$\mathcal{Z}$HE: Efficient Zero-Knowledge Proofs for HE Evaluations*

Zhelei Zhou[1,2], Yun Li[2], Yuchen Wang[2], Zhaomin Yang[2], Bingsheng Zhang[1,✉], Cheng Hong[2,✉], Tao Wei[2], Wenguang Chen[2]
[1]Zhejiang University [2]Ant Group
bingsheng@zju.edu.cn, vince.hc@antgroup.com

July 17, 2025

Abstract

Homomorphic Encryption (HE) allows computations on encrypted data without decryption. It can be used where the users' information are to be processed by an untrustful server, and has been a popular choice in privacy-preserving applications. However, in order to obtain meaningful results, we have to assume an honest-but-curious server, i.e., it will faithfully follow what was asked to do. If the server is malicious, there is no guarantee that the computed result is correct. The notion of verifiable HE (vHE) is introduced to detect malicious server's behaviors, but current vHE schemes are either more than four orders of magnitude slower than the underlying HE operations (Atapoor et. al, CIC 2024) or fast but incompatible with server-side private inputs (Chatel et. al, CCS 2024).

In this work, we propose a vHE framework $\mathcal{Z}$HE: efficient Zero-Knowledge Proofs (ZKPs) that prove the correct execution of HE evaluations while protecting the server's private inputs. More precisely, we first design two new highly-efficient ZKPs for modulo operations and (Inverse) Number Theoretic Transforms (NTTs), two of the basic operations of HE evaluations. Then we build a customized ZKP for HE evaluations, which is scalable, enjoys a fast prover time and has a non-interactive online phase. Our ZKP is applicable to all Ring-LWE based HE schemes, such as BGV and CKKS. Finally, we implement our protocols for both BGV and CKKS and conduct extensive experiments on various HE workloads. Compared to the state-of-the-art works, both of our prover time and verifier time are improved; especially, our prover cost is only roughly 27-36$\times$ more expensive than the underlying HE operations, this is two to three orders of magnitude cheaper than state-of-the-arts.

*This is the full version of the paper published in IEEE S&P 2025. In this version, we fix a bug that appeared in Section 4 of the proceeding version. As a result, the numbers in the abstract no longer hold. We estimate that by applying the fix, the prover's and the verifier's overhead will increase severalfold.

Contents

1 Introduction **1**
- 1.1 Our Contributions . . . 1
- 1.2 Related Works . . . 3

2 Preliminaries **3**
- 2.1 Notions . . . 3
- 2.2 Security Model . . . 4
- 2.3 Homomorphic Encryption . . . 4
- 2.4 RNS Representation and NTT Form . . . 5
- 2.5 Subfield Vector Oblivious Linear Evaluation . . . 6
- 2.6 Zero-Knowledge Proofs based on sVOLE . . . 7

3 Technical Overview **7**
- 3.1 For Modulo Operation . . . 8
- 3.2 For (I)NTT Operation . . . 8
- 3.3 For HE Evaluations . . . 9

4 VOLE-ZKP for Modulo Operation **9**
- 4.1 Preprocessing Phase . . . 9
- 4.2 Main Protocol . . . 9

5 VOLE-ZKP for (I)NTT Operation **10**

6 VOLE-ZKP for HE Evaluations **11**
- 6.1 Admissible HE Evaluations . . . 11
- 6.2 Instantiations . . . 12
  - 6.2.1 VOLE-ZKP for modulus-switching of CKKS . . . 12
- 6.3 Putting Things Together . . . 12

7 Implementation and Benchmark[1] **12**
- 7.1 Performance Evaluation . . . 14
  - 7.1.1 Evaluation of Online Protocol . . . 14
  - 7.1.2 Evaluation of Preprocessing Protocol . . . 14
- 7.2 Comparison with Other Works . . . 14
  - 7.2.1 Qualitative Comparison . . . 14
  - 7.2.2 Quantitative Comparison . . . 15
- 7.3 Scalability Analysis . . . 15

8 Discussions **16**

9 Conclusion **17**

A Security Proofs **20**
- A.1 Proof of Theorem 1 . . . 20
- A.2 Proof of Theorem 2 . . . 20

B VOLE-ZKPs for More HE Evaluation **21**
- B.1 For Key-Switching . . . 21
- B.2 For Ciphertext-Ciphertext Multiplication . . . 21
- B.3 For Rotation . . . 21

C Discussion on Proving Bootstrapping of CKKS **21**

[1] This section keeps aligned with the proceeding version. We will update this section after completely implementing the fix proposed in Section 4. We estimate that by applying the fix, the prover's and the verifier's overhead will increase severalfold.

1 Introduction

Homomorphic Encryption (HE) enables computations on encrypted data without decryption, and has been extensively studied since Gentry's breakthrough work [43]. Among the various HE schemes, Ring-LWE (RLWE) based HE schemes such as BFV [33], BGV [15] and CKKS [24] support Single Instruction Multiple Data (SIMD) [67], thus they are suitable for larger workloads and have been widely adopted in many popular application scenarios, examples include privacy-preserving machine learning [16, 55], private set intersection [27, 58] and private genome-wide association studies [54].

In most of the works using HE, there are two participants, whom we refer to as client and server. The server is asked to perform evaluations on client's encrypted data. Usually, the authors assume a threat model that only considers semi-honest (a.k.a, passive, or honest-but-curious) security, where the server will honestly follow what was asked to do. However, if the server is *malicious*, it might evaluate arbitrary functions instead of the designated functions on client's input. Indeed, the client has few countermeasures if he doubts the correctness of the HE computation. What's worse, such correctness issues may further lead to privacy violations. Take symmetric private set intersection for example, where client and server jointly compute a set intersection using their private sets, and both of them will obtain the result without knowing the other party's input. Without verification, a malicious server might homomorphically add some client's input to some less significant bits of the output in advance, allowing it to learn extra information.

Verifiable HE. To address above problem, *verifiable* HE (vHE) is proposed, which allows the client to verify if the server has correctly executed the given computation task. Some research works on vHEs (e.g. [13, 36, 37, 40, 41, 46, 51]) focus on the theoretical side and offer no implementations. For the recent works focusing on building practical vHEs, we have identified the following issues.

Issue I: Unaffordable server's overhead. A line of works employ Zero-Knowledge Proof (ZKP) [47] to build vHE schemes. More precisely, once the server receives the ciphertext sent by the client, the server can use the received ciphertext (and possibly its private inputs) as the witness to prove that he has performed the correct computation using ZKPs. Since the server's input are protected via ZKPs, it is easy to see that this type of vHEs can be used to construct secure two-party computation protocols. However, the current state-of-the-art (SOTA) vHE works in this line ([5, 68]) express the HE evaluations as circuits, and prove these circuits using the existing ZK succinct non-interactive argument of knowledge (zkSNARK) schemes, which places a huge burden on server. We evaluate their open-sourced codes, and it turns out that their prover cost is more than four orders of magnitude higher than the HE computation cost. Given that the HE computation is already orders of magnitude more expensive than plaintext computation, such vHE schemes would not be affordable for practical usage.

Issue II: Lack of server's privacy. Some other works focus on the verifiable computation scenario where all the inputs are belong to the client, so that the "zero-knowledge" property is not required. In [20], Chatel et al. designed new error-detecting plaintext encoding schemes to construct vHE. By their new encoding schemes, the client can check if the server has correctly executed the computation by checking the encoding of the decrypted results. This method is very fast: it only introduces about 3-11$\times$ overhead to the server's computation time when used for genomic-data analysis. However, in practice it is quite common that the server also has some private inputs, such as private neural network inference [55] (the server holds the model, and the client holds the data), or private set intersection [27] (each party holds a part of dataset). Verifiable computation does not help in those use cases.

Research question. The above discussions motivate our main research question:

> *Can we construct a ZKP with (relatively) fast prover time for HE evaluations?*

We believe that if above research question could be answered, it would be a significant advancement towards truly practical and useful vHE schemes.

1.1 Our Contributions

We provide an affirmative answer to our research question. We have thoroughly analyzed the evaluation operations of modern RLWE-based HE schemes, and find that there are four basic operations being widely used: (i) additions, (ii) multiplications, (iii) modulo operation and (iv) (Inverse) Number-Theoretic Transforms (NTTs) (to be defined in Section 2.4). Previous works first express these operations as circuits, then prove these circuits using the existing zkSNARKs, introducing unaffordable overhead. In this work, we choose to follow the paradigm of ZKP that is based on Vector Oblivious Linear Evaluations (VOLE) [14] (hereafter, we call it VOLE-ZKP). Additions can be processed for free in VOLE-ZKP, and multiplications can be proved by a prior work called QuickSilver [75].

For modulo and (I)NTT operations, we provide two novel highly efficient VOLE-ZKPs; we prove these operations by checking the consistency of the inputs/outputs directly, avoiding the expensive circuit expressions. Given these two VOLE-ZKPs as main building blocks, we are able to build a customized VOLE-ZKP for HE evaluations, including modulus-switching, key-switching, etc. See the overview in Figure 1.

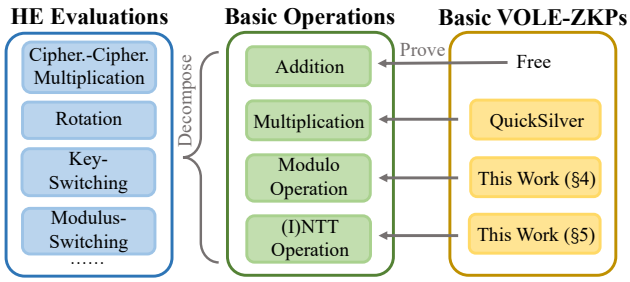


Figure 1: Overview of our VOLE-ZKP for HE evaluations.

The first VOLE-ZKP for modulo operation. We build *the first* VOLE-ZKP for modulo operation; for example, given committed $x_1 \in \mathbb{F}_{p_1}$ and $x_2 \in \mathbb{F}_{p_2}$ where p_1, p_2 are primes, a prover wishes to prove that $x_2 = x_1 \mod p_2$. This new VOLE-ZKP not only serves as one main building block of the VOLE-ZKP for HE evaluations, but also has other applications where modulo operations are needed.

VOLE-ZKP for (I)NTT operation with $O(N)$ time. Both NTT operation and its inverse (i.e., INTT operation) require $O(N \log N)$ time, and they can be viewed as linear transformations. Since VOLE-ZKP are friendly to prove linear transformation, one can directly obtain a VOLE-ZKP for (I)NTT with $O(N \log N)$ time. In this work, we provide a novel VOLE-ZKP for (I)NTT operation with only $O(N)$ time, which is crucial for proving HE evaluations.

Customized VOLE-ZKP for HE evaluations. Given our VOLE-ZKPs for modulo and (I)NTT operations as the main building blocks, we present $\mathcal{Z}$HE: a customized VOLE-ZKP for HE evaluations, whose online phase can be made *non-interactive*. In this work, we take BGV [15] and CKKS scheme [24] as examples but $\mathcal{Z}$HE is also applicable to other RLWE-based HEs, such as BFV [33].

Implementation and benchmark. We implement our VOLE-ZKPs for two HE schemes (i.e., CKKS and BGV) based on Microsoft SEAL library [1]. To illustrate the practical efficiency and the scalability of our protocols, we conduct experiments under 4 scenarios of different scales: (i) ciphertext-ciphertext multiplication, (ii) matrix-vector multiplication, (iii) logistic regression training approximation and (4) comparison approximation (in the latter 2 scenarios, we only consider our protocols for CKKS). It turns out that under all of above scenarios, our prover's overhead[2] is around $30\times$ for CKKS and is around $32\times$ for BGV.

