle access to the prover's messages and $\mathsf{Ind}(\mathtt{i})$.

A holographic IOP can be compiled into a SNARK in the random oracle model using BCS compilation [5]. Furthermore, if the holographic IOP satisfies a special variant of soundness called round-by-round soundness (or round-by-round knowledge soundness, respectively), then the resulting scheme satisfies standard soundness (or knowledge soundness, respectively) in the random oracle model. Additionally, if the holographic IOP satisfies HVZK, then the resulting SNARK satisfies standard zero-knowledge in the random oracle model.

Polynomial Interactive Oracle Proofs (PIOPs). PIOPs is a variant of IOPs where the oracles sent by the prover to the verifier are essentially polynomials. 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 $\mathcal{I}(\mathtt{i})$ encodes the index $\mathtt{i}$ as several polynomial functions $\mathsf{Ind}(\mathtt{i})$ and $\langle \mathcal{P}(\mathsf{Ind}(\mathtt{i}), \mathtt{x}, \mathtt{w}), \mathcal{V}^{\mathsf{Ind}(\mathtt{i})}(\mathtt{x}) \rangle$ 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 $\mathsf{Ind}(\mathtt{i})$, 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.

2.2 Polynomial commitment

Polynomial commitments enable a committer to commit a polynomial and then open the value of the polynomial at arbitrary points. In this paper, we only consider polynomial commitments for multilinear polynomial. The definition is shown as follows:

Definition 5 (Polynomial Commitment Scheme). *A multilinear polynomial commitment over a field $\mathbb{F}$ consists of four protocols* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{deCom}, \mathsf{Eval})$*:*

- $\mathsf{Setup}(1^\lambda)$*: On input the security parameter* 1^λ*, the* Setup *algorithm outputs a public parameter* pp*.*
- $\mathsf{Com}(\mathsf{pp}, f)$*: On input a multilinear polynomial* $f \in \mathbb{F}[X_1, \cdots, X_d]$ *and the public parameter* pp*, the* Com *algorithm outputs a commitment* C *and a decommitment* τ*.*
- $\mathsf{deCom}(\mathsf{pp}, C, \tau)$*: On input a public parameter* pp*, a commitment* C *and a decommitment* τ*, the* deCom *algorithm outputs the committed multilinear polynomial* $f \in \mathbb{F}[X_1, \cdots, X_d]$ *or* $\perp$ *if it believes that the decommitment is invalid.*
- $\mathsf{Eval}(\mathsf{pp}, C, z, y; f, \tau)$*:* Eval *is an argument system* $\langle P, V \rangle$ *with the setup algorithm* Setup *for the relation:*

$$\mathcal{R}_{eval} = \{(\mathsf{pp}, C, z, y; f, \tau) : \mathsf{deCom}(\mathsf{pp}, C, \tau) = f \wedge f(z) = y\}$$

A polynomial commitment scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{deCom}, \mathsf{Eval})$ should satisfy the following properties:

– **Completeness**: For all $\lambda \in \mathbb{N}$, multilinear polynomial $f \in \mathbb{F}[X_1, \cdots, X_d]$ and any z,

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

– **Computational Binding**: For all $\lambda \in \mathbb{N}$ and any PPT adversary $\mathcal{A}$,

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

– **Knowledge Soundness**: Eval is an argument of knowledge (satisfying knowledge soundness) for the relation $\mathcal{R}_{eval}$.

3 Model: From Verifiable Computation to 2PC

In this section, we delve into the model formalization specifically focusing on two-party (client-server) protocols. We begin by adapting the standard verifiable computation (VC) model from [30], and explore the VC model in various scenarios, differing by whether the server possesses its own inputs. Next, we introduce a type of attack, termed an input-dependent attack, to highlight the vulnerabilities in existing security definitions. To defend against such attacks, we propose an additional security property. Finally, we establish both the gaps and equivalences between VC under our new security definition and 2PC.

3.1 Verifiable Computation

We adapt the definition from [30] and describe the difference and new properties we introduce below.

Definition 6. ***(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* $(\mathsf{sk}, \mathsf{pk})$*.*
- $(\mathit{Enc}(m), \mathit{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, m)$*: an encoding algorithm takes an input data* m *and public key* pk*, and outputs an encoded* $\mathit{Enc}(m)$ *and a statement* st*.*
- $(\mathit{Enc}(y), \pi) \leftarrow \mathsf{VC.Compute}(\mathsf{pk}, \mathit{Enc}(m), \mathsf{C}, w)$*: the computation algorithm takes the public key* pk*, the encoded* $\mathit{Enc}(m)$*, circuit* C *and server's input* w *(if have), it outputs an encoded computing result* $\mathit{Enc}(y)$ *and a proof* π*.*
- $y/\perp \leftarrow \mathsf{VC.Verify}(\mathsf{sk}, \mathit{Enc}(y), \mathit{st}, \pi)$*: given the secret key* sk*, the encoded result* $\mathit{Enc}(y)$*, statement* st*, and proof* π*, the verification algorithm returns* y *behind its encoding* $\mathit{Enc}(y)$ *if* $y = \mathsf{C}(m, w)$ *or outputs* $\perp$ *otherwise.*

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

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

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

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

Security. In this paper, we categorize the security model of verifiable computation into several tiers, progressing from the simplest to the most complex scenarios. Such stratification promotes a more comprehensive understanding and robust implementation of the concept. The tiers are as follows:

I. Client with private input, server with **no** input: In this scenario, all data accessible to the server is integrated into the public circuit to be computed.
II. Client with private input, server with **public** input: In this scenario, the server possesses some public data that is independent of the setup. This allows a single setup (independent of the server's data) to serve different data computations and verifications, enhancing flexibility and practicality.
III. Client and server both have **private** input: In the first two scenarios, we only need to consider the server as the adversary. However, in the third scenario, it is necessary to consider the client as the adversary as well, as briefly mentioned in HELIOPOLIS [3] regarding prover privacy.

Client security: Previous work [29] 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 VC 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 $\mathsf{Adv}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda) = \Pr[\mathsf{G}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda) \to 1] - 1/2 = \mathsf{negl}(\lambda)$.

$\underline{\mathsf{G}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda)}$:
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** $b \stackrel{?}{=} \hat{b}$

– Soundness: meaning that a malicious server cannot produce an invalid computational result that passes verification. More formally, a VC 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 $\mathsf{Adv}^{\mathsf{Soundness}}_{\mathsf{VC},\mathcal{A}}(\lambda) := \Pr[\mathsf{G}^{\mathsf{Soundness}}_{\mathsf{VC},\mathcal{A}}(\lambda) \rightarrow 1] = \mathsf{negl}(\lambda)$. Without loss of generality, we add the server's input (w) to the game $\mathsf{G}^{\mathsf{Soundness}}_{\mathsf{VC},\mathcal{A}}(\lambda)$. In case I, w is empty.

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

Server security: In the three scenarios above, only the third case requires server security: as in this case, the server has its own private input. Specifically, in scenario III, a VC scheme needs to additionally satisfy server-privacy. 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 VC 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 $(\delta_y, \pi) \leftarrow \mathsf{VC.Compute}(\mathsf{pk}, \delta_x, \mathsf{C}, w)$.

$\mathcal{D}_2$: $\{\mathsf{pk}, \mathsf{sk}, r, \mathcal{S}_2^{\mathcal{A}, \mathcal{O}_w}(1^\lambda, \tau, \delta_x)\}_\lambda$, 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 $\mathcal{O}_w$ 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 VC protocol is called honest-client server-private if it satisfies the property above against a honest-client.

3.2 Input Dependent Attack and Knowledge Soundness

In this section, we discuss in more detail about how the three different scenarios fit into the VC model. In case I, the VC definition and the soundness property are enough to guarantee the correctness of computation.

However, in case II, things are slightly different. Due to the existence of the server's input, now the verified statement is changed to $C(x, w) = y$, where

w is the input from the server. To ensure correctness in this context, there are currently two mainstream solutions: The first is to embed string w into the setup algorithm, which can be categorized as case I. The disadvantage of this method is that the setup phase is no longer one-time and no longer independent of the server's input. The second is to slightly modify the definition of VC, treating w as an input to the verify algorithm [3]. In this situation, the verifier has to read w entirely, which introduces additional costs in both communication and computation, potentially making the scheme **no longer outsourseable**, due to the increased Verify cost. Therefore, we deem it unreasonable to use w as an input for the verify algorithm.

Thus, neither of these methods solves the problem well. An alternative method is to treat w as if the client would not learn, akin to case III. This method avoids all the complicacies of the previous two methods. We will therefore merge this discussion with case III. However, this case then introduces new challenges to client privacy.