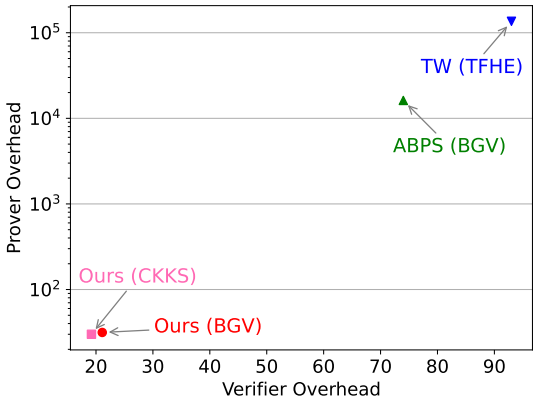


Figure 2: Comparison of prover's/verifier's overhead during the online phase among our work and SOTA works.

[2]Prover's overhead $= \frac{\text{Prover time}}{\text{HE computation time}} - 1$, so as verifier's overhead.

We compare our protocols with two SOTA works: the work by Atapoor et al. [5] (hereafter, we call it ABPS protocol) which implements a ZKP for BGV, and the work by Thibault and Walter [68] (hereafter, we call it TW protocol) which implements a ZKP for TFHE [26]. Since these ZKPs are designed for different HE schemes, we use prover's/verifier's overhead as the performance metrics for comparison, and we put the comparison results in Figure 2. The numbers of our work are obtained under the scenario of matrix-vector multiplication; whereas the scenarios of ABPS and TW are described in Section 7.2. As shown in Figure 2, we make improvements in both prover's and verifier's overhead; particularly, we can reduce prover's overhead by roughly 2-3 orders of magnitude.

1.2 Related Works

Verifiable HE. The notion of vHE can be traced back to the work by Gennero, Gentry and Parno [41]. Later, several works (e.g., [13,36,37,40,46,51]) continue the study of [41]; however, these works only provide feasibility and do not offer any implementation. In the following, we focus on recent works that aim for practical efficiency.

Very recently, in [20], Chatel et al. proposed two new plaintext encoding schemes to construct vHE. Their newly proposed encoding schemes are error-detecting: if the server acts maliciously, the client can detect it by checking the encoding of the returned results. Although this method is very fast, it cannot support the case where the server has private inputs; thus, its applications are limited.

Another line of vHE works use ZKP as the main component, and these vHE systems can support the case where server has private inputs. Most of these works express the HE evaluations as circuits, and use the existing ZKPs to prove these circuits. In [69], Viand et al. instantiate their protocols for BGV using a variety of ZKPs; however, their implemented vHE schemes cannot support key-switching, which is a crucial operation of BGV scheme. Later, this work is improved by ABPS protocol [5]; ABPS protocol utilizes a latticed-based zk-SNARK [52], and it achieves 4-6$\times$ improvement in prover time and can prove the key-switching operation. Concurrently, TW protocol is proposed [68] to support TFHE. More precisely, TW protocol uses a zkSNARK called plonky2 [63] to prove a full bootstrapping operation of TFHE. Note that, both ABPS and TW protocols require $O(N \log N)$ prover time, where N is the circuit size. In contrast, the ZK protocols introduced in this work only require $O(N)$ prover time.

Above ZKP-based vHEs focus on verifying operations on the ciphertext space; in contrast, Aranha et al. proposed a protocol [4] that verifies the operations on the plaintext space. However, their protocol cannot support the CKKS, since CKKS is designed for approximate arithmetic. Furthermore, [4] only offered a toy implementation, which is not for a full HE application scenario; thus, it is hard to estimate their actual efficiency.

VOLE-ZKP. Since the original work by Boyle et al. [14], VOLE-ZKPs received a lot of attention due to its advantages, including fast prover, small memory and post-quantum security; more details could be found in the excellent SoK paper of Baum et al. [10].

Most of prior VOLE-ZKPs (e.g., [30,31,72,74,75]) focus on circuit satisfiability, and they mainly aim to provide highly-efficient VOLE-ZKPs for multiplication gates. VOLE-ZKPs are also used in some real-world applications, e.g., privacy-preserving machine learning [59,73]. More concretely, they designed VOLE-ZKPs for arithmetic-boolean conversion, range proof, etc. To the best of our knowledge, we are the first to construct VOLE-ZKP for HE evaluations.

2 Preliminaries

2.1 Notions

We let λ be the computational security parameter. We use the abbreviation PPT to denote probabilistic polynomial-time. We denote by $x \leftarrow S$ the event where x is uniformly sampled from a set S. For $a, b \in \mathbb{N}$, we use $[a, b]$ to denote by $\{a, a+1, \ldots, b\}$. For $a \in \mathbb{N}$, we use $[a]$ to denote by $\{1, 2, \ldots, a\}$. We write $\lfloor x \rceil$ for rounding x to the nearest integer. We use bold lower-case letters (say, $\boldsymbol{x}$) to denote vectors, and we denote by x_i the i-th component of $\boldsymbol{x}$. We use $\mathsf{negl}(\cdot)$ to denote the negligible function. For two sets $\mathcal{X}, \mathcal{Y}$, we use $\mathcal{X} \overset{c}{\approx} \mathcal{Y}$ (resp. $\mathcal{X} \equiv \mathcal{Y}$) to denote that $\mathcal{X}$ and $\mathcal{Y}$ are computationally (resp. perfectly) indistinguishable.

For an extension field $\mathbb{F}_{p^r}$ of a subfield $\mathbb{F}_p$, where $p \geq 2$ is a prime and $r \geq 1$ is an integer, we fix some monic, irreducible polynomial $f(X)$ of degree r and use the representation $\mathbb{F}_{p^r} \cong \mathbb{F}_p[X]/f(X)$. Every $w \in \mathbb{F}_{p^r}$ can be uniquely written as $w = \sum_{i \in [r]} w_i X^{i-1}$; thus, we can view every element over $\mathbb{F}_{p^r}$ as a vector from $\mathbb{F}_p^r$.

2.2 Security Model

In this work, we consider a client-server setting: a semi-honest client with a private input or a malicious server (either with or without a private input).

UC framework. All of our protocols are proven in the Canetti's Universal Composability (UC) framework [17]. In the following, we provide a brief introduction about UC framework, and we refer readers to see more details in [17].

The UC framework is based on the ideal/real world paradigm [45]. In the ideal world, the parties will send their inputs to an trusted entity called ideal functionality $\mathcal{F}$, who will honestly perform the computation task and send to each party its respective output. The corrupted parties are controlled by an ideal-world adversary S (a.k.a., the simulator). In the real-world, parties communicate with each other to execute the protocol Π. The corrupted parties in the real-world are controlled by the real-world adversary A. There is an additional entity called environment Z, which delivers the inputs to parties and receives the results outputted by those parties. The environment Z can communicate with A (resp. S) and corrupt the parties through the adversary in the real (resp. ideal) world. In UC framework, the security of a protocol is argued by comparing the ideal world execution to the real world execution. More concretely, for any PPT A who is attacking an execution of Π, there exists a PPT S attacking the ideal process that interacts with $\mathcal{F}$ (by corrupting the same set of parties), such that the executions of Π with A is computationally indistinguishable from that of $\mathcal{F}$ with S to Z. We denote by $\mathsf{EXEC}_{\mathcal{F},\mathsf{S},\mathsf{Z}}$ (resp. $\mathsf{EXEC}_{\Pi,\mathsf{A},\mathsf{Z}}$) the output of Z in the ideal world (resp. real world) execution. Formally, we have the following definition.

Definition 1. *We say a protocol* Π UC-realizes *the functionality* $\mathcal{F}$*, if for any PPT* Z *and any PPT* A*, there exists a PPT* S *s.t.* $\mathsf{EXEC}_{\Pi,\mathsf{A},\mathsf{Z}} \overset{c}{\approx} \mathsf{EXEC}_{\mathcal{F},\mathsf{S},\mathsf{Z}}$.

We then introduce the hybrid world. We say a protocol Π is in the $\mathcal{G}$-hybrid world if Π invokes an ideal functionality $\mathcal{G}$ as the subroutine. We note that, $\mathcal{G}$ can be treated as a separate entity and its security can be analyzed separately, which makes the protocol design and security analysis simpler. Formally, we have the following definition.

Definition 2. *We say a protocol* Π UC-realizes *the functionality* $\mathcal{F}$ *in the* $\mathcal{G}$*-hybrid world, if for any PPT* Z *and any PPT* A*, there exists a PPT* S *s.t.* $\mathsf{EXEC}^{\mathcal{G}}_{\Pi,\mathsf{A},\mathsf{Z}} \overset{c}{\approx} \mathsf{EXEC}_{\mathcal{F},\mathsf{S},\mathsf{Z}}$.

Notice that, our proposed UC-secure ZKP protocols offer great security guarantees. For example, while normal ZKPs only achieve soundness, UC-secure ZKPs can achieve straight-line knowledge extractability.

2.3 Homomorphic Encryption

Our ZKP is applicable to all RLWE-based HEs including BFV [33], BGV [15] and CKKS [24]. Here we take CKKS as an example to describe HE. CKKS scheme operates over a ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ where N is a power-of-two integer, and we write $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$ for the residue ring of $\mathcal{R}$ modulo an integer Q. Notice that, a CKKS plaintext $m \in \mathcal{R}$ encodes a vector $\boldsymbol{a} \in \mathbb{C}^{\frac{N}{2}}$, and m can be decoded into $\boldsymbol{a}$. For simplicity, we omit the encoding/decoding algorithms here (ref. [24]), and describe the rest algorithms as follows.

- $\mathsf{ParamGen}(1^\lambda, q, L, \eta)$: Given the security parameter λ, a base module q, a maximum level L of computation, the precision η, the algorithm generates the parameters:
  - Choose $L + 2$ primes $\mathcal{D} := \{p_0, q_0, q_1, \ldots, q_L\}$. We write $Q_\ell := \prod_{i=0}^{\ell} q_i$ for $\ell \in [0, L]$.
  - Choose a power-of-two integer N.
  - Choose small distributions for secret key/encryption key/error $\chi_{key}/\chi_{enc}/\chi_{err}$ over $\mathcal{R}$ respectively.

  Finally, Output $\mathsf{params} := (\mathcal{D}, N, \chi_{key}, \chi_{enc}, \chi_{err})$ which is a default input to the following algorithms.
- $\mathsf{KSGen}(s_1, s_2)$: Given $s_1, s_2 \in \mathcal{R}$, sample $\tilde{a}_i \leftarrow \mathcal{R}_{p_0 Q_L}$ and $\tilde{e}_i \leftarrow \chi_{err}$ for each $i \in [0, L]$. Output switching keys $\mathsf{swk} = \{\mathsf{swk}_i = (\tilde{b}_i, \tilde{a}_i)\}_{i \in [0,L]}$, where $\tilde{b}_i := -\tilde{a}_i \cdot s_2 + \tilde{e}_i + p_0 \cdot B_i \cdot s_1 \in \mathbb{Z}_{p_0 Q_L}$ for integer $B_i \in \mathbb{Z}_{Q_L}$ s.t. $B_i = 1 \mod q_i$ and $B_i = 0 \mod q_j$ for all $j \neq i$.
- $\mathsf{KeyGen}(1^\lambda)$: Given the security parameter λ, the algorithm samples the following keys:
  - Sample $s \leftarrow \chi_{key}$, and set secret key as $\mathsf{sk} := (1, s)$.

- Sample $a \leftarrow \mathcal{R}_{Q_L}$ and $e \leftarrow \chi_{err}$, and set the public key as $\mathsf{pk} := (b, a)$, where $b := -a \cdot s + e \mod Q_L$.
- Compute the relinearization key $\mathsf{rlk} \leftarrow \mathsf{KSGen}(s^2, s)$.
- Compute the rotation key $\mathsf{rk}^{(k)} \leftarrow \mathsf{KSGen}(s^{(k)}, s)$ for each rotation step $k \in [1, \frac{N}{2} - 1]$, where $s^{(k)}$ is the rotated secret with respect to s.

Finally, output $\mathsf{pk}, \mathsf{sk}, \mathsf{rlk}, \mathsf{rk} := \{\mathsf{rk}^{(k)}\}_{k \in [1, \frac{N}{2}-1]}$.

- $\mathsf{Enc}(\mathsf{pk}, m)$: Given the public key pk and the plaintext $m \in \mathcal{R}_{Q_0}$, sample $\nu \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$. Output the ciphertext $\mathsf{ct} := \nu \cdot \mathsf{pk} + (m + e_0, e_1) \in \mathcal{R}^2_{Q_L}$. We will abuse the notion of encryption so that if $\boldsymbol{v} \in \mathbb{C}^{\frac{N}{2}}$ is encoded by m then we will also say ct encrypts $\boldsymbol{v}$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: Given secret key sk and a ciphertext $\mathsf{ct} \in \mathcal{R}^2_{Q_\ell}$, output the plaintext $m = \langle \mathsf{ct}, \mathsf{sk} \rangle \mod Q_0 \in \mathcal{R}_{Q_0}$.
- $\mathsf{Add}(\mathsf{ct}, \mathsf{ct}')$: Given two ciphertext $\mathsf{ct}, \mathsf{ct}' \in \mathcal{R}^2_{Q_\ell}$, output the ciphertext $\mathsf{ct}_{\mathsf{Add}} := \mathsf{ct} + \mathsf{ct}' \in \mathcal{R}^2_{Q_\ell}$.
- $\mathsf{ModSwitch}(\mathsf{ct}, Q_\ell, Q_{\ell-1})$: Given the ciphertext $\mathsf{ct} \in \mathcal{R}^2_{Q_\ell}$ and two modulus $Q_\ell, Q_{\ell-1}$, output the ciphertext $\mathsf{ct}' := \lfloor (Q_\ell / Q_{\ell-1})^{-1} \cdot \mathsf{ct} \rceil \mod Q_{\ell-1}$.
- $\mathsf{KeySwitch}(\mathsf{swk}, \mathsf{ct})$[3] Given the switching key $\mathsf{swk} = \{\mathsf{swk}_i\}_{i \in [0,L]}$ and a ciphertext $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}^2_{Q_\ell}$, the algorithm performs the following:
  - Let $c_{1,i} = c_1 \mod q_i$ for $i \in [0, \ell]$.
  - Compute $\tilde{\mathsf{ct}} := \sum_{i=0}^{\ell} c_{1,i} \cdot \mathsf{swk}_i \mod p_0 Q_\ell$.
  - Output $\mathsf{ct}' := (c_0, 0) + \mathsf{ModSwitch}(\mathsf{ct}, p_0 Q_\ell, Q_\ell)$.
- $\mathsf{Mult}(\mathsf{rlk}, \mathsf{ct}, \mathsf{ct}')$: Given the relinearization key rlk and two ciphertexts $\mathsf{ct} = (c_0, c_1), \mathsf{ct}' = (c'_0, c'_1) \in \mathcal{R}^2_{Q_\ell}$, the algorithm performs the following:
  - Compute $d_0 := c_0 c'_0, d_1 := c_0 c'_1 + c'_0 c_1, d_2 := c_1 c'_1 \mod Q_\ell$.
  - Invoke $\tilde{ct}' := \mathsf{KeySwitch}(\mathsf{rlk}, (0, d_2))$.
  - Output $\mathsf{ct}_{\mathsf{Mult}} := (d_0, d_1) + \tilde{ct}' \mod Q_\ell$.
- $\mathsf{Rotate}(\mathsf{rk}^{(k)}, \mathsf{ct})$: Given the rotation key $\mathsf{rk}^{(k)}$ and the ciphertext $\mathsf{ct} \in \mathcal{R}^2_{Q_\ell}$, it performs the following:
  - Rotate the ciphertext $\mathsf{ct} = (b, a)$ and obtain $(b^{(k)}, a^{(k)})$.
  - Output $\mathsf{ct}' := \mathsf{KeySwitch}(\mathsf{rk}^{(k)}, (b^{(k)}, a^{(k)}))$.

Both modulus-switching (ModSwitch) and key-switching (KeySwitch) are crucial, since they can be used to build two important HE evaluations: ciphertext-ciphertext multiplication (Mult) and rotation (Rotate). Given two ciphertexts $\mathsf{ct}_a, \mathsf{ct}_b$ which encrypt $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively, Mult outputs a ciphertext ct_c which encrypts $\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b}$. Given a ciphertext ct which encrypts $\boldsymbol{a} = (a_1, \ldots, a_{N/2})$, Rotate can output a ciphertext ct′ which encrypts a shifted vector $(a_{k+1}, \ldots, a_{N/2}, a_1, \ldots, a_k)$, where k is a parameter.

2.4 RNS Representation and NTT Form

RNS representation. Typically, Q_ℓ in HE is very large, where $Q_\ell = q_0 q_1 \cdots q_\ell$. For efficiency consideration, in practice, Residue Number System (RNS) is used [23]. By Chinese Reminder Theorem (CRT), we have

$$\mathcal{R}_{Q_\ell} = \mathbb{Z}_{Q_\ell}[X]/(X^N + 1) \cong \prod_{i=0}^{\ell} \mathbb{Z}_{q_i}[X]/(X^N + 1) \ .$$

Therefore, given an element $a \in \mathcal{R}_{Q_\ell}$, we can obtain $\ell + 1$ polynomials $(a_0(X), a_1(X), \ldots, a_\ell(X)) \in \mathcal{R}_{q_0} \times \mathcal{R}_{q_1} \times \cdots \times \mathcal{R}_{q_\ell}$. If we use a row vector $\mathsf{Coe}(a_i) \in \mathbb{Z}_{q_i}^{1 \times N}$ to denote the coefficients of the polynomial $a_i(X)$, then the ring element $a \in \mathcal{R}_{Q_\ell}$ can be viewed as a matrix $\mathsf{CMat}(a) := (\mathsf{Coe}(a_i))_{i \in [0,\ell]} \in \mathbb{Z}^{(\ell+1) \times N}$.

[3] We consider the key-switching method and the corresponding switching key [23, 54] that are implemented in Microsoft SEAL library [1].

NTT form. One may notice that it is not easy to multiply $a, b \in \mathcal{R}_{Q_\ell}$ when given $\mathsf{CMat}(a), \mathsf{CMat}(b)$. Therefore, in common practice, we often represent the ring element in the Number-Theoretic Transform (NTT) form. More concretely, given an element $a_i(X) \in \mathcal{R}_{q_i}$, we compute the NTT form of $a_i(X)$ as follows:

$$\mathsf{NTT}_i(a_i) := (a_i(\omega_i), a_i(\omega_i^3), \ldots, a_i(\omega_i^{2N-1})) \in \mathbb{Z}_{q_i}^N,$$

where q_i is a prime congruent to 1 modulo $2N$ and $\omega_i \in \mathbb{Z}_{q_i}$ is the $2N$-th root of unity. Put things together, a ring element $a \in \mathcal{R}_Q$ will be stored as a matrix $\mathsf{NMat}(a) := (\mathsf{NTT}_i(a_i))_{i\in[0,\ell]} \in \mathbb{Z}^{(\ell+1)\times N}$.

By using NTT form, we can easily implement the addition and multiplication operations for elements in $\mathcal{R}_{Q_\ell}$ by performing point-wise operations of the corresponding matrices. For example, given $a, b \in \mathcal{R}_{Q_\ell}$, we can easily compute $a \cdot b \in \mathcal{R}_{Q_\ell}$ by computing $\mathsf{NMat}(a) \odot \mathsf{NMat}(b)$.

Conversion. For a ring element $a \in \mathcal{R}_{Q_\ell}$, we call $\mathsf{CMat}(a)$ the coefficient form and call $\mathsf{NMat}(a)$ the NTT form. In some HE evaluation operations, we need to perform the modulo operations on a, and it should be performed on $\mathsf{CMat}(a)$ instead of $\mathsf{NMat}(a)$. To achieve this goal, we need to perform the Inverse NTT (INTT) on $\mathsf{NMat}(a)$ to obtain $\mathsf{CMat}(a)$. We use INTT to denote the INTT operations, which can be viewed as the inverse of NTT.

2.5 Subfield Vector Oblivious Linear Evaluation

Here we introduce the subfield Vector Oblivious Linear Evaluation (sVOLE) (a.k.a, information-theoretic message authentication codes [12, 61]), which works over a subfield $\mathbb{F}_p$ with a prime order p and its extension field $\mathbb{F}_{p^r}$. In sVOLE, there is a prover P and a verifier V. For a vector $\boldsymbol{x} \in \mathbb{F}_p^\ell$ held by P, we let V holds the randomly sampled MAC keys $\Delta \in \mathbb{F}_{p^r}$ and $\boldsymbol{k} \in \mathbb{F}_{p^r}^\ell$, and we let P hold the corresponding MAC tag $\boldsymbol{m} \in \mathbb{F}_{p^r}^\ell$ s.t. $\boldsymbol{m} = \boldsymbol{k} - \Delta\boldsymbol{x}$. In different instances, Δ can be reused while $\boldsymbol{k}$ cannot, so we call Δ the global MAC key and call $\boldsymbol{k}$ the local MAC key. We use $[\![\boldsymbol{x}]\!]$ to denote such an authentication. The above notions also hold for a value $x \in \mathbb{F}_p$, which we write as $[\![x]\!]$. The sVOLE correlations have the following properties:

- *Hiding.* V cannot learn anything about $\boldsymbol{x}$ from $\Delta, \boldsymbol{k}$, since $\Delta, \boldsymbol{k}$ are independent of $\boldsymbol{x}$.
- *Binding.* P cannot open $[\![\boldsymbol{x}]\!]$ to another $\boldsymbol{x}' \neq \boldsymbol{x}$ except for a probability at most p^{-r}, since P cannot learn Δ.
- *Additive Homomorphism.* Given $[\![\boldsymbol{x}_1]\!], \ldots, [\![\boldsymbol{x}_n]\!]$ and public coefficients $a_1, \ldots, a_n$ and $\boldsymbol{b}$, P and V can locally compute $[\![\boldsymbol{y}]\!]$ such that $\boldsymbol{y} = \boldsymbol{b} + \sum_{i=1}^n a_i \cdot \boldsymbol{x}_i$.

Due to above properties, sVOLE are widely used for authentication, and we put the formal description in Figure 3.

Procedure $\mathsf{Auth}(\boldsymbol{x}, \Delta, p, r)$

On input $\boldsymbol{x} \in \mathbb{F}_p^\ell$, $\Delta \in \mathbb{F}_{p^r}$ and p, r, this procedure interacts with a prover P, a verifier V and an adversary S, and this procedure will generate $[\![\boldsymbol{x}]\!]$ as follows:

- If V is honest, sample $\boldsymbol{k} \leftarrow \mathbb{F}_{p^r}^\ell$; else, receive $\boldsymbol{k} \in \mathbb{F}_{p^r}^\ell$ from the adversary S.
- If P is honest, compute $\boldsymbol{m} := \boldsymbol{k} - \boldsymbol{x}\Delta \in \mathbb{F}_{p^r}^\ell$. Else, receive $\boldsymbol{m} \in \mathbb{F}_{p^r}^\ell$ from S and recompute $\boldsymbol{k} := \boldsymbol{m} + \boldsymbol{x}\Delta \in \mathbb{F}_{p^r}^\ell$.
- Return $(\boldsymbol{x}, \boldsymbol{m})$ to P and $\boldsymbol{k}$ to V.