Our discussion begins with the *inadequacy* of soundness in such cases. While soundness ensures the correctness of computation, it only ensures the existence of w rather than the knowledge of w. In other words, the malicious server might complete the computation and proof without actually knowing w. 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. The client has its bid, m, and the server possesses a bid, w. The server computes a circuit as follows: $\mathsf{C}(m, w) := m > w?1 : 0$. This means the circuit outputs 1 if the client's bid is greater and 0 otherwise. A malicious server can exploit this by inputting $w' = \mathsf{Enc}(m + 1)$ instead of the honest w. This manipulation ensures that the server always wins. Additionally, since the proof is generated over $m, \mathsf{Dec}(w')$ homomorphically, the proof is always valid, even though the server does not know m.

More generally, the server can craft a function f, and compute $w' \leftarrow f(\mathsf{Enc}(m))$ homomorphically to generate the malicious input w'. The server can then compute $\mathsf{VC.Compute}(\mathsf{pk}, \mathsf{Enc}(m), \mathsf{C}, f(\mathsf{Enc}(m)))$, and return the result to the client. Because w' is a valid input, 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 VC 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 holds,

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

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

3.3 The Gap from Secure Two-Party Computation (2PC)

Interestingly, VC 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:

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

To illustrate this observation, we first present a toy construction that satisfies client-privacy and knowledge soundness. The verifier uses a FHE scheme to encrypt its input. The server then sends 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 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.

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.

Observation II: Compared to 2PC, VC leakage is bounded by 1 bit.

In the following, we introduce a stronger notion of client-privacy for VC, 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 [24]. 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, $\mathcal{O}_{verify}$ 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: $\hat{b} \leftarrow \mathcal{A}^{\mathcal{O}_{verify}}(c_{m_b}, \mathsf{state})$
6: **return** $b \stackrel{?}{=} \hat{b}$

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

Due to space limitations, the security proof is deferred to Appendix B.

3.4 FHE-SNARK and SNARK-FHE

Now, we revisit the two paradigms for building Verifiable Computation (VC) 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 VC 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. Specifically, 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 VC 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 [43] 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 [11,15] 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 [44] or using much more conservative parameters [15], 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

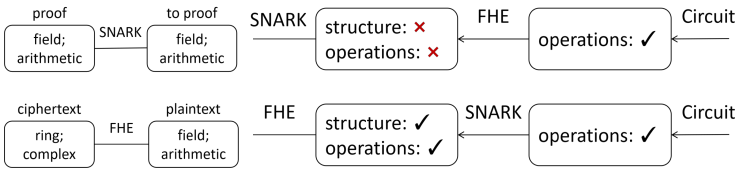

Fig. 4. SNARK-FHE vs FHE-SNARK

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.

In a word, the alignment between algebraic structures and supported operations is pivotal in determining the efficiency of cryptographic schemes. While SNARK-FHE encounters challenges due to structural and operational mismatches, FHE-SNARK benefits from a seamless integration of FHE plaintext arithmetic with SNARK requirements, thereby offering superior performance characteristics.This explains why FHE-SNARK can be orders of magnitude faster than SNARK-FHE, as discussed in Section 1.

4 Polynomial Commitment on Hybrid Values

In our verifiable computation, the server generates a SNARK proof for an encrypted instance. The prover encodes statement and witness as evaluations of polynomials on specific sets and then uses polynomial commitment to provide the oracle for these polynomials.

In this section, we present a polynomial commitment scheme for a multilinear polynomial represented by its evaluations on the hypercube, where some of these evaluations are encrypted and others are in plaintext. It fully harnesses FHE ciphertext packing capabilities while satisfying a strong knowledge soundness additionally allowing for extractability witiout secret key.

4.1 Definition of Polynomial Commitment on Hybrid Values

In this section, we introduce a new variant of polynomial commitments for polynomials of which the evaluations are hybrid values (evaluations on a set

$S_p \subseteq \{0,1\}^n$ are plaintexts, while evaluations on the remaining set $\{0,1\}^n \backslash S_p$ are encrypted). We call such a representation a cryptomix point-value representation, or shortly, cryptomix representation, and formalize it as follows:

Definition 7 (Cryptomix Representation). *Recall that a d-variate multilinear polynomial f is uniquely determined by its evaluations on the hypercube $\{0,1\}^d$. We define $[f]_p = \{f(x)\}_{x\in\{0,1\}^d}$ as the plaintext point-value representation, or shortly, plaintext representation, of f. When some evaluations are encrypted, the resulting representation $[f]mix = (f(x)x \in S_p, \mathsf{Enc}_{\mathsf{pk}}(f(x))_{x\in S_c})$ is called a cryptomix point-value representation (or simply a cryptomix representation), of f, where $S_p \subseteq \{0,1\}^d$ and $S_c = \{0,1\}^d \backslash S_p$. We refer to the set S_p as the plaintext set and the set S_c as the ciphertext set.*

Now we provide a formal definition for polynomial commitment in which the committer only knows a cryptomix representation of the committed polynomial. We assume that its variable number and plaintext set are already known by both the committer and the receiver.

Definition 8 (Polynomial Commitment on Hybrid Values). *A polynomial commitment for multilinear polynomials in cryptomix representation consists of (Setup, Commit, deCommit, Eval) with the exception that the committer has public key of underline encryption scheme and only the cryptomix representation $[f]_{mix}$ as input while the receiver has the corresponding secret key. Furthermore, in the evaluation protocol Eval, since the prover (also the committer) does not know the secret key, it proves for a public plaintext z and ciphertext y^* that the committed polynomial f satisfies $f(z) = y$, where $y = \mathsf{Dec}(y^*)$.*

*Besides the **Completeness** and **Binding** properties defined similarly to standard polynomial commitments, such a commitment needs to satisfy a special variant of knowledge soundness property defined as follows:*

- ***Knowledge Soundness**: Let $\mathsf{Eval} = \langle P, V\rangle$ and $(\mathsf{pk}, \mathsf{sk})$ be an honestly generated key pair of ENC. Then there exists two PPT oracle machines $\mathcal{E}_{w/\ sk}, \mathcal{E}_{w/o\ sk}$, called the extractor with secret key and the extractor without secret key, such that for any pair of PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$, the following holds:*

$$\Pr\left[\begin{array}{r|l} \langle \mathcal{A}_2(st), V(\mathsf{sk})\rangle(\mathsf{pp},\mathsf{pk},C,z,y^*,S_p) = 1 \wedge & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ f(z) \neq y \ \ or\ \ \mathsf{deCom}(\mathsf{pp},C,\tau,\mathsf{sk}) \neq [f]_p \wedge & (C,z,y^*,S_p,st) \leftarrow \mathcal{A}_1(\mathsf{pp},\mathsf{pk}) \\ for\ all\ x \in S_p, f(x) = f^*(x) & ([f]_p,\tau) \leftarrow \mathcal{E}^{\mathcal{A}_1,\mathcal{A}_2}_{w/\ sk}(\mathsf{pp},\mathsf{sk},C,z,y^*,S_p) \\ & \{f^*(x)\}_{x\in S_p} \leftarrow \mathcal{E}^{\mathcal{A}_1,\mathcal{A}_2}_{w/o\ sk}(\mathsf{pp},C,z,y^*,S_p) \end{array}\right] \leq \mathsf{negl}(1^\lambda)$$

where f is the polynomial represented by $[f]_p$ and y is the decryption of y, i.e., $y = \mathsf{Dec}_{\mathsf{sk}}(y^*)$.

A more detailed definition, including the description of each algorithm, is deferred to Appendix C.

Remark 1. Comparison with the polynomial commitment defined in [29]. [29] introduced the polynomial commitment scheme for hidden values, to commit polynomials whose evaluations are pure ciphertexts. This scheme's extractability allows an extractor without the secret key, to extract the ciphertexts of the

evaluations. Specifically, a standard polynomial commitment scheme, such as one based on FRI, can be easily transformed into a polynomial commitment scheme for hidden values by replacing the evaluations by corresponding ciphertexts and performing operations homomorphically. The extractor are also performed homomorphically and now extracts the ciphertext. However, such a scheme cannot be easily transformed into a polynomial commitment scheme for hybrid values, since the verifier lacks the ability to verify that specific evaluations are indeed in plaintext rather than ciphertext. In other words, once the plaintexts and ciphertexts are mixed through homomorphic evaluations, it is difficult to retrieve the plaintexts again without the secret key.