Figure 3: Procedure $\mathsf{Auth}(\boldsymbol{x}, p, r, \Delta)$

Authentications over different fields. In this work, we will take CKKS scheme as an example to illustrate our proposals; so, we will deal with $L+2$ prime-order fields $\mathbb{F}_{p_0}, \{\mathbb{F}_{q_i}\}_{i\in[0,L]}$, which are the underlying fields of CKKS as discussed in Section 2.3, and their extension fields. Additionally, we also need to deal with the binary field $\mathbb{F}_2$ and its extension field $\mathbb{F}_{2^\lambda}$.

For simplicity, from now on, we denote $q_{L+1} := p_0$. We pick degrees $\{r_i\}_{i\in[0,L+1]}$ s.t. $q_i^{-r_i} = \mathsf{negl}(\lambda)$ for each $i \in [0, L+1]$. When the underlying field needs to be clarified, we use $[\![\cdot]\!]_i$ to denote the authenticated values over the field $\mathbb{F}_{p_i}$ and its extension field $\mathbb{F}_{p_i^{r_i}}$ for each $i \in [0, L+1]$, and we use $[\![\cdot]\!]_\mathsf{B}$ to denote the authenticated values over $\mathbb{F}_2$ and its extension field $\mathbb{F}_{2^r}$.

Authenticated HE ciphertext. We use sVOLE correlations to authenticate HE ciphertext. Notice that, as discussed in 2.4, the HE ciphertexts are often stored in RNS representations, i.e., for a ciphertext $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_{Q_\ell}^2$, we will use two matrices (in either coefficient or NTT form) in $\mathbb{Z}^{(\ell+1)\times N}$ to represent ct. We here take the NTT form (i.e.,

$\mathsf{NMat}(\mathsf{ct}) = (\mathsf{NMat}(c_0), \mathsf{NMat}(c_1)))$ for example to show how we authenticate HE ciphertext: for each $i \in [0, \ell]$ and $b \in \{0, 1\}$, we let $\boldsymbol{v}^{(b,i)} \in \mathbb{F}_{q_i}^N$ be the i-th row of $\mathsf{NMat}(c_b)$, then we execute procedure Auth to obtain $[\![\boldsymbol{v}^{(b,i)}]\!]_i$. For convenience, we denote by $\langle\!\langle \mathsf{NMat}(\mathsf{ct}) \rangle\!\rangle := (\langle\!\langle \mathsf{NMat}(c_0) \rangle\!\rangle, \langle\!\langle \mathsf{NMat}(c_1) \rangle\!\rangle)$ the authenticated HE ciphertexts, and they can be obtained by setting the i-th row of $\langle\!\langle \mathsf{NMat}(c_b) \rangle\!\rangle$ as $[\![\boldsymbol{v}^{(b,i)}]\!]_i$ for each $i \in [0, \ell], b \in \{0, 1\}$.

2.6 Zero-Knowledge Proofs based on sVOLE

Recently, there are a line of beautiful works building highly efficient ZKPs based on sVOLE correlations [9, 11, 30, 31, 57, 72–74, 76], and we call them VOLE-ZKPs. These works follow the "commit-and-prove" paradigm: the prover P first commits the witness $\boldsymbol{w}$ to the verifier V using sVOLE, then P proves that the authenticated witness $[\![\boldsymbol{w}]\!]$ satisfies some public relations. Similarly to [73], we here formally define the ZK functionality based on authenticated values, which is put in Figure 4.

Functionality $\mathcal{F}_{\mathsf{authZK}}$

It interacts with a prover P, a verifier V and an adversary S. It is parameterized with $L + 3$ finite prime-order fields $\{\mathbb{F}_{q_i}\}_{i\in[0,L+1]}, \mathbb{F}_2$ and their extension fields. It can invoke procedure Auth in Figure 3.

Initialize: Upon receiving (INIT) from P and V, sample $\{\Delta_i\}_{i\in[0,L+1]} \leftarrow \mathbb{F}_{q_i^{r_i}}$ and $\Gamma \leftarrow \mathbb{F}_{2^r}$ if V is honest, and receive $\{\Delta_i\}_{i\in[0,L+1]} \in \mathbb{F}_{q_i^{r_i}}$ and $\Gamma \in \mathbb{F}_{2^r}$ otherwise. Store $\{\Delta_i\}_{i\in[0,L+1]}, \Gamma$ and send them to V. Ignore any subsequent (INIT) command.

Basic Commands:

Input: Upon receiving $(\text{INPUT}, \boldsymbol{x}, i)$ from P where $i \in [0, L+1] \cup \{\mathsf{B}\}$, if $i \in [0, L+1]$, invoke $\mathsf{Auth}(\boldsymbol{x}, \Delta_i, q_i, r_i)$ so that the parties can obtain $[\![\boldsymbol{x}]\!]_i$; if $i = \mathsf{B}$, invoke $\mathsf{Auth}(\boldsymbol{x}, \Gamma, 2, \lambda)$ so that the parties can obtain $[\![\boldsymbol{x}]\!]_{bi}$.

Output: Upon receiving $(\text{OUTPUT}, [\![\boldsymbol{x}]\!]_i)$ from P and V where $i \in [0, L + 1] \cup \{\mathsf{B}\}$, check if $[\![\boldsymbol{x}]\!]_i$ is valid, and abort if not. Send $\boldsymbol{x}$ to V.

Multiplication: Upon receiving $(\text{MULT}, [\![\boldsymbol{a}]\!]_i, [\![\boldsymbol{b}]\!]_i, [\![\boldsymbol{c}]\!]_i)$ from P and V where $i \in [0, L+1] \cup \{\mathsf{B}\}$, check if $[\![\boldsymbol{a}]\!]_i$, $[\![\boldsymbol{b}]\!]_i$ and $[\![\boldsymbol{c}]\!]_i$ are valid, and abort if not. Check if $\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b} \mod q_i$ holds. If so, send TRUE to V; else, send FALSE to V.

New Commands Introduced in This Work:

Modulo operation: Upon receiving $(\text{MOD}, \mathsf{pos}, \{[\![\boldsymbol{x}^{(i)}]\!]_i\}_{i\in\mathcal{M}})$ from P and V, where $\mathcal{M} \subset [0, L + 1]$ and $\mathsf{pos} \in \mathcal{M}$, check if $\{[\![\boldsymbol{x}^{(i)}]\!]_i\}_{i\in\mathcal{M}}$ are valid, and abort if not. If $\boldsymbol{x}^{(j)} = \boldsymbol{x}^{(\mathsf{pos})} \mod q_j$ for all $j \in \mathcal{M} \setminus \{\mathsf{pos}\}$, send TRUE to V; else, send FALSE to V.

(I)NTT operation: Upon receiving $(\text{NTT}, [\![\boldsymbol{a}]\!]_i, [\![\boldsymbol{b}]\!]_i)$ from P and V, where $i \in [0, L + 1]$, $\boldsymbol{a} \in \mathbb{F}_{q_i}^N$ denotes the coefficient form and $\boldsymbol{b} \in \mathbb{F}_{q_i}^N$ represents the NTT form. Check if $[\![\boldsymbol{a}]\!]_i$ and $[\![\boldsymbol{b}]\!]_i$ valid, and abort if not. Reconstruct the degree-$(N - 1)$ polynomial $a(x)$ over $\mathbb{F}_{q_i}$ from $\boldsymbol{a}$, and check if $\mathsf{NTT}_i(a) = \boldsymbol{b}$ holds. If so, send TRUE to V; else, send FALSE to V.

Figure 4: The Functionality $\mathcal{F}_{\mathsf{authZK}}$

Previous works for circuit satisfiability. Previous VOLE-ZKPs mainly focus on proving circuit satisfiability, e.g. [30, 31, 72, 74, 75]. Since sVOLE correlations have additive homomorphism, it is easy to handle addition gates of the circuit. Therefore, the prior works mainly focus on proving the multiplication gates. In this work, we consider the multiplication protocol in QuickSilver [75], which can UC-realize the MULT command of $\mathcal{F}_{\mathsf{authZK}}$.

This work for HE evaluations. In this work, we aim to design highly-efficient VOLE-ZKP for HE evaluations. As discussed earlier, we have analyzed the HE schemes and find that modulo and (I)NTT operations are crucial in HE evaluations. Therefore, we propose new VOLE-ZKPs for these two operations. Formally, these two functions are captured by the MOD and NTT commands of the functionality $\mathcal{F}_{\mathsf{authZK}}$ respectively. Jumping ahead, in Section 6, we will show how to prove HE evaluations using $\mathcal{F}_{\mathsf{authZK}}$.

3 Technical Overview

In this section, we provide the intuition of our newly proposed VOLE-ZKPs, and leave the full protocols as well as the security proofs to later sections. Before delving into to the details of our technique, we think it is necessary to introduce the "commit-

and-prove" paradigm, which is followed by most of VOLE-ZKPs including this work.

"Commit-and-prove" paradigm. Let f be a function and $[\![\boldsymbol{a}]\!]$ be the input held by both P and V. The prover P wants to convince V that $\boldsymbol{b} = f(\boldsymbol{a})$, and works as follows:

1. *Committing phase.* P computes $\boldsymbol{b} := f(\boldsymbol{a})$ and commits to $\boldsymbol{b}$ to V by using sVOLE correlations (i.e., invoking the procedure Auth in Figure 3), so P and V obtain $[\![\boldsymbol{b}]\!]$.
2. *Proving phase.* P proves that $[\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!]$ satisfies the given relation (i.e., $\boldsymbol{b} := f(\boldsymbol{a})$) by sending a proof to V.

The committing phase is easy to achieve, and the core of above paradigm lies in the proving phase. Therefore, for simplicity, in our VOLE-ZKPs for modulo and (I)NTT operations, we assume the committing phase has been done and we focus on the proving phase.

3.1 For Modulo Operation

Challenge. In VOLE-ZKP for modulo operation, given a subset $\mathcal{M} \subset [0, L+1]$, a set of $\{[\![x^{(i)}]\!]_i\}_{i\in\mathcal{M}}$ and an index $\mathsf{pos} \in \mathcal{M}$, P wishes to convince V that $x^{(i)} = x^{(\mathsf{pos})} \mod q_i$ for each $i \in \mathcal{M}\backslash\{\mathsf{pos}\}$. Considering more than one field is very challenging, and most prior VOLE-ZKPs only consider a fixed field/ring. The most related work is the arithmetic-boolean shares conversion protocol in [73], which considered two fields $\mathbb{F}_p$ and $\mathbb{F}_2$. More concretely, given $[\![x]\!]$ and $\{[\![x_i]\!]_\mathsf{B}\}_{i\in[0,m-1]}$ where $m = \lceil \log p \rceil$, their protocol can be used to prove $\sum_{i\in[0,m-1]} x_i \cdot 2^i \mod p = x$.

We observe that their shares conversion protocol can be used to prove modulo operation. More concretely, for each $j \in \mathcal{M}$, we can invoke their protocols to convert $[\![x^{(j)}]\!]_\mathsf{pos}$ into a set of boolean shares $\{[\![x_i^{(j)}]\!]_\mathsf{B}\}_{i\in[0,m^{(j)}-1]}$ where $m^{(j)} = \lceil \log q_j \rceil$. Then for each $j \in \mathcal{M} \setminus \{\mathsf{pos}\}$, we can let the parties to compute a boolean circuit $([\![z_0^{(j)}]\!]_\mathsf{B}, \ldots, [\![z_{m^{(j)}-1}^{(j)}]\!]_\mathsf{B}) := \mathsf{AdderMod}q_j([\![x_0^{(\mathsf{pos})}]\!]_\mathsf{B}, \ldots, [\![x_{m^{(\mathsf{pos})}-1}^{(\mathsf{pos})}]\!]_\mathsf{B})$ such that $\sum_{k\in[0,m^{(j)}-1]} z_k^{(j)} \cdot 2^k = \sum_{k\in[0,m^{(\mathsf{pos})}-1]} x_k^{(\mathsf{pos})} \cdot 2^k \mod q_j$. Finally, the parties can locally compute $[\![h_i^{(j)}]\!]_\mathsf{B} := [\![z_i^{(j)}]\!]_\mathsf{B} \oplus [\![x_i^{(j)}]\!]_\mathsf{B}$ and let V check if $h^{(j)} = 0$ holds for each $j \in \mathcal{M}\setminus\{\mathsf{pos}\}, i \in [0, m^{(j)}-1]$. If all checks pass, then V will accept the statement. However, this solution is relatively expensive because multiple circuits need to be computed. In this work, we will provide a more efficient protocol.