4.2 Construction of Polynomial Commitment on Hybrid Values

In this section, we construct a polynomial commitment for cryptomix-represented polynomials *where the plaintext set satisfies a specific structure*. We will show that such a construction is sufficient for our VC scheme in the following sections. Furthermore, when instantiated with BFV, our construction is friendly for the packed homomorphic operations.

Our construction builds upon the multilinear version of Ligero's polynomial commitment scheme[1] (Brakedown scheme in [34]). Specifically, our construction supports cryptomix-represented multilinear polynomials where their plaintext sets S_p take the form $S_p = S \times \{0,1\}^{n-d}$, where $S \subseteq \{0,1\}^d$ and d is close to $n/2$. In this case, when viewing the cryptomix representation $[f]_{\text{mix}}$ as a $2^d \times 2^{n-d}$ matrix $M = \{u_{i,j}\}_{i\in[2^d],j\in[2^{n-d}]}$ (i.e., $u_{i,j} = f(\mathsf{bin}_n(i \cdot 2^{n-d} + j))$ or $\mathsf{Enc}_{\mathsf{pk}}(f(\mathsf{bin}_n(i \cdot 2^{n-d} + j)))$, depending on whether the corresponding value in $[f]_{\text{mix}}$ is in plaintext or ciphertext), all elements in plaintext collectively form $|S|$ rows of the matrix. For each column of M, the remaining $2^d - |S|$ elements are all ciphertexts. When using an FHE scheme that supports packing, such as BFV or BGV, one can pack these $2^d - |S|$ ciphertexts into one (or more) packed ciphertext(s).

Our construction is transparent, meaning it does not require any trusted setup. The setup algorithm simply selects a random string k to serve as the index for the hash function h_k and outputs $\mathsf{pp} = k$.

Commitment. To commit a polynomial with its cryptomix representation $[f]_{\text{mix}}$, we first view $[f]_{\text{mix}}$ as a $2^d \times 2^{n-d}$ matrix M as described above. Let $u_i = \{u_{i,j}\}_{j\in[2^{n-d}]}$ be the row of M. The committer then encodes each row of M using the Reed-Solomon code scheme Encode to obtain a new matrix $\hat{M}$, as follows (see also Fig.5):

- For row u_i with elements $u_{i,j}$ in plaintext (i.e., $\mathsf{bin}_d(i) \in S$), encode it to obtain the corresponding row $\hat{u}_i$ of $\hat{M}$; that is, $\hat{u}_i \leftarrow \mathsf{Encode}(u_{i,0}, \cdots, u_{i,2^{n-d}-1})$.
- For row u_i with elements $u_{i,j}$ in ciphertext (i.e., $\mathsf{bin}_d(i) \notin S$), homomorphically encode it to obtain the corresponding row $\hat{u}_i$ of $\hat{M}$; this is denoted by $\hat{u}_i \leftarrow \mathsf{HE.Encode}(u_{i,0}, \cdots, u_{i,2^{n-d}-1})$.

Finally, apply a Merkle-hashing to all columns of $\hat{M}$ (where hash is modeled as a random oracle in the security proof, so that the subsequent evaluation protocol

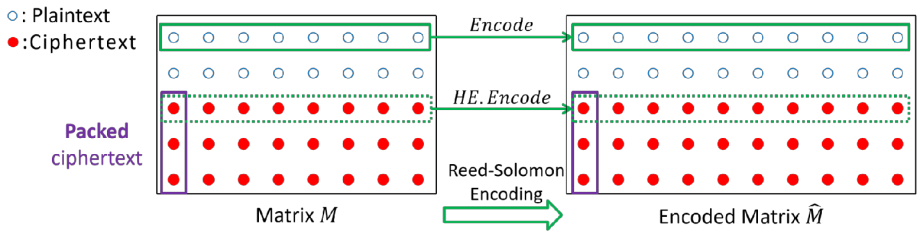

Fig. 5. Encoding of cryptomix representation matrix M

can be seen as an IOP protocol and can be transformed into non-interactive protocol via the BCS transformation [5]). The resulting hash root serves as the commitment C.

The decommitment phase is straightforward: the committer sends $[f]_{\text{mix}}$, $\hat{M}$, and the randomness used during the commitment phase as the decommitment τ. Upon receiving the decommitment, the receiver recomputes the Merkle-hash of $\hat{M}$, verifies the consistency of the resulting hash root with the commitment C, and checks that $\hat{M}$ is close to the encoding of M (for each row, the distance between $[\hat{M}]_p$ and $\mathsf{Encode}([M]_p)$ is less than $\gamma/3$, where γ is the minimal distance of the underline linear code). If all checks pass, the receiver outputs f (recall that the receiver has the secret key sk as input).

Our commitment algorithm exhibits *excellent compatibility with packing*. Suppose that for each column j of M, all elements in ciphertext are already packed into a single ciphertext ct_j. Recall that the encoding algorithm Encode is deterministic and and comprises only linear operations. Therefore, $\mathsf{HE.Encode}$, which consists solely of scalar multiplications, can be directly applied to the packed ciphertexts $(\mathsf{ct}_0, \ldots, \mathsf{ct}_{2^{n-d}-1})$ to obtain the ciphertexts in $\hat{M}$ in a packed form. To further reduce the computation complexity, one might consider using the fast Number Theoretic Transform (NTT) algorithm to accelerate the encoding phase, thereby reducing the computational complexity from $O(2^n)$ to $O(n2^{n/2})$ scalar multiplications (where we assume $d = n/2$). However, the NTT algorithm introduces a logarithmic multiplication depth, which significantly increases the cost of single homomorphic evaluation of ciphertexts. Hence, we instead employ a constant-layer NTT algorithm[5] as done in [29], to achieve a balance between the circuit depth and the number of scalar multiplications (e.g., setting the depth to 3 results in a computational complexity of $O(2^{2n/3})$).

Evaluation. 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.

[5] Recall that an 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.

Importantly, the testing phase only needs to be executed once and is independent of the evaluation point.

Testing phase. Our testing phase mirrors that of Ligero's polynomial commitment scheme, with the exception that certain operations are performed homomorphically over FHE ciphertexts, and the verifier additionally verifies which rows in M are in plaintext.

In this phase, the receiver begins by sending a set of random values $r = \{r_i\}_{i\in[2^d]}$. The committer computes and sends the homomorphic linearly combination of the rows $\{[u_i]_c\}$ of $[M]_c$ with scalars $\{r_i\}$, i.e., $\Sigma_{i\in[2^d]} r_i[u_i]_c$. Let $\{\hat{u}_i\}$ be the rows of $\hat{M}$ and $\{\hat{v}_i\}$ be the columns. Assuming that their exists a row $[\hat{u}_i]_p$ of $[\hat{M}]_p$ that is far from codewords, then, with overwhelming probability, $\Sigma_{i\in[2^d]} r_i[\hat{u}_i]_p$ is also far from the encoding of $\Sigma_{i\in[2^d]} r_i[u_i]_p$. 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, a) the inner production of their decryption $[\hat{v}_i]_p$ and vector r, is consistent with the i-th element of the encoding of $\Sigma_{i\in[2^d]} r_i[u_i]_p$, and b) for each $\mathsf{bin}_d(k) \in S$, the k-th element in $\hat{v}_i$ is in plaintext. The former check ensures that the rows in $\hat{M}$ are close to (encrypted) codewords, while the latter check ensures that most elements in $\{\hat{u}_i\}_{\mathsf{bin}_d(i)\in S}$ are in plaintext, which guarantees the knowledge of the plaintext evaluations on S_p given the decodability of the Reed-Solomon code.

The testing phase also exhibits *excellent compatibility with packing.* For the case that all ciphertexts in one column of M have been pack into a single ciphertext, the committer can homomorphically calculate the inner product of these packed ciphertexts and the random vector r to obtain the encrypted resulting linear combination.

Evaluation phase. The evaluation phase closely resembles the testing phase. From the fact that for any $\alpha = (\alpha_0, \cdots, \alpha_{n-1}) \in \mathbb{F}^n$, $f(\alpha) = r_1 \cdot [M]_p \cdot r_2$, for $r_1 = ((1-\alpha_0, \alpha_0)\otimes\cdots\otimes(1-\alpha_{d-1}, \alpha_{d-1})$ and $r_2 = ((1-\alpha_d, \alpha_d)\otimes\cdots\otimes(1-\alpha_{n-1}, \alpha_{n-1})$ where $[M]_p$ represents the matrix of the plaintext point-value representation of f. Consequently, the receiver can choose r_1 as the r used in the testing phase to obtain $[r_1 \cdot M]_c$ and, finally, compute $f(\alpha)$ itself.

The formalized construction is shown in Appendix. D, and the security proof of following theorem is deferred to Appendix E.