Our solution. Here we show the intuition behind our protocol. For simplicity, here we assume $\mathcal{M} = \{1, 2\}$ and $\mathsf{pos} = 1$. If ZK property is not required, a simple solution is to directly let P open $[\![x^{(1)}]\!]_1$ and $[\![x^{(2)}]\!]_2$ to V, who checks if $x^{(2)} = x^{(1)} \mod q_2$ holds. To add ZK property, our first attempt is to prepare a pair of correlated randomness $[\![t^{(1)}]\!]_1$ and $[\![t^{(2)}]\!]_2$ such that $t^{(2)} = t^{(1)} \mod q_2$, and use these randomness to mask $[\![x^{(1)}]\!]_1$ and $[\![x^{(2)}]\!]_2$ (i.e., we let P open $[\![u^{(i)}]\!]_i := [\![x^{(i)}]\!]_i + [\![t^{(i)}]\!]_i$ to V for each $i \in \{0, 1\}$). However, since $x^{(1)} + t^{(1)} = u^{(1)} + b \cdot q_1$ where $b \in \{0, 1\}$ and V does not know b, V cannot verify the correctness by checking $u^{(2)} = u^{(1)} + b \cdot q_1 \mod q_2$. To address this issue, we observe that: if we can generate a correct $[\![b]\!]_2$, then we can let P open $[\![u^{(1)}]\!]_1$ and $[\![h]\!]_2 := [\![u^{(2)}]\!]_2 - (u^{(1)} + [\![b]\!]_2 \cdot q_1)$ to V, and V can check the correctness by checking if $h \mod q_2 = 0$ holds.

In order to generate $[\![b]\!]_2$, our key observation is that $b = 1$ if $u^{(1)} < t^{(1)}$ and $b = 0$ otherwise. Therefore, we can let the parties obtain $[\![b]\!]_\mathsf{B} := \mathsf{Comp}(([\![t_0^{(1)}]\!]_\mathsf{B}, ..., [\![t_{m-1}^{(1)}]\!]_\mathsf{B}), (u_0^{(1)}, ..., u_{m-1}^{(1)}))$ by calling the basic commands of $\mathcal{F}_{\mathsf{authZK}}$, where $m = \lceil \log q_1 \rceil$,$(u_0^{(1)}, ..., u_{m-1}^{(1)})$ (resp. $(t_0^{(1)}, ..., t_{m-1}^{(1)})$) is the bit-decomposition of $u^{(1)}$ (resp. $t^{(1)}$) and Comp is a Boolean circuit that outputs $b = 1$ if $\sum_{i\in[0,m-1]} t_i^{(1)} \cdot 2^i > \sum_{i\in[0,m-1]} u_i^{(1)} \cdot 2^i$ and outputs $b = 0$ otherwise. Finally, we show how to convert $[\![b]\!]_\mathsf{B}$ into $[\![b]\!]_2$ efficiently. We let the parties prepare a tuple of authenticated bits $[\![r]\!]_2, [\![r]\!]_\mathsf{B}, [\![(1-2r)^{-1}]\!]_2$, where $r \in \mathbb{F}_2$. Then we let P open $[\![c]\!]_\mathsf{B} := [\![b]\!]_\mathsf{B} \oplus [\![r]\!]_\mathsf{B}$ to V. Since $c = b \oplus r = b + r - 2br \mod q_2$, we can let the parties obtain $[\![b]\!]_2 := (c - [\![r]\!]_2) \cdot [\![(1-2r)^{-1}]\!]_2$ by calling the basic commands of $\mathcal{F}_{\mathsf{authZK}}$.

Now the only thing left is how to generate the correlated randomness $[\![t^{(1)}]\!]_1, \{[\![t_i^{(1)}]\!]_\mathsf{B}\}_{i\in[0,m-1]}, [\![t^{(2)}]\!]_2, [\![r]\!]_2, [\![r]\!]_\mathsf{B}, [\![(1-2r)^{-1}]\!]_2$ such that $t^{(2)} = t^{(1)} \mod q_2$, $r \in \mathbb{F}_2$ and $\{t_i^{(1)}\}_{i\in[0,m-1]}$ is the bit-decomposition of $t^{(1)}$. We observe that these correlated randomness can be generated using the "cut-and-bucketing" technique as in previous work [3,38,73]. Finally, we conjecture that our techniques could be extended to prove modulo operations over ring.

3.2 For (I)NTT Operation

Challenge. In VOLE-ZKP for (I)NTT operation, given $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{b}]\!]$, P wishes to convince V that $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{b}]\!]$ correspond to the coefficient form and NTT form of the same ring element. Since both NTT and INTT are linear transformation, there is a straightforward approach: the parties locally apply NTT on $[\![\boldsymbol{a}]\!]$ to obtain $[\![\boldsymbol{c}]\!]$ where $\boldsymbol{c} = \mathsf{NTT}(a)$, then they locally compute $[\![\boldsymbol{d}]\!] := [\![\boldsymbol{c}]\!] - [\![\boldsymbol{b}]\!]$. Finally, P opens $[\![\boldsymbol{d}]\!]$ to V so V can check if $\boldsymbol{d}$ is a all-zero vector. However, this approach requires $O(N \log N)$ time due to NTT, which is not aligned with our goal of achieving fast prover time.

Our solution. In this work, we provide a novel VOLE-ZKP for (I)NTT operation with only $O(N)$ time. Our key observation is that: a ring element can be viewed as a polynomial f, both $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{b}]\!]$ are merely two different representations of the same polynomial f. Therefore, we can let V sample a random evaluation point s and sends it to P at first. We then let the parties compute $[\![f(s)]\!]$. Notice that, $[\![f(s)]\!]$ can be computed from either coefficient form $[\![\boldsymbol{a}]\!]$ (we denote the computed result as $[\![u]\!]$) and NTT form $[\![\boldsymbol{b}]\!]$ (we denote the computed result as $[\![v]\!]$). If $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{b}]\!]$ indeed correspond to the same polynomial, then $u = v$ holds with overwhelming probability due to Schwartz-Zippel lemma [65,77].

The computation of $[\![u]\!]$ from $[\![\boldsymbol{a}]\!]$ is easy: the parties can locally compute $[\![u]\!] := \sum_{j\in[N]}[\![a_j]\!]\cdot s^{j-1}$, where $[\![a_j]\!]$ is the j-th component of $[\![\boldsymbol{a}]\!]$. The computation of $[\![v]\!]$ from $[\![\boldsymbol{b}]\!]$ requires Lagrange interpolation formula. Note that, naively using Lagrange interpolation formula requires $O(N^2)$ time; we can reduce the time complexity from $O(N^2)$ into $O(N)$ by exploiting the structures of evaluation points of NTT operations. We refer readers to see the time complexity analysis in Section 5.

3.3 For HE Evaluations

In the main body of this paper, we use our VOLE-ZKPs for modulo and (I)NTT operations as main building blocks to construct VOLE-ZKPs for key-switching and modulus-switching. Given these, it easy to construct VOLE-ZKPs for the rest HE evaluations (e.g., ciphertext-ciphertext multiplication, rotation), and we place them in Appendix B. Furthermore, to formally argue the security of our protocols, we introduce the notion of *admissible HE evaluations* and prove that our VOLE-ZKPs can securely realize the admissible HE evaluations. See more details in Section 6.

4 VOLE-ZKP for Modulo Operation

In this section, we provide a VOLE-ZKP that UC-realizes the MOD command of $\mathcal{F}_{\mathsf{authZK}}$ in Figure 4.

4.1 Preprocessing Phase

As discussed in Section 3.1, our protocol requires a preprocessing phase where a tuple of correlated randomness $(\{[\![t^{(i)}]\!]_i\}_{i\in\mathcal{M}}, \{[\![t_i^{(\mathsf{pos})}]\!]_{\mathsf{B}}\}_{i\in[0,m-1]}, [\![r]\!]_{\mathsf{B}}, \{[\![r]\!]_i, [\![s^{(i)}]\!]_i\}_{i\in\mathcal{M}\setminus\{\mathsf{pos}\}})$, where $m = \lceil \log q_{\mathsf{pos}}\rceil$, $\sum_{i\in[0,m-1]} t_i^{(\mathsf{pos})}\cdot 2^i \mod q_{\mathsf{pos}} = t^{(\mathsf{pos})}$, $r\in\mathbb{F}_2$, and $\forall i\in\mathcal{M}\setminus\{\mathsf{pos}\} : t^{(i)} = t^{(\mathsf{pos})} \mod q_i, s^{(i)} = (1-2r)^{-1} \mod q_i$. To formally describe this preprocessing phase, we provide an ideal functionality $\mathcal{F}_{\mathsf{T\text{-}Rand}}$ in Figure 5.

Functionality $\mathcal{F}_{\mathsf{T\text{-}Rand}}$

The functionality interacts with a prover P, a verifier V and an adversary S. It can invoke procedure Auth which is described in Figure 3.

Random: Upon receiving $(\text{RAND}, \{q_i, r_i\}_{i\in\mathcal{M}}, \mathsf{pos})$ from P and $(\text{RAND}, \{q_i, r_i, \Delta_i\}_{i\in\mathcal{M}}, \Gamma)$ from V where $\Gamma\in\mathbb{F}_{2^\lambda}$, q_i is a prime, r_i is a positive integer and $\Delta_i\in\mathbb{F}_{q_i^{r_i}}$ for each $i\in\mathcal{M}$, it works as follows:

- If P is honest, sample $t^{(\mathsf{pos})}\leftarrow\mathbb{F}_{q_{\mathsf{pos}}}$ and $r\leftarrow\mathbb{F}_2$; otherwise, receive $t^{(\mathsf{pos})}\in\mathbb{F}_{q_{\mathsf{pos}}}$ and $r\in\mathbb{F}_2$ from S.
- For each $i\in\mathcal{M}\setminus\{\mathsf{pos}\}$: compute $t^{(i)} := t^{(\mathsf{pos})} \mod q_i$ and $s^{(i)} := (1-2r)^{-1} \mod q_i$.
- Decompose $t^{(\mathsf{pos})}$ into bits $t_0^{(\mathsf{pos})},\ldots,t_{m-1}^{(\mathsf{pos})}$ such that $\sum_{i\in[0,m-1]} t_i^{(\mathsf{pos})}\cdot 2^i \mod q_{\mathsf{pos}} = t^{(\mathsf{pos})}$.
- For each $i\in\mathcal{M}$: invoke $\mathsf{Auth}(t^{(i)}, \Delta_i, q_i, r_i)$ so that the parties obtain $[\![t^{(i)}]\!]_i$.
- For each $i\in[0,m-1]$: invoke $\mathsf{Auth}(t_i^{(\mathsf{pos})}, \Gamma, 2, \lambda)$ so that the parties obtain $[\![t_i^{(\mathsf{pos})}]\!]_{\mathsf{B}}$.
- Invoke $\mathsf{Auth}(r, \Gamma, 2, \lambda)$ so that the parties obtain $[\![r]\!]_{\mathsf{B}}$.
- For each $i\in\mathcal{M}\setminus\{\mathsf{pos}\}$: invoke $\mathsf{Auth}(r, \Delta_i, q_i, r_i)$ and $\mathsf{Auth}(s^{(i)}, \Delta_i, q_i, r_i)$ so that the parties obtain $[\![r]\!]_i$ and $[\![s^{(i)}]\!]_i$.