Theorem 2. *The scheme described above is a polynomial commitment for multilinear polynomial with cryptomix representation, as defined in Def.8.*

5 SNARK for VC Instances

For the same reasons outlined in the previous section, to prevent input-dependent attacks, our construction needs to satisfy an additional knowledge-soundness property that allows for the extraction of plaintext elements in the witness without the secret key. Furthermore, to ensure the resulting scheme is practical,

we fully leverage the packing capability available in FHE, thereby significantly reducing the proving cost. We begin by formalizing the knowledge soundness properties customed for SNARK/AoK for VC instances.

5.1 Definition of Arguments of Knowledge for Hybrid Relations

We summarize and formalize the relations that need to be proven in VC and introduce a new notion called hybrid relations.

Definition 9 (Hybrid relation). *For an indexed relation R and an FHE scheme $\mathit{FHE} := (\mathit{KeyGen}, \mathit{Enc}, \mathit{Dec}, \mathit{Eval})$, we define the indexed hybrid relation $[R]_{pk,\ sk}$ with respect to any $(\mathsf{pk}, \mathsf{sk}) \in \mathit{KeyGen}$:*

$$[R]_{pk,sk} = ((\mathbb{i}, S_p), x, w) : (\mathbb{i}, [x]_p, w^*) \in R$$

where S_p is the set indicating the indexes of plaintext elements. This means that for all $i \in S_p$, $w^*[i] := w[i]$ and for $i \notin S_p$, $w^*[i] := [w[i]]_p$. For convenience, we call R the basic relation of $[R]_{\mathsf{pk},\mathsf{sk}}$.

Definition 10 (Argument of Knowledge for Hybrid Relations). *Like standard argument of knowledge (Definition 2 and 3), an argument of knowledge for hybrid relations is also an interactive protocol $\Pi = (\mathit{Setup}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ with the exception that, the deterministic algorithm $\mathcal{I}$ now additionally takes S_p as input, and the prover $\mathcal{P}$ (and verifier $\mathcal{V}$, respectively) algorithms additionally take pk (and sk, respectively) as input.*

*Besides the **Completeness**, an argument of knowledge for hybrid relations needs to satisfy a new variant of knowledge soundness which allows an addition extractability without secret key:*

Knowledge Soundness. *There exist two PPT oracle machines $\mathcal{E}_{w/\ sk}$ and $\mathcal{E}_{w/o\ sk}$, called the extractor with secret key and the extractor without secret key, such that for any pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ and any key pairs $(\mathsf{pk}, \mathsf{sk}) \in \mathit{KeyGen}(1^\lambda)$, the following holds:*

$$\Pr\left[\begin{array}{r|l} \langle \mathcal{A}_2(\mathsf{pp}, \mathsf{pk}, x, st), \mathcal{V}(\mathsf{vp}, \mathsf{sk}, x)\rangle = 1 \wedge & \mathsf{crs} \leftarrow \mathit{Setup}(1^\lambda) \\ (\mathbb{i}, [x]_p, w) \notin R \wedge & ((\mathbb{i}, S_p), x, st) \leftarrow \mathcal{A}_1(\mathsf{pk}) \\ \exists i \in S_p, w[i] \neq w^*[i] & (\mathit{vp}, \mathit{pp}) \leftarrow \mathcal{I}(\mathbb{i}, S_p) \\ & w \leftarrow \mathcal{E}^{\mathcal{A}_1, \mathcal{A}_2}_{w/\ sk}(\mathsf{crs}, \mathbb{i}, S_p, x, \mathsf{sk}) \\ & w^* \leftarrow \mathcal{E}^{\mathcal{A}_1, \mathcal{A}_2}_{w/o\ sk}(\mathsf{crs}, \mathbb{i}, S_p, x) \end{array}\right] \leq negl(1^\lambda)$$

where R is the basic relation of $[R]_{\mathsf{pk},\mathsf{sk}}$.

5.2 Construction for SNARKs for VC instances

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 hybrid relations, we need to demonstrate how to execute PIOPs homomorphically to

reduce the verification of these relations to queries on cryptomix-represented polynomials, and how to reduce the multiplication-depth while achieving great compatibility with packing.

Before presenting a concrete construction, we first show how to run the well-known sumcheck protocol, the core of multivariate polynomial-based PIOPs/SNARKs, on encrypted polynomials while maintaining low multiplication-depth and compatibility with FHE packing.

Sumcheck on encrypted polynomials. The sumcheck protocol efficiently proves the sum of polynomial evaluations over the hypercube. Given an n-variate, low-degree polynomial $f : \mathbb{F}^n \to \mathbb{F}$, the prover aims to prove $y = \Sigma_{x\in\{0,1\}^n} f(x)$ for a verifier who has oracle access to f. The sumcheck protocol provides a way for the verifier to verify the above claim via *a single query* to $\mathcal{O}(f)$.

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}. Now $f_1(x') := f(x', r_{n-1})$ is a $n-1$-variate polynomial. The prover and verifier recursively run the above steps on this new sumcheck claim. And 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 $\mathcal{O}(f)$.

However, when considering encrypted polynomials, the prover now can only computes all evaluations of f in ciphertext homomorphically. 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. Now executing sumcheck homomorphically means that the prover will all evaluations of $\{g_i\}$ used in one round via simple linear combinations on the evaluations obtained in the last round. This computation method increases the multiplicative depth, 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.

Furthermore, the construction is friendly for FHE packing as well. One can write the cryptomix representation of g_i as an $2^{n/2} \times 2^{n/2}$ matrix M and pack each column into a single ciphertext, as done in the polynomial commitment construction. Remind that, in each round, the prover needs to compute:

- The evaluations of f_i^*, which are the sum of $\Sigma_{x'\in\{0,1\}^{n-i}} f(x', k, r')$ for $k \in [t+1]$ and $r' \in \mathbb{F}_p^{i-1}$, where t denotes the degree of f.

This is computed from all $g_j(x', k, r')$, $k \in [t+1]$ and $x' \in \{0,1\}^{n-i}$. Following the fact that $g_j(r_0, \cdots, r_{n-1}) := \Sigma_{x_0,\cdots,x_{n-1}\in\{0,1\}} g_j(x_0, \cdots, x_{n-1}) \cdot \Pi_{i\in[n]}(x_i r_i + (1-x_i)(1-r_i))$, for any fixed $x' \in \{0,1\}^{n-i}$, $g_j(x', k, r')$ equals

$$\Sigma_{x_{n-i},\cdots,x_{n-1}\in\{0,1\}} g_j(x', x_{n-i}, \cdots, x_{n-1}) \cdot l_{x_{n-i},\cdots,x_{n-1}},$$

a linear combination of $\{g_j(x', x_{n-i}, \cdots, x_{n-1})\}_{x_{n-i},\cdots,x_{n-1}\in\{0,1\}}$, where the scalars are independent of x'.

Note that for each $x_{n/2}, \cdots, x_{n-1}$, we already pack $\{g_j(x'_0, \cdots, x'_{n/2-1}, x_{n/2}, \cdots, x_{n-1})\}_{x'_0,\cdots,x'_{n/2-1}\in\{0,1\}}$ into a single ciphertext. We can perform the linear combinations directly on packed ciphertexts (via scalar multiplications) obtaining packed ciphertext results for the first $n/2$ layers. As for the last $n/2$ layers, one can use plaintext-ciphertext multiplications to obtain the results ciphertexts using the packed ciphertext obtained in the $n/2$-th layer. In our construction, we let the sumcheck protocol terminate in the $n/2$-th layer, allowing the verifiers to obtain complete descriptions of $f_{n/2}$ and verify its correctness through a single query to $\mathcal{O}(f)$, adding an acceptable additional cost to the verifier.

From hybrid relations to queries on cryptomix polynomials. In what follows, for any vector x, we denote by $\tilde{x}$ the multilinear polynomial whose representation satisfies $\tilde{x}(i) = x[\text{int}(i)]$ 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 $\tilde{A}(i,j) := a_{\text{int}(i),\text{int}(j)}$ for all $i, j \in \{0,1\}^{\log n}$.

Our construction build upon the well-known PIOP scheme put forward in Spartan. Let $((\mathbb{i}, S_p), x, w)$ be a hybrid relation instance. The prover provides the oracle of cryptomix-represented polynomial $\tilde{z}$ with corresponding plaintext set S'_p (determined by S_p) to the verifier. And the verifier can query the oracle to learn the encrypted evaluations.