Figure 5: The Functionality $\mathcal{F}_{\mathsf{T\text{-}Rand}}$

We note that, we can use a standard technique called "cut-and-bucketing" [3,38,73] to realize the functionality $\mathcal{F}_{\mathsf{T\text{-}Rand}}$, and here we omit the detailed protocol construction.

4.2 Main Protocol

The intuition of our protocol is put in Section 3.1, and we provide the formal protocol $\Pi_{\mathsf{ZK\text{-}Mod}}$ in Figure 6. The security of our protocol is argued by Theorem 1 and the proof can be found in Appendix A.1.

Theorem 1. *The protocol $\Pi_{\mathsf{ZK\text{-}Mod}}$ UC-realizes the* MOD *command of $\mathcal{F}_{\mathsf{authZK}}$ in the $\{\mathcal{F}_{\mathsf{T\text{-}Rand}}, \mathcal{F}_{\mathsf{authZK}}\}$-hybrid world (using only basic commands of $\mathcal{F}_{\mathsf{authZK}}$) with statistical error $\max\{(q_i^{-r_i})_{i\in\mathcal{M}}\}$ against a static malicious adversary.*

Optimization. We can use a standard technique called *batch opening* [73] to reduce the communication cost, when there are multiple values need to be opened. We refer readers to see more details about this technique in [73].

Protocol $\Pi_{\mathsf{ZK\text{-}Mod}}$

Input: P and V hold a set $\mathcal{M}$ and $\{[\![x^{(i)}]\!]_i\}_{i\in\mathcal{M}}$ where $x^{(i)} \in \mathbb{F}_{q_i}$ for each $i \in \mathcal{M}$ and a public $\mathsf{pos} \in \mathcal{M}$.

Preprocessing phase: P sends $(\textsc{Extend}, \{q_i, r_i\}_{i\in\mathcal{M}}, \mathsf{pos})$ and V sends $(\textsc{Extend}, \{q_i, r_i, \Delta_i\}_{i\in\mathcal{M}})$ to $\mathcal{F}_{\mathsf{T\text{-}Rand}}$, which returns $(\{[\![t^{(i)}]\!]_i\}_{i\in\mathcal{M}}, \{[\![t_i^{(\mathsf{pos})}]\!]_{\mathsf{B}}\}_{i\in[0,m-1]}, [\![r]\!]_{\mathsf{B}}, \{[\![r]\!]_i, [\![s^{(i)}]\!]_i\}_{i\in\mathcal{M}\setminus\{\mathsf{pos}\}})$ to the parties, where $m = \lceil \log q_{\mathsf{pos}} \rceil$, $\sum_{i\in[0,m-1]} t_i^{(\mathsf{pos})} \cdot 2^i \mod q_{\mathsf{pos}} = t^{(\mathsf{pos})}$, $r \in \mathbb{F}_2$, and $\forall i \in \mathcal{M} \setminus \{\mathsf{pos}\} : t^{(i)} = t^{(\mathsf{pos})} \mod q_i, s^{(i)} = (1-2r)^{-1} \mod q_i$.

Online phase:

1. For each $i \in \mathcal{M}$: P and V locally compute $[\![u^{(i)}]\!]_i := [\![x^{(i)}]\!]_i + [\![t^{(i)}]\!]_i$.
2. P opens $[\![u^{(\mathsf{pos})}]\!]_{\mathsf{pos}}$ to V. Then P commits the output $[\![b]\!]_{\mathsf{B}} := \mathsf{Comp}(([\![t_0^{(\mathsf{pos})}]\!]_{\mathsf{B}}, ..., [\![t_{m-1}^{(\mathsf{pos})}]\!]_{\mathsf{B}}), (u_0^{(\mathsf{pos})}, ..., u_{m-1}^{(\mathsf{pos})}))$ to V, where $(u_0^{(\mathsf{pos})}, ..., u_{m-1}^{(\mathsf{pos})})$ is the bit-decomposition of $u^{(\mathsf{pos})}$ and Comp is a Boolean circuit that outputs $b = 1$ if $\sum_{i\in[0,m-1]} t_i^{(\mathsf{pos})} \cdot 2^i > \sum_{i\in[0,m-1]} u_i^{(\mathsf{pos})} \cdot 2^i$ and outputs $b = 0$ otherwise. The parties invoke the basic commands of $\mathcal{F}_{\mathsf{authZK}}$ to convince V that $[\![b]\!]_{\mathsf{B}}$ is correctly computed.
3. P and V locally compute $[\![c]\!]_{\mathsf{B}} := [\![b]\!]_{\mathsf{B}} \oplus [\![r]\!]_{\mathsf{B}}$. Then P opens $[\![c]\!]_{\mathsf{B}}$ to V.
4. For each $i \in \mathcal{M} \setminus \{\mathsf{pos}\}$:
   (a) P commits the result $[\![b]\!]_i := (c - [\![r]\!]_i) \cdot [\![s^{(i)}]\!]_i$ to V. The parties invoke the MULT command of $\mathcal{F}_{\mathsf{authZK}}$ to convince V that $[\![b]\!]_i$ is correctly computed.
   (b) P and V locally compute $[\![h^{(i)}]\!]_i := [\![u^{(i)}]\!]_i - (u^{(\mathsf{pos})} + [\![b]\!]_i \cdot q^{(\mathsf{pos})})$. Then P opens $[\![h^{(i)}]\!]_i$ to V, who will check if $h^{(i)} \mod q_i = 0$ holds.
5. If all of the above checks pass, V outputs TRUE; else, V outputs FALSE.

Figure 6: Protocol $\Pi_{\mathsf{ZK\text{-}Mod}}$

5 VOLE-ZKP for (I)NTT Operation

We provide a VOLE-ZKP that UC-realizes the NTT command of $\mathcal{F}_{\mathsf{authZK}}$ in Figure 4. The intuition of our protocol is put in Section 3.2, and we provide the formal protocol description $\Pi_{\mathsf{ZK\text{-}NTT}}$ in Figure 7. The security of $\Pi_{\mathsf{ZK\text{-}NTT}}$ is proved by Theorem 2 and the proof can be found in Appendix A.2.

Protocol $\Pi_{\mathsf{ZK\text{-}NTT}}$

Input: P and V hold $[\![\boldsymbol{a}]\!]$, $[\![\boldsymbol{b}]\!]$, where $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_{q_i}^N$.

Preprocessing phase: Let $x_j := \omega_i^{2j-1}$ for each $j \in [N]$, where $\omega_i \in \mathbb{F}_{q_i}$ is the $2N$-th root of unity. P and V locally compute $\ell_j := \Pi_{k\in[N]\setminus\{j\}}(x_j - x_k)$ for each $j \in [N]$.

Online phase:

1. V samples a random $s \in \mathbb{F}_{q_i}$ and sends s to P.
2. P and V locally compute $[\![u]\!] := \sum_{j\in[N]} [\![a_j]\!] \cdot s^{j-1}$, where $[\![a_j]\!]$ is the j-th component of $[\![\boldsymbol{a}]\!]$.
3. P and V locally compute $[\![v]\!] := \sum_{j\in[N]} L_j \cdot [\![b_j]\!]$, where $[\![b_j]\!]$ is the j-th component of $[\![\boldsymbol{b}]\!]$ and $L_j := \ell_j^{-1}\Pi_{k\in[N]\setminus\{j\}}(s - x_k)$ is the j-th Lagrangian coefficient.
4. P and V locally computes $[\![w]\!] := [\![u]\!] - [\![v]\!]$. Then P opens $[\![w]\!]$ to V so that V can obtain w.
5. If $w = 0$, V outputs TRUE; else, V outputs FALSE.

Figure 7: Protocol $\Pi_{\mathsf{ZK\text{-}NTT}}$

Theorem 2. *The protocol $\Pi_{\mathsf{ZK\text{-}NTT}}$ UC realizes the* NTT *command of the functionality $\mathcal{F}_{\mathsf{authZK}}$ in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid world (using only basic commands of $\mathcal{F}_{\mathsf{authZK}}$) with statistical error $\frac{N-1}{q_i}$ against a static malicious adversary.*

Time complexity analysis. The time complexity of protocol $\Pi_{\mathsf{ZK\text{-}NTT}}$ is $O(N)$. We first focus on the online phase. Given s, it requires $O(N)$ time to compute $(s, s^2, \ldots, s^{N-1})$, so the time complexity of computing $[\![u]\!]$ is $O(N)$. To compute the Lagrangian coefficients, we can let the parties to first compute $L_{all} := \Pi_{k\in[N]}(s - x_k)$, which requires $O(N)$ time. Then for each $j \in [N]$, the parties can compute $L_j := \frac{L_{all}}{\ell_j \cdot (s - x_j)}$ in $O(1)$ time; notice that, ℓ_j is computed in the preprocessing phase. Therefore, the time complexity of our online protocol is $O(N)$.

As for the preprocessing phase, we make use of the fact that the evaluation points are structured (i.e., $x_j := \omega_i^{2j-1}$ for $j \in [N]$) to show that the time complexity of our preprocessing protocol is also $O(N)$. We observe that, for each $j \in [1, N-1]$,

we have

$$\begin{aligned}\ell_{j+1} &= (\omega_i^{2j+1} - \omega_i) \cdot \Pi_{k\in[2,N]\setminus\{j+1\}}(\omega_i^{2j+1} - \omega_i^{2k-1}) \\ &= (\omega_i^{2j+1} - \omega_i) \cdot \Pi_{k\in[1,N-1]\setminus\{j\}}\omega_i^2 \cdot (\omega_i^{2j-1} - \omega_i^{2k-1}) \\ &= (\omega_i^{2j+1} - \omega_i) \cdot \omega_i^{2N-4} \cdot \ell_j \cdot (\omega_i^{2j-1} - \omega_i^{2N-1})^{-1} \ .\end{aligned} \tag{1}$$

Given $\{\omega_i^{2j-1}\}_{j\in[N]}$ and ℓ_1 which can both be computed in $O(N)$ time, we can sequentially compute $(\ell_2, \ldots, \ell_N)$ using Equation 1; for each $j \in [2, N]$, ℓ_j can be computed in only $O(1)$ time. Therefore, the time complexity of our preprocessing protocol is $O(N)$. We also note that this cost is *one-time*, and the results $\{\ell_j\}_{j\in[N]}$ obtained from the preprocessing phase can be reused for multiple times.

Towards non-interactive online phase. Notice that, during the online phase of our protocol $\Pi_{\mathsf{ZK\text{-}Mod}}$, V needs to send a randomly sampled $s \in \mathbb{F}_{q_i}$ to P. We can remove this interaction by applying Fiat-Shamir transform [35].

At first glance, it seems impossible to apply Fiat-Shamir transform since there seems to be no common transcript for P and V. However, we stress that the common transcript is "hidden" since we assume both P and V already hold $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{b}]\!]$. In practice, taking $\boldsymbol{a}$ for an example, P and V obtain $[\![\boldsymbol{a}]\!]$ as follows: firstly, P and V prepare random $[\![\boldsymbol{r}]\!]$ via the random sVOLE generation protocols [64,72] in advance; then P can send $\boldsymbol{d} := \boldsymbol{x} - \boldsymbol{r}$ so both parties can obtain $[\![\boldsymbol{a}]\!] := [\![\boldsymbol{r}]\!] + \boldsymbol{d}$. Now we know that $\boldsymbol{d}$ are known by P and V, we can apply Fiat-Shamir transform easily. Notice that, by applying Fiat-Shamir transform, $\Pi_{\mathsf{ZK\text{-}NTT}}$ will be degraded into computational security, and the soundness error will be at most $N_{\mathsf{H}} \cdot (N-1) \cdot q_i^{-1}$ where N_{H} is the number of maximum queries to RO. Notice that, we can use the parallel repetition strategy to achieve negligible soundness error, and this strategy is widely used in other ZKP works, e.g. [19,44,53].