Following the Spartan PIOP approach, and with only a negligible soundness error, checking whether $Az \circ Bz = Cz$ is equivalent to checking whether the following identity holds with the oracle access to $\tilde{z}$:

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

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

Remind that some elements in z are encrypted for the prover, both parties run Spartan PIOP for above checking 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 for encrypted polynomials as underline sumcheck protocol for better efficiency and communications.

As a result, the checking is finally reduced to the queries to the oracle of cryptomix-represented polynomial $\tilde{z}$ (and the oracles of plaintext polynomials $\tilde{A}, \tilde{B}, \tilde{C}$, which are provided by the algorithm $\mathcal{I}$). And the oracle of cryptomix-represented polynomial can be instantiated using polynomial commitment for cryptomix-represented polynomial (and the oracles of plaintext polynomials can be instantiated using standard polynomial commitment and the technique provided in Spartan and Brakedown on committing sparse polynomials).

Furthermore, given the plaintext representation of $[\tilde{z}]_p$ (the plaintext part of $[\tilde{z}]_{\text{mix}}$), one can easily extract the entire witness in plaintext $[w]_p$ (the plaintext part $\{w[i]\}_{i\in[S_p]}$ of w, respectively). Due to that running Spartan PIOP homomorphically won't breaks its soundness property, when the prover generates an convincing proof, the extracted witness satisfies the requirement of the knowledge soundness.

Argument of knowledge for hybrid relations

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

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

$\mathcal{I}(crs, \mathbb{i})$: Assume that the sizes of statement and witness, $|x|$ and $|w|$, are $k_x\sqrt{N}, k_w\sqrt{N}$, and that $|S_p|$ is $k'\sqrt{N}$ for some integers k_x, k_w, k'. (This requirement can always be met by padding with an appropriate number of "0"s) Choose a permutation t over $[n]$ with permutation matrix T such that when we rewrite the vector $Tz = (z[t(0)], z[t(1)], \cdots, z[t(n-1)])$ as a $\sqrt{N} \times \sqrt{N}$ matrix M_z, it satisfies that:

1. The plaintext elements of w lie in the first k' rows of M_z;
2. The ciphertext elements of w lie in the $(k'+1)$-th row to k_w-th row of M_z;
3. The elements of x lie in the (k_w+1)-th row to $\sqrt{N}$-th row of M_z;

Run the the Index algorithm $\mathcal{I}$ of Spartan's PIOP on input $\mathbb{i}' = (\mathbb{F}, AT^{-1}, BT^{-1}, CT^{-1}, M, N)$ and use a (standard) polynomial commitment scheme to instantiate the polynomial oracles. This involves committing to the sparse multilinear polynomials $\widetilde{AT^{-1}}, \widetilde{BT^{-1}}, \widetilde{CT^{-1}}$ using the polynomial commitment and the techniques provided in Brakedown[34] (or Spartan[51]). Output the commitments and $S'_p = S \times \{0,1\}^{n/2}$ where $S = \{\mathsf{bin}_{n/2}(i) | i \in [k']\}$ as the verifier key vk, and output these three polynomials as the prover key pk.

$\langle P, V\rangle$: The prover sends the commitment of cryptomix-represented polynomial $\tilde{z}'$ with plaintext set S'_p to the verifier, where $z' = (w', z') = T \cdot (w, z)$. Both parties run Spartan's PIOP homomorphically for the hybrid R1CS instance $(\mathbb{i}', [k'\sqrt{N}], x', w')$ as described before, and when the verifier want to query $\tilde{z}$ and $\tilde{A}$,$\tilde{B}$,$\tilde{C}$, it runs the evaluation protocols of underline polynomial commitment respectively.

Fig. 6. Argument of knowledge for hybrid relation

We provide a formalized description for the resulting PIOP for hybrid relations and its properties in Appendix F.2,F.1.

Last step to our argument of knowledge for hybrid relation. While we have reduce the verification of VC instance to queries on cryptomix-represented polynomials, we only provide a polynomial commitment construction for cryptomix-represented polynomials whose plaintext set satisfy a specific structure, i.e., $S_p = S \times \{0,1\}^{n-d}$ for some set S and integer d close to $n/2$. Fortunately, when using the R1CS relation $Az \circ Bz = Cz$ to arithmetize the unproven relation, the order of elements in z (containing the statement and witness) can be adjusted via any permutation T. The R1CS relation is then modified to:

$$(AT^{-1} \cdot Tz) \circ (BT^{-1} \cdot Tz) = CT^{-1} \cdot Tz.$$

As a result, our construction of argument of knowledge for hybrid relation is constructed in Figture.6.

Theorem 3. *The protocol provided in Figure.6 is an argument of knowledge for hybrid relations.*

proof sketch. Roughly, our construction can be seen as a compilation of a variant of Spartan's PIOP for hybrid relations and the polynomial commitment for cryptomix-represented polynomials. The extractor first extract the representation of $\tilde{z}$ from the knowledge soundness of underline polynomial commitment and then extract the witness directly from these representations. The extractor with $\mathsf{sk}_{\mathsf{FHE}}$ can extract the plaintext representation of $\tilde{z}$, and thereby can extract the entire witness in plaintext; while the extractor without $\mathsf{sk}_{\mathsf{FHE}}$ can only extract the plaintext part in $[\tilde{z}]_{\mathrm{mix}}$, and thereby can only extract the plaintext part of witness. The knowledge soundness of underline polynomial commitment and Spartan's PIOP ensures the correctness of extracted witness. A more detailed discussion can be find in Appendix.F. □

Non-interactive and Fiat-Shamir heuristic. Our construction is public coin, enabling the use of the Fiat-Shamir heuristic [23] 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[5,27]. Furthermore, since our construction can be viewed as an encrypted Spartan scheme (with Ligero/Shockwave polynomial commitment), it exhibits the same knowledge error.

Zero-knowledge. Similar to Brakedown (and Spartan), our construction can be made zero-knowledge using standard techniques with minimal overhead[34,1,17]. There are roughly three potential sources of leakage, and we outline how to mitigate them to achieve zero-knowledge (and omit the details since the techniques are standard):

- Leakage in the polynomial commitment: Both the testing and evaluation phases can leak combinations of the witness. We can add blinding vectors during commitment to prevent this leakage, as demonstrated in [1].
- Leakage in the PIOP protocol: The sumcheck protocol can also leak combinations of the witness. We can add blinding polynomials to the sumcheck protocols to avoid this, as shown in [17].
- Leakage in the noise of encryption: Since we perform evaluations over homomorphic ciphertexts, the noise might leak information about the witness. Therefore, we require the HE scheme to support circuit privacy, meaning that all ciphertexts of the SNARKs exposed to the verifier can be re-randomized.

Optimization. In the Spartan's PIOP construction, one need to check that the statement encoded in the polynomial $\hat{z}$ is indeed x. However, 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_z, with x composing several rows of M_z. It then uses the hash of the encoding of the rows as the commitment. In the testing phase, the prover sends a random combination of rows of M_z and demonstrates that its encoding is close to the same combination of the encodings of the rows. Since the verifier knows the statement x itself, it can compute the combination and check that its encoding is close to the combination

Verifiable Computation

$\mathsf{VC.Setup}(1^\lambda, \mathsf{C})$: Run the setup algorithm and the deterministic algorithm $\mathcal{I}$ of our SNARKs on the R1CS arithmetization of C as well as plaintext set S_p to obtain crs and the prove/verify 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 $\mathsf{PK} = (crs, \mathsf{pk}, \mathsf{pp}), \mathsf{SK} = (crs, \mathsf{sk}, \mathsf{vp})$.

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

$\mathsf{VC.Compute}(\mathsf{PK}, \mathsf{Enc}(m), \mathsf{C}, y)$:

1. Compute the output of the circuit $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{out}) \leftarrow \mathsf{Eval}_{\mathsf{pk}}(C(\cdot, y), \mathsf{Enc}_{\mathsf{pk}}(x))$.
2. Retrieve the witness w of R1CS hybrid relation instance from above computation (the elements of w are either in plaintext or in ciphertext).
3. Compute the proof π for R1CS hybrid relation instance $((\mathbb{i}, S_p), (\mathsf{Enc}(x), \mathsf{Enc}(\mathsf{out})), w)$ using our SNARK scheme for hybrid relations.
4. Send $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{out}), \pi$ to the client.

$\mathsf{VC.Verify}(\mathsf{SK}, \mathsf{Enc}(y), \mathsf{st}, \pi)$: Check the validness of π. If the proof is valid, output $\mathsf{out} \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(\mathsf{out}))$. Otherwise, output $\perp$.

Fig. 7. Verifiable Computation

of the corresponding encoding rows, demonstrating that these encodings are indeed (close to) the codewords of x.

Compatibility with FHE Packing Techniques. Remind that both the underlying polynomial commitment scheme and the sumcheck protocol are compatible with FHE packing. The resulting SNARK scheme is also compatible with packing. 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.

6 Construction of Verifiable Computation

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 hybrid relations. The formal construction is shown in Fig.6:

Theorem 4. *The protocol put forward in Figure.7 is a verifiable computation with knowledge soundness.*

Completeness follows directly from our construction, while the knowledge soundness follows the knowledge soundness (without the secret key) of our SNARK scheme. And the client-privacy follows directly from the CPA security of the FHE scheme.

Server-privacy. Our construction can also achieve server-privacy against an honest client by using a zero-knowledge version of the underlying SNARK scheme and employing circuit-private FHE, as well as re-randomizing the noise of the encoding output. 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.

7 Implementation

BGV/BFV parameter. In our evaluation, we use $t \approx 2^{50}$ being a prime as our 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 "SNARKs for VC instance"), 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.

Polynomial commitment on hybrid values. For a n-variate multilinear polynomial with its cryptomix 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 $2^{n/2}$ ciphertexts into a single packed ciphertext (assumes that the input has been packed already), then the committing algorithm requires $O(t \cdot 2^{\frac{n}{2}+\frac{n}{2t}})$ scalar multiplications when using a t-level NTT, and the evaluation requires $O(2^{n/2})$ scalar multiplications and additions. As a comparison, the FRI-based polynomial commitment requires at least $O(2^{n/2})$ 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 underline Reed-Solomon code is $1/4$ and we adapt 3-level 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. The computation and communication costs are shown as follows:

[6] 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.

security param.	commit	prove[6]	verify	communication
$\lambda = 80$	$2^{16}\,\mathrm{HE}_{sca}$/level+hash	$2^{11}\,\mathrm{HE}_{inn}$	$388\,\mathrm{HE}_{dec}+V_{hash}+V_{remain}$	$388(\mathrm{Enc}+\pi_{merkle})$
$\lambda = 100$	$2^{16}\,\mathrm{HE}_{sca}$/level+hash	$2^{11}\,\mathrm{HE}_{inn}$	$482\,\mathrm{HE}_{dec}+V_{hash}+V_{remain}$	$482(\mathrm{Enc}+\pi_{merkle})$

Table 1. Performance of our polynomial commitment on hybrid values. $\mathrm{HE}_{sca}, \mathrm{HE}_{inn}$ and HE_{dec} mean the cost of single scalar multiplication, plaintext-ciphertext inner-product, and decryption on packed ciphertexts. $\cdot$/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.

We run our polynomial commitment on a FHE field, 50-bit prime field, for 80-bit and 100-bit security respectively. In the former case, to ensure the 80-bit security, our evaluation protocol is run over the Galois degree-2 extension field of this field. And in the latter case, our evaluation protocol is run over the Galois degree-3 extension field of this field. Note that our commitment algorithm is still run 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 estimated performance, measured on a GCP instance N4 with CPU Intel Emerald Rapids with 16 GB RAM, is shown as follows:

security param.	commit	prove	verify	communication
$\lambda = 80$	22.6s	365s	350ms	46.8MB
$\lambda = 100$	22.6s	548s	430ms	58.1MB

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

SNARKs for VC instance. We provide the performance of our SNARKs for the VC 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 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 $z = (w, z)$, compute Az, Bz, Cz, which costs $3 \cdot 2^{20}$ scalar multiplications over unpacked ciphertexts. [level-6 $\rightarrow$ level-5]
2. Packing: for z, Az, Bz, Cz (containing $4 \cdot 2^{20}$ ciphertexts in total), pack 1000 ciphertexts into a single ciphertext, resulting 2^{12} packed ciphertexts. [Level-5 $\rightarrow$ level-4]
3. Sumcheck for $\Sigma_x eq(x, \tau)F(x)$: 1). [level-4 $\rightarrow$ level-3]: 15×2^{13} scalar multiplications and additions. 2). [level-3 $\rightarrow$ level-2]: 2^{12} ciphertext-ciphertext multiplications and additions. 3). [level-2 $\rightarrow$ level-1]: 2^{12} plaintext-ciphertext vector multiplications and additions.
4. Sumcheck for $\Sigma_x M(r, x) \cdot z(z)$: 1). [level-4 $\rightarrow$ level-3]: 15×2^{11} scalar multiplications and additions. 2). [level-3 $\rightarrow$ level-2]: 3×2^{10} plaintext-ciphertext vector multiplications and additions.

5. Polynomial commitment for $\tilde{z}$: only 1 query [level-4 $\rightarrow$ 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 underline polynomial commitment on hybrid values and some other verification contained in the standard SNARKs for plaintext instance.

Suppose that the VC instance is over a FHE-friend field, $\mathbb{F}_p$, with a 50-bit prime p. To achieve 100-bit security, our construction is over the Galois degree-3 extension field of $\mathbb{F}_p$. The estimated performance, measured on a GCP instance N4 with CPU Intel Emerald Rapids with 16 GB RAM, is shown as follows:

security param.	prove	verify	communication
$\lambda = 100$	12934s $\approx$ 3.6h	2.7s	65.3MB

Table 3. Single core performance of our polynomial commitment on hybrid values. Moreover, the implementation can be further accelerated by multi-core processors since all operations are parallelizable.

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. 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)
8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
9. 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)
10. 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)
11. 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)
12. 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
13. 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
14. 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
15. 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)
16. 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)
17. 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

18. 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)
19. 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
20. 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
21. 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)
22. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), https://ia.cr/2012/144
23. 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
24. 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
25. 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
26. 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)
27. 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
28. 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
29. 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

30. 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
31. 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
32. 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)
33. 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
34. 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
35. 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
36. 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
37. Halevi, S., Shoup, V.: Bootstrapping for HElib. Journal of Cryptology **34**(1), 7 (Jan 2021)
38. 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)
39. Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. In: ASIACRYPT 2021. p. 608–639. Springer (2021)
40. 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
41. 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)
42. 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

43. 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)
44. 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)
45. 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)
46. 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)
47. Liu, Z., Tromer, E., Wang, Y.: Group oblivious message retrieval. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 4367–4385 (2024)
48. 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)
49. 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
50. 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)
51. 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
52. 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
53. 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
54. 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

A Cryptographic Primitives

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

A.1 Secure Two-Party Computation

We follow the the real/ideal world paradigm as outlined in [33] 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 11 (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 $\tau \in \{0,1\}^*$.*

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

A.2 B/FV Leveled Homomorphic Encryption

The BFV leveled homomorphic encryption scheme is first introduced in [7] using standard LWE assumption, and later adapted to ring LWE assumption by [22].

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. [7]

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 $\boldsymbol{m} = (m_1, \ldots, m_N) \in \mathbb{Z}_t^N$. 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 $\boldsymbol{m}$ 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., [7,22,39]) 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 $\mathsf{BFV.Dec}$ 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 $\mathsf{BFV.Enc}$ 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) $\mathsf{BFV.Dec}(\mathsf{rot}(\mathsf{ct}, j))[i] = \mathsf{BFV.Dec}(\mathsf{ct})[i + j \pmod N], \forall i, j \in [N]$
- (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 $D > 0$.

[7] Note that this is the relationship between t, N does not need to be satisfied in general (e.g., see [37,36] 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.

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 [7,22,39,37,36] 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., $\mathsf{BFV.Dec}$).

B Security proof for Theorem.1

We consider a VC scheme as a 2PC scheme where the client executes $\mathsf{VC.ProbGen}$ and $\mathsf{VC.Verify}$, and the server executes $\mathsf{VC.Compute}$. 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,

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

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 $\mathsf{HYD}_1(1^\lambda) \overset{c}{\approx} \mathsf{Ideal}_{\mathcal{F}_{2pc},S}(m_1, m_2, \tau)$ due to the client-privacy with verification oracle of the VC 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 VC scheme, we know that both outputs are identical, which concludes the proof.