6 VOLE-ZKP for HE Evaluations

6.1 Admissible HE Evaluations

Inspired by [39], we formalize a class of HE evaluations that are amenable to our VOLE-ZKPs and refer to them as *admissible HE evaluations*.

Definition 3 (Admissible HE Evaluation.)**.** *Let* EvalOp *be a HE evaluation operation. We say* EvalOp *is* admissible *if it consists of some combination of a subset or all of the following 4 operations: (1) additions, (2) multiplications, (3) modulo operations and (4) (I)NTT operations.*

In this work, we focus on the ZKPs for admissible HE evaluations. To capture this type of ZKPs, we define an ideal functionality $\mathcal{F}_{\mathsf{adZK}}$ and put it in Figure 8. Informally speaking, given an admissible HE evaluation operation EvalOp and the output $\boldsymbol{o}$, $\mathcal{F}_{\mathsf{adZK}}$ allows the verifier V to check if $\boldsymbol{o}$ is the correct result of $\mathsf{EvalOp}(\boldsymbol{w})$, where $\boldsymbol{w}$ is the witness held by the prover P. We now present our main theorem in the following. Jumping head, this theorem can be used to argue the security of the protocols appeared in Section 6.2.

Functionality $\mathcal{F}_{\mathsf{adZK}}$

The functionality interacts with a prover P, a verifier V and an adversary S. It is parameterized with an admissible HE evaluation operation EvalOp in Definition 3.

Prove: Upon receiving $(\textsc{Prove}, \boldsymbol{w}, \boldsymbol{o})$ from P and $(\textsc{Prove}, \boldsymbol{o})$ from V, compute $\boldsymbol{y} := \mathsf{EvalOp}(\boldsymbol{w})$. Check if $\boldsymbol{o} \stackrel{?}{=} \boldsymbol{y}$. If so, return TRUE to V; otherwise, return FALSE to V.

Figure 8: The Functionality $\mathcal{F}_{\mathsf{adZK}}$

Theorem 3. *For any admissible HE evaluation operation* EvalOp*, there exists a protocol* Π *that UC-realizes* $\mathcal{F}_{\mathsf{adZK}}$ *(which is parameterized with* EvalOp*) in the* $\mathcal{F}_{\mathsf{authZK}}$*-hybrid world against a static malicious adversary.*

Proof. Correctness. Here we show how to construct such a VOLE-ZKP protocol Π using $\mathcal{F}_{\mathsf{authZK}}$. The prover P first commits to witness $\boldsymbol{w}$ by invoking INPUT command of $\mathcal{F}_{\mathsf{authZK}}$, so the parties obtain $[\![\boldsymbol{w}]\!]$. As defined in Definition 3, EvalOp only comprises of the following operations: additions, multiplications, modulo and (I)NTT operations. Additions can be processed for free due to the linear homomorphism of sVOLE correlations. For the rest three operations, P can commit to the computed results by invoking INPUT of $\mathcal{F}_{\mathsf{authZK}}$, then prove the committed results are correct by invoking Mult, MOD and NTT commands of $\mathcal{F}_{\mathsf{authZK}}$. When all basic operations of EvalOp are computed and proven, the parties will hold $[\![\boldsymbol{y}]\!]$. At this time, P can invoke OUTPUT command of $\mathcal{F}_{\mathsf{authZK}}$ to reveal $\boldsymbol{y}$ to V. The verifier V accepts $\boldsymbol{o}$ if and only if $\boldsymbol{o} = \boldsymbol{y}$ holds.

Security. It is easy to see that there are only local computations in Π except for the invocations of $\mathcal{F}_{\mathsf{authZK}}$. Therefore, the security of Π can be reduced to the security of $\mathcal{F}_{\mathsf{authZK}}$, which is guaranteed by [75], Theorem 1 and Theorem 2. □

6.2 Instantiations

We note that, for RLWE-based HEs (including CKKS and BGV), their HE evaluation operations satisfies Definition 3. Therefore, we can directly build VOLE-ZKPs for these HEs' evaluation operations using Theorem 3.

For completeness, here we take CKKS as an example to explicitly show how we construct VOLE-ZKPs for modulus-switching. In Appendix B, we will provide VOLE-ZKPs for the key-switching, ciphertext-ciphertext multiplication and rotation of CKKS.

6.2.1 VOLE-ZKP for modulus-switching of CKKS

We here provide a VOLE-ZKP for modulus-switching of CKKS, i.e., given the witness $\mathsf{ct} \in \mathcal{R}^2_{Q_\ell}$ and the public output $\mathsf{ct}' \in \mathcal{R}^2_{Q_{\ell-1}}$, P wants to convince V that $\mathsf{ct}' = \mathsf{ModSwitch}(\mathsf{ct}, Q_\ell, Q_{\ell-1})$ without leaking anything about ct. Formally, we put the protocol $\Pi_{\mathsf{ZK\text{-}MS}}$ in Figure 9. The security of $\Pi_{\mathsf{ZK\text{-}MS}}$ is guaranteed by Theorem 3.

Notice that, as discussed in Section 2.5, we use $\langle\!\langle \mathsf{NMat}(\mathsf{ct}) \rangle\!\rangle := (\langle\!\langle \mathsf{NMat}(c_0) \rangle\!\rangle, \langle\!\langle \mathsf{NMat}(c_1) \rangle\!\rangle)$ to denote the authenticated ciphertext for a given $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}^2_{Q_\ell}$. We also note that, in Section 2.3, we only provide a simplified description of ModSwitch; in $\Pi_{\mathsf{ZK\text{-}MS}}$, we provide the detailed description of ModSwitch that is implemented in Microsoft SEAL library. For better expression, in $\Pi_{\mathsf{ZK\text{-}MS}}$, we describe ModSwitch step by step and demonstrate how we prove ModSwitch are honestly executed in colored boxes.

6.3 Putting Things Together

Now we show how our VOLE-ZKP protocols can be used to prove the correctness of HE evaluations while supporting server's input privacy. Once the server receives the ciphertext ct sent by the client, the server sets ct and his private input (encoded as the plaintext pt) as the witness $\boldsymbol{w}$. Then the server locally evaluates $\boldsymbol{o} := \mathsf{EvalOp}(\boldsymbol{w})$, where EvalOp is a pre-agreed admissible HE evaluation. Finally, the server can convince the client that $\boldsymbol{o}$ is correctly computed by sending $(\textsc{Prove}, \boldsymbol{w}, \boldsymbol{o})$ to $\mathcal{F}_{\mathsf{adZK}}$.

7 Implementation and Benchmark[4]

We provide a prototype implementation of our VOLE-ZKP protocols (named $\mathcal{Z}$HE) in C++ with single thread, and our codes will be made publicly available. All our experiments are running over a cloud instance equipped with Intel Xeon Platinum 8575 CPU at 2.80GHz and 64 GB of RAM. All implementations achieve 128-bit computational security. For Fiat-Shamir transform, we use SHA256 hash function to instantiate RO.

Table 1: The performance of our online protocol under four scenarios.

HE Scheme	Scenario†	HE Parameter‡ N	HE Parameter‡ L	HE Time (ms)⋆	Prover Time (s)	Verifier Time (s)	Proof Size (MB)
BGV	Mult	8192	2	3.32	0.11	0.07	4.38
	MVM	8192	2	11.20	0.37	0.24	11.46
CKKS	Mult	8192	2	2.93	0.09	0.06	3.88
	MVM	8192	2	19.07	0.60	0.40	18.93
	LRA	16384	6	176.56	4.98	3.25	186.26
	CA	16384	7	555.92	16.28	10.86	628.52

† Mult: ciphertext-ciphertext multiplication; MVM: matrix-vector multiplication; LRA: logistic regression approximation; CA: comparison approximation.
‡ N: the degree of the ring element; L: maximum level of computation.
⋆ Refer to the evaluation time, excluding the key generation time, encoding time, encryption time.

Underlying HE schemes. We implement our protocols for BGV and CKKS. We follow the algorithms that are implemented in Microsoft SEAL library [1].

Metrics. We mainly consider the following metrics: (1) preprocessing time: the time cost of the preprocessing phase; (2) prover/verifier time: the prover's/verifier's computation time during the online phase; (3) proof size: the communication cost of the online phase; (4) overhead: we use HE computation time as baseline (denoted as x_b), and calculate the overhead of x by $(x - x_b)/x_b$, where x can be prover time, verifier time, etc.

[4] This section keeps aligned with the proceeding version. We will update this section after completely implementing the fix proposed in Section 4. We estimate that by applying the fix, the prover's and the verifier's overhead will increase severalfold.

Protocol $\Pi_{\mathsf{ZK\text{-}MS}}$

Input: P and V hold two modulus $Q_\ell, Q_{\ell-1}$ and the public output $\mathsf{NMat}(\mathsf{ct}') := (\mathsf{NMat}(c'_0), \mathsf{NMat}(c'_1))$ where $\mathsf{ct}' \in \mathcal{R}^2_{Q_{\ell-1}}$. P privately holds $\mathsf{NMat}(\mathsf{ct}) := (\mathsf{NMat}(c_0), \mathsf{NMat}(c_1))$ where $\mathsf{ct} \in \mathcal{R}^2_{Q_\ell}$.

Protocol:

(Generating authenticated ciphertexts.) P and V obtain $\langle\!\langle \mathsf{NMat}(\mathsf{ct}) \rangle\!\rangle := (\langle\!\langle \mathsf{NMat}(c_0) \rangle\!\rangle, \langle\!\langle \mathsf{NMat}(c_1) \rangle\!\rangle)$ by invoking INPUT command of $\mathcal{F}_{\mathsf{authZK}}$, as described in Section 2.5.

(Proving ModSwitch *has been executed correctly.)* P performs ModSwitch on $\mathsf{NMat}(\mathsf{ct})$, and P proves that ModSwitch has been performed correctly as follows:

1. For each $b \in \{0,1\}$: Let $\boldsymbol{s}^{(b,\ell)} \in \mathbb{F}_{q_\ell}^N$ be the last row vector of $\mathsf{NMat}(c_b)$, and P performs the INTT operation on $\boldsymbol{s}^{(b,\ell)}$ to obtain $\boldsymbol{t}^{(b)} := \mathsf{INTT}_\ell(\boldsymbol{s}^{(b,\ell)}) \in \mathbb{F}_{q_\ell}^N$.

   P proves above as follows. For each $b \in \{0,1\}$: Let $[\![\boldsymbol{s}^{(b,\ell)}]\!]_\ell$ be the last row vector of $\langle\!\langle \mathsf{NMat}(c_b) \rangle\!\rangle$. P invokes INPUT command of $\mathcal{F}_{\mathsf{authZK}}$ using $\boldsymbol{t}^{(b)}$ so both parties can obtain $[\![\boldsymbol{t}^{(b)}]\!]_\ell$. P and V invoke NTT command of $\mathcal{F}_{\mathsf{authZK}}$ using $[\![\boldsymbol{t}^{(b)}]\!]_\ell$ and $[\![\boldsymbol{s}^{(b,\ell)}]\!]_\ell$ so V can be convinced that $[\![\boldsymbol{t}^{(b)}]\!]_\ell$ is correctly computed.

2. For each $b \in \{0,1\}$: P computes $\boldsymbol{t}^{(b)} := \boldsymbol{t}^{(b)} + (q_\ell - 1)/2 \cdot \mathbf{1}^N \mod q_\ell$, where $\mathbf{1}^N \in \mathbb{F}_{q_\ell}^N$ is a all-one vector.

   For each $b \in \{0,1\}$: P and V locally compute $[\![\boldsymbol{t}^{(b)}]\!]_\ell := [\![\boldsymbol{t}^{(b)}]\!]_\ell + (q_\ell - 1)/2 \cdot \mathbf{1}^N$.

3. For each $b \in \{0,1\}$ and $i \in [0, \ell-1]$: P computes $\boldsymbol{u}^{(b,i)} := \boldsymbol{t}^{(b)} \mod q_i$.

   P proves above as follows. For each $b \in \{0,1\}$: P invokes INPUT command of $\mathcal{F}_{\mathsf{authZK}}$ using $\boldsymbol{u}^{(b,i)}$ so both parties can obtain $[\![\boldsymbol{u}^{(b,i)}]\!]_i$ for all $i \in [0, \ell-1]$. P and V invoke MOD command of $\mathcal{F}_{\mathsf{authZK}}$ using $[\![\boldsymbol{t}^{(b)}]\!]_\ell$ and $\{[\![\boldsymbol{u}^{(b,i)}]\!]_i\}_{i\in[0,\ell-1]}$ so V can be convinced that $\{[\![\boldsymbol{u}^{(b,i)}]\!]_i\}_{i\in[0,\ell-1]}$ are correctly computed.

4. For each $b \in \{0,1\}$ and $i \in [0, \ell-1]$: P computes $\boldsymbol{u}^{(b,i)} := \boldsymbol{u}^{(b,i)} - (q_\ell - 1)/2 \cdot \mathbf{1}^N \mod q_i$.

   For each $b \in \{0,1\}$ and $i \in [0, \ell-1]$: P and V locally compute $[\![\boldsymbol{u}^{(b,i)}]\!]_i := [\![\boldsymbol{u}^{(b,i)}]\!]_i - (q_\ell - 1)/2 \cdot \mathbf{1}^N$.

5. For each $b \in \{0,1\}$ and $i \in [0, \ell-1]$: P computes $\boldsymbol{z}^{(b,i)} := \mathsf{NTT}_i(\boldsymbol{u}^{(b,i)}) \in \mathbb{F}_{q_i}^N$.

   P proves above as follows. For each $b \in \{0,1\}$ and $i \in [0, \ell-1]$: P invokes INPUT command of $\mathcal{F}_{\mathsf{authZK}}$ using $\boldsymbol{z}^{(b,i)}$ so that both parties can obtain $[\![\boldsymbol{z}^{(b,i)}]\!]_i$. P and V invoke NTT command of $\mathcal{F}_{\mathsf{authZK}}$ using $[\![\boldsymbol{u}^{(b,i)}]\!]_i$, $[\![\boldsymbol{z}^{(b,i)}]\!]_i$ so that V can be convinced that $[\![\boldsymbol{z}^{(b,i)}]\!]_i$ are correctly computed.

6. For each $b \in \{0,1\}$ and $i \in [0, \ell-1]$: Let $\boldsymbol{s}^{(b,i)} \in \mathbb{F}_{q_i}^N$ be the i-th row vector of the matrix $\mathsf{NMat}(c_b)$, and P computes $\boldsymbol{z}^{(b,i)} := (\boldsymbol{s}^{(b,i)} - \boldsymbol{z}^{(b,i)}) \cdot q_\ell^{-1}$.

   For each $b \in \{0,1\}$ and $i \in [0, \ell-1]$: Let $[\![\boldsymbol{s}^{(b,i)}]\!]_i$ be the i-th row vector of the matrix $\langle\!\langle \mathsf{NMat}(c_b) \rangle\!\rangle$. P and V locally compute $[\![\boldsymbol{z}^{(b,i)}]\!]_i := ([\![\boldsymbol{s}^{(b,i)}]\!]_i - [\![\boldsymbol{z}^{(b,i)}]\!]_i) \cdot q_\ell^{-1}$.

7. For each $b \in \{0,1\}$: P constructs a new matrix $\boldsymbol{Z}^{(b)} \in \mathbb{Z}^{\ell \times N}$ by setting the i-th row of $\boldsymbol{Z}^{(b)}$ as $\boldsymbol{z}^{(b,i)}$ for $i \in [0, \ell-1]$. Notice that, $\boldsymbol{Z}^{(b)}$ can be viewed as a NTT form matrix $\mathsf{NMat}(z_b)$ of a ring element $z_b \in \mathcal{R}_{Q_{\ell-1}}$.

   For each $b \in \{0,1\}$: P and V locally construct $\langle\!\langle \mathsf{NMat}(z_b) \rangle\!\rangle$ by setting the i-th row of $\langle\!\langle \mathsf{NMat}(z_b) \rangle\!\rangle$ as $[\![\boldsymbol{z}^{(b,i)}]\!]_i$ for $i \in [0, \ell-1]$.

8. If P honestly executes the above steps, then $\mathsf{NMat}(z_b) = \mathsf{NMat}(c'_b)$ for $b \in \{0,1\}$ should hold.

   P proves above as follows. For each $b \in \{0,1\}$: P invokes OUTPUT of $\mathcal{F}_{\mathsf{authZK}}$ to reveal $\mathsf{NMat}(z_b)$ to V. V checks if $\mathsf{NMat}(z_b) = \mathsf{NMat}(c'_b)$ holds. If so, V outputs TRUE; else, V outputs FALSE.

Figure 9: Protocol for modulus-switching of CKKS $\Pi_{\mathsf{ZK\text{-}MS}}$

7.1 Performance Evaluation

7.1.1 Evaluation of Online Protocol

To illustrate the practical efficiency and the scalability of our protocols, we conduct experiments for the following four scenarios:

Scenario (1): Ciphertext-ciphertext multiplication. We evaluate the performance for proving 10 ciphertext-ciphertext multiplications and report the amortized results. For both BGV and CKKS, we set $N = 8192$ and $L = 2$, and select two 58-bit primes and two 44-bit primes as the HE modulus.

Scenario (2): Matrix-vector multiplication (MVM). We consider the case where an encrypted order-n square matrix $\boldsymbol{M}$ is multiplied with an encrypted length-n vector, and we use the MVM technique in [50] where the matrix is stored in the diagonal order. For both BGV and CKKS, we use the same parameters and modulus as in Scenario (1), and we set $n = 256$. That is, we will run 16 (resp. 32) instances in a batch for CKKS (resp. BGV); thus, for MVM, we will report the amortized performance.

Scenario (3): Logistic regression training approximation (LRA). We consider the homomorphic evaluation for LRA using CKKS in [54]. We report the evaluation result for one iteration training of LRA. We use the parameter $N = 16384$ and $L = 6$, and select two 55-bit primes and six 50-bit primes as HE modulus. For LRA parameter, we set the number of features as 15, and set the number of samples as 512, so all features of samples can be encrypted in a single CKKS ciphertext.

Scenario (4): Comparison approximation (CA). We consider the case where both the client and server have some private numbers and wish to compare whose numbers are larger without disclosing their inputs. We take the homomorphic comparison approach for CKKS in [25], which uses high-degree polynomials to approximate the sign function. We use the parameter $N = 16384$ and $L = 7$, and use two 50-bit primes and seven 48-bit primes as HE modulus. Under these parameters, 8192 comparisons are made in a batch.

We put the evaluation results for above scenarios in Table 1. Our prover time is very fast: when compared with HE computation time, it occurs roughly 27-30$\times$ overhead for CKKS and 32$\times$ overhead for BGV.

7.1.2 Evaluation of Preprocessing Protocol

In many ZKP schemes (e.g. [42, 48, 52]), they assume there is a trusted third party (TTP) who is responsible for generating the correlated randomness during the preprocessing phase. Our preprocessing protocol can be instantiated with this straightforward approach. However, our preprocessing protocol can also be instantiated with the secure two-party computation (2PC) approach, avoiding the TTP assumption.

Our preprocessing protocol highly relies on random sVOLE correlations. We observe that, although there are some highly-efficient sVOLE protocols [64, 72], their implementations [62, 70] are tailored for the special cases $r = 1$ or $r = 2$ and are not suitable for HE evaluations scenario, where r denotes the degree of the extension field. Therefore, we re-implement the sVOLE protocol [72] using the NTL library [66]. Our implementation can support larger r and is suitable for our VOLE-ZKP for HE evaluations.

We report the performance of our preprocessing protocol for LRA scenario under different network conditions in Table 2. The network conditions are emulated using Linux traffic control command. In our first experiment, the bandwidth is set to 1Gbps, 200Mbps and 50Mbps, respectively. In our second experiment, the bandwidth is fixed to 1Gbps, and we simulate the network latency of 2, 10 and 30ms. Since our preprocessing protocol is constant-round, and it requires sublinear communication, the change in network conditions only has a minor impact on its runtime.

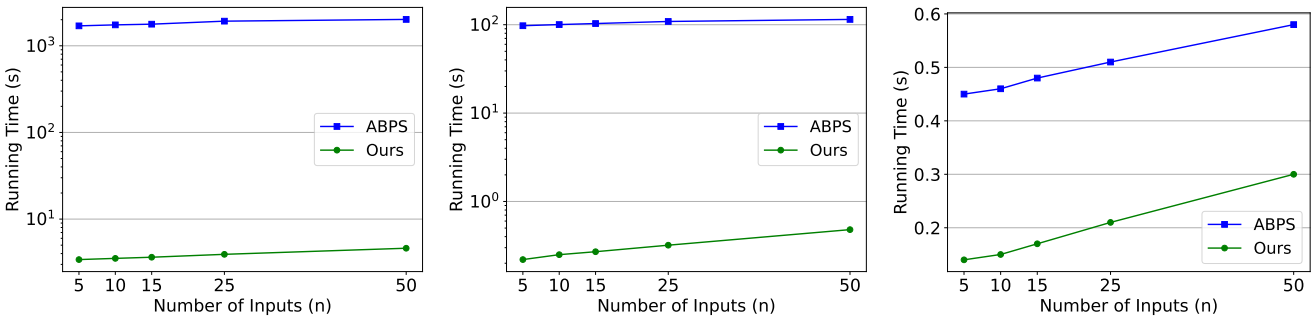
Table 2: The performance of our preprocessing protocol under LRA scenario.

Bandwidth	1Gbps	200Mbps	50Mbps
Time (s)	63.75	64.09	64.55
Latency	2ms	10ms	30ms
Time (s)	64.21	64.77	65.94

7.2 Comparison with Other Works

7.2.1 Qualitative Comparison

We provide a qualitative comparison with the most related works: zkSNARK-based vHEs. Take the SOTA works [4, 5, 40, 68] as examples, due to the use of zkSNARKs, they require $O(N \log N)$ prover time while their proof size and verifier time are sublinear in N, where N denotes the circuit size. In contrast, our VOLE-ZK protocols requires $O(N)$ prover time, verifier time and proof size. Therefore, from the aspect of asymptotic complexity, our prover overhead is much better than this line of



(a) Preprocessing Time (b) Prover Time (c) Verifier Time

Figure 10: Comparison with ABPS [5] for varying input numbers.

works, at the cost of higher verifier overhead and proof size. However, from the aspect of concrete performance, our verifier overhead is still better than [5,68] for common workloads; see details in Section 7.2.2.

7.2.2 Quantitative Comparison

We provide a quantitative comparison with ABPS protocol [5] and TW protocol [68], since they offer great open-sourced implementations.

Comparison with ABPS. In ABPS protocol, the