C Definition of Polynomial Commitment on Hybrid Values

Definition 12 (Polynomial Commitment on Hybrid Values). *A polynomial commitment for multilinear polynomials in cryptomix representation with respect to an encryption scheme* ENC *consists of four algorithms*

$$\mathsf{COM} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{deCom}, \mathsf{Eval}) :$$

- $\mathsf{Setup}(1^\lambda)$*: On input the security parameter* 1^λ*, the* Setup *algorithm outputs a public parameter* pp*.*
- $\mathsf{Com}(\mathsf{pp}, \mathsf{pk}, [f]_{mix})$*: On input the public parameter* pp*, an encryption public key* pk*, and a multilinear polynomial* f *in its cryptomix representation* $[f]_{mix} = (\{f(x)\}_{x\in S_p}, \{\mathsf{Enc}_{\mathsf{pk}}(f(x))\}_{x\in S_c})$*, 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 its plaintext representation* $[f]_p = \{f(x)\}_{x\in\{0,1\}^d}$*, or* $\perp$ *if the decommitment is invalid.*
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{pk}, C, z, y^*, S_p; \mathsf{sk}, f, \tau)$*:* Eval *is an interactive protocol*

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

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

  $$R_{eval} = \{(\mathsf{pp}, \mathsf{pk}, C, z, y^*; \mathsf{sk}, f, \tau) : \mathsf{deCom}(\mathsf{pp}, C, \tau, \mathsf{sk}) = [f]_p \wedge y = \mathsf{Dec}_{\mathsf{sk}}(y^*) \wedge f(z) = y\}$$

  where f *is the multilinear polynomial of which plaintext representation is* $[f]_p$*.*

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

– **Completeness**: For all $\lambda \in \mathbb{N}, d \in \mathbb{N}$, honestly generated key pair $(\mathsf{pk}, \mathsf{sk})$ of ENC, multilinear polynomial $f \in \mathbb{F}[X_1, \cdots, X_d]$ with its cryptomix representation $[f]_{mix}$ and plaintext set S_p, and any $z \in \mathbb{F}^d$,

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

– **Computational Binding**: For all $\lambda \in \mathbb{N}, d \in \mathbb{N}$, honestly generated key pair $(\mathsf{pk}, \mathsf{sk})$ of ENC, and any polynomial-size adversary $\mathcal{A}$,

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

– **Knowledge Soundness**: Let $\mathsf{Eval} = \langle P, V \rangle$ and $(\mathsf{pk}, \mathsf{sk})$ be an honestly generated key pair of ENC. Then there exists two PPT oracle machines $\mathcal{E}_{\text{w/ sk}}, \mathcal{E}_{\text{w/o sk}}$, called the extractor with secret key and the extractor without secret key, such that for any pair of PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$, the following holds:

$$\Pr\left[\begin{array}{l|r} \langle \mathcal{A}_2(st), V(\mathsf{sk}) \rangle(\mathsf{pp}, \mathsf{pk}, C, z, y^*, S_p) = 1 \wedge & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ f(z) \neq y \text{ or } \mathsf{deCom}(\mathsf{pp}, C, \tau, \mathsf{sk}) \neq [f]_p \wedge & (C, z, y^*, S_p, st) \leftarrow \mathcal{A}_1(\mathsf{pp}, \mathsf{pk}) \\ \text{for all } x \in S_p, f(x) = f^*(x) & ([f]_p, \tau) \leftarrow \mathcal{E}^{\mathcal{A}_1, \mathcal{A}_2}_{\text{w/ sk}}(\mathsf{pp}, \mathsf{sk}, C, z, y^*, S_p) \\ & \{f^*(x)\}_{x \in S_p} \leftarrow \mathcal{E}^{\mathcal{A}_1, \mathcal{A}_2}_{\text{w/o sk}}(\mathsf{pp}, C, z, y^*, S_p) \end{array}\right] \leq \mathsf{negl}(1^\lambda)$$

where f is the polynomial represented by $[f]_p$ and y is the decryption of y, i.e., $y = \mathsf{Dec}_{\mathsf{sk}}(y^*)$.

D Formalized Construction of Polynomial Commitment on Hybrid Values

The formalized construction of our polynomial commitment is shown in Fig.8.

E Security Proof for Theorem. 2

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 values, $\hat{M}$ and $\hat{M}'$, that hash to the same root, thereby breaking the collision-resistance of the hash function, or provide two distinct matrices, M and M', whose encoding matrices are both close to the same matrix $[\hat{M}]_p$. This would violate the relative distance of the linear code.

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. We first discuss the existence of extractor with secret key:

As discussed in [34] 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.

Polynomial commitment on hybrid values

Public Input: the number of variables $n \in \mathbb{N}$ in the polynomial, the plaintext set S_p in the form of $S_p = S \times \{0,1\}^{n-d}$ where $S \subseteq \{0,1\}^d$, and the public key pk of the FHE scheme;
Private Input for the committer: the cryptomix representation $[f]_{\text{mix}}$ of the polynomial f;
Private Input for the receiver: the secret key sk of the FHE.

Setup: Choose a random hash function h used for Merkle-hashing, and output $\mathsf{pp} = h$.

Commit: The committer proceeds as follows:

1. Rewrite $[f]_{\text{mix}}$, 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 $\hat{u}_i \leftarrow \mathsf{HE.Encode}(u_{i,1}, \cdots, u_{i,2^{n-d}})$.
   (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 of the columns of $\hat{M}$ via $\mathsf{HE.Encode}(\mathsf{ct}_1, \cdots, \mathsf{ct}_{2^{n-d}})$.
   (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 $[f]_{\text{mix}}$ and $\hat{M}$, constitutes the decommitment τ.

Decommit: 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 $[\hat{M}]_p$ and $\mathsf{Encode}([M]_p)$ is less than $\gamma/3$, where γ is the minimal distance of the linear code. If both checks pass, output $[f]_p$.

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 $[M]_c$ with scalars $\{r_i\}$, i.e., $\Sigma_{i\in[2^d]} r_i \cdot [u_i]_c$.
   (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 column $\hat{v}_j$ of $\hat{M}$: a) the inner product $\langle r, [\hat{v}_j]_p \rangle$ 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 $f(\alpha)$ with point $\alpha = (\alpha_0, \cdots, \alpha_{n-1}) \in \mathbb{F}^n$, the receiver proceeds as follows:

1. The receiver computes $r_1 = ((1-\alpha_0, \alpha_0) \otimes (1-\alpha_1, \alpha_1) \otimes \cdots \otimes (1-\alpha_{d-1}, \alpha_{d-1})$ and $r_2 = ((1-\alpha_d, \alpha_d) \otimes \cdots \otimes (1-\alpha_{n-1}, \alpha_{n-1})$. The receiver runs a testing phase with the committer, except that r is set to r_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 $f(\alpha) = \langle r_2, u^* \rangle$.

Fig. 8. Polynomial commitment on hybrid values

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 $[\hat{M}]_p$ is sufficiently close to a codeword, the extractor can decode them to obtain $[M]_p$, and subsequently recover $[f]_p$. We will demonstrate the following: 1) for each row of $[\hat{M}]_p$, 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 decoded matrix M and the randomness, constitutes a valid decommitment), 2) $[M]_p$ represents a polynomial f that satisfies $f(z) = y$, 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. To prove that the rows of $[\hat{M}]_p$ are close to codewords, we introduce a lemma proved in Ligero:

Lemma 1 (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 $\Delta := \{j \in [n] | \exists i \in [m] \text{ s.t. } \hat{u}'_i[j] \neq \hat{u}_i[j]\}$ is lager 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 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 row of extracted encoding metrix $\hat{M}$.*

Proof. Assume that there exists a row $\hat{u}'$ of the extracted encoding matrix $\hat{M}$ such that $d(c', [\hat{u}']_p) \geq \gamma/3$ 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, $d(\Sigma_{i\in 2^d} r_i \cdot [\hat{u}_i]_p, L) < \gamma/3$.
- **Case II.**Given randomness $\{r_i\}_{i\in[2^d]}$ chosen by the receiver, $d(\Sigma_{i\in 2^d} r_i \cdot [\hat{u}_i]_p, L) \geq \gamma/3$.

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.1, 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 $\Sigma_{i\in 2^d} r_i \cdot [\hat{u}_i]_p$ 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.

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 f satisfies $f(z) = y$. Otherwise, suppose that $f(z) \neq y$, it must be that $\Sigma_i r'_i \cdot u_i \neq u^*$, where r'_i is the vector sent by the receiver, and u^* is the decoding of the vector sent by the committer. From the properties of linear code and the fact that $d(\Sigma_i r'_i \cdot [\hat{u}_i]_p, \Sigma_i r'_i c_i) < \lambda/3$, we have that $d(\Sigma_i r'_i \cdot [\hat{u}_i]_p, \mathsf{Encode}(u^*)) \geq 2\lambda/3$. 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[35], 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 $[\hat{M}]_p$, and then decoding it to obtain $[M]_p$, the plaintext representation of f. 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 $\mathsf{bin}_d(i) \in S$, 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 $\{\hat{u}_i\}_{\mathsf{bin}_d(i) \in S}$ to break the decoding, the receiver checks in step 5 of the testing phase that the corresponding elements of $\{\hat{u}_i\}_{\mathsf{bin}_d(i) \in S}$ are indeed plaintexts. Therefore, we can modify above analysis by replacing any ciphertexts in $\{\hat{u}_i\}_{\mathsf{bin}_d(i) \in S}$ 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 $\{\hat{u}_i\}_{\mathsf{bin}_d(i) \in S}$ for decoding.

F PIOP for hybrid relations

F.1 Definition of PIOP for hybrid relations

Similar to standard PIOP system, we formalize the variant of PIOP for hybrid relations as follows:

Definition 13 (PIOP for hybrid relations). *A PIOP system Π for hybrid relations $[R]_{\mathsf{pk},\mathsf{sk}}$ consists of the following algorithms:*

- $\mathcal{I}(1^\lambda, \mathtt{i}, S_p)$*: $\mathcal{I}$ is a deterministic algorithm that on input $\mathtt{i}, S_p$, outputs an encoding polynomial f and a public parameter pp containing several sets, $pp = \{S_{p_j}\}_j$.*
- $\langle P, V\rangle$ *is an interactive protocols where the prover $\mathcal{P}$ (and verifier $\mathcal{V}$, respectively) algorithms additionally take* pk *(and* sk*, respectively) as input and:*
  - *The prover algorithm P returns a message as follows*

$$P(f, \{[f_j]_{mix}\}_j, tr', x, w)$$

in round i, where $\{[f_j]_{mix}\}_j$ is the set of polynomials with cryptomix representation contained in current transcript and their plaintext sets are exactly S_{p_j}, and tr' is the remaining part of the current transcript. Furthermore, the returned message might contain several polynomials represented by their cryptomix representation $[f_j]_{mix}$.

  - *The verifier algorithm, V, returns*

$$V^{O_p(f), O_{mix}(\{[f_j]_{mix}, S_{p_j}\}_j)}(pp, tr', x)$$

in round i. Here $O_p(f)$ is the polynomial oracle that, on query y, outputs $z = f(y)$, and $O_{mix}(\{[f_j]_{mix}, S_{p_j}\}_j)$ is the cryptomix polynomial oracle that, on query y, if the plaintext set of $[f_j]_{mix}$ is indeed S_{p_j}, it outputs $[z]_c$ where $z = f_j(y)$ and f_j is the polynomial represented by $[f_j]_{mix}$; otherwise, it outputs $\perp$.

Besides the ***Completeness****, we require such a PIOP system to satisfy the* ***knowledge soundness*** *property defined similar to Definition.10, where the extractor (and partial extractor, respectively) have the ability to invoke and rewind the adversary, and obtain the plaintext representation of f_j (the plaintext part of $[f_j]_{mix}$, respective) sent by the adversary.*

A PIOP for hybrid relations can be *public-coin*, similar to a PIOP. This means that: a) all messages sent by the verifier are random strings (with the proper length); and b) all queries to the polynomials depend only on the messages sent by the verifier, which implies that the prover knows these queries.

Similar to the standard PIOP compilation, which demonstrates that by replacing the polynomial oracle with polynomial commitments, one can obtain an interactive argument of knowledge without oracles, we have a similar compilation for PIOP for hybrid relations and polynomial commitment on hybrid values. Specifically, for each oracle of a polynomial with cryptomix representation, we replace it with a polynomial commitment for hybrid values. That is, for every polynomial with cryptomix representation sent by the prover in the PIOP for hybrid relations, the prover sends a commitment to that polynomial (with cryptomix representation) to the verifier instead. Whenever the verifier wants to query the polynomial, the verifier and prover execute the evaluation protocol of the underlying polynomial commitment scheme. If the verifier fails to obtain an evaluation (means that the evaluation protocol outputs $\perp$), it aborts directly.

Theorem 5 (PIOP Compilation for Hybrid Relations). *If the polynomial commitment on hybrid values has knowledge soundness property, and the PIOP protocol for hybrid relations is also knowledge sound, then the result of the compiler is a secure (non-oracle) argument with knowledge soundness defined above.*

Proof (Proof sketch). We can construct the extractor for the resulting argument using the extractors from the PIOP and the polynomial commitment scheme directly. Specifically, each time the prover sends a commitment of a polynomial with cryptomix representation, we use the extractor of the polynomial commitment to extract the polynomial and send the polynomial and polynomial oracle instead. Now, an adversary for the resulting argument becomes an adversary for the underlying PIOP system for hybrid relations, and we can use the extractor of the PIOP to extract the witness. Suppose that the adversary convinces the verifier; the extractability ensures that queries to a polynomial are consistent with the extracted polynomials. Therefore, the new adversary can convince the verifier of the PIOP for hybrid relations with the same (or even higher) probability, except for a negligible difference. Consequently, the extractor of the PIOP will extract a witness that satisfies the relation. The extractability without secret key can be proved similarly except that the extractor of polynomial commitment only extracts the plaintext part and the polynomial oracle are now constructed using the adversary.

F.2 construction of PIOP for hybrid relations

The resulting PIOP for hybrid relations in Section 5.2 is formalized as follows:

Theorem 6. *The construction in Figure.9 is a secure PIOP system for hybrid relations.*

Proof (Proof sketch). We begin with a concise proof demonstrating knowledge soundness with secret key. The extractor $\mathcal{E}$ executes the adversary to obtain the cryptomix representation of $\hat{z}$. It then employs the secret key to extract all elements in z and directly outputs the witness w contained within z. The correctness of w stems from the soundness of the (encrypted) Spartan PIOP protocol: Observe that the check $Az \circ Bz = Cz$ is equivalent to verifying "for all $x \in \{0,1\}^d$, $F(x) = \Sigma_{y\in\{0,1\}^s} \tilde{A}(x,y) \cdot \tilde{z}(y) \times \Sigma_{y\in\{0,1\}^s} \tilde{B}(x,y) \cdot \tilde{z}(y) - \Sigma_{y\in\{0,1\}^s} \tilde{C}(x,y) \cdot \tilde{z}(y) = 0$". The Spartan protocol can be viewed as a PIOP scheme tailored for this relation w.r.t. the polynomial oracle of $\tilde{z}$. Since our construction can be interpreted as an encrypted variant of Spartan, its soundness mirrors that of Spartan. Specifically, given the secret key, any prover capable of breaking our PIOP can be transformed into a prover that breaks the standard Spartan PIOP by decrypting all messages sent by the prover. Consequently, the witness extracted using the secret key from a valid adversary is correct.

Extractability without secret key follows directly from above extraction. Note that, in Spartan's PIOP scheme, the witnesses are encoded in the representation

PIOP for hybrid relations based on Spartan

Index: $(\mathbb{i} = (\mathbb{F}, A, B, C, M, N), S_p)$,
Statement: x. Witness: w.

$\mathsf{Setup}(1^\lambda, \mathbb{i}, S_p)$: Run the setup 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.) Additionally, setup algorithm also outputs a set S'_p that indicates the plaintext set of the polynomial $\tilde{z}$ for $z = (x, w)$. Note that S'_p is determined by S_p and is independent of the specific values of z.

$\langle P, V \rangle$:

1. Let $z = (w, x)$. The prover sends the polynomial oracle of $\tilde{z}$ with cryptomix representation 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 computes Az, Bz, Cz to obtain the cryptomix 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 hybrid values 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:

$$\Sigma_{y\in\{0,1\}^n} \tilde{A}(r_x, y)\cdot\tilde{z}(y) = [z_1]_p, \quad \Sigma_{y\in\{0,1\}^n} \tilde{B}(r_x, y)\cdot\tilde{z}(y) = [z_2]_p, \quad \Sigma_{y\in\{0,1\}^n} \tilde{C}(r_x, y)\cdot\tilde{z}(y) = [z_3]_p$$

   The verifier chooses and sends three randomness r_1, r_2, r_3 and both prover and verifier runs the following sumcheck on hybrid values

$$\Sigma_{y\in\{0,1\}^n} (r_1\tilde{A}(r_x, y) + r_2\tilde{B}(r_x, y) + r_3\tilde{C}(r_x, y))\cdot\tilde{z}(y) = r_1[z_1]_p + r_2[z_2]_p + r_3[z_3]_p$$

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 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 $r^* \in \mathbb{F}^{\log|x|}$.

Fig. 9. PIOP for hybrid relations

of the polynomial $\tilde{z}$, and its plaintext portion is exactly the plaintext part of the witness. Therefore, the extractor can directly output these elements, and the soundness of Spartan ensures that the extracted elements are identical to the corresponding part of the witness extracted by the extractor with the secret key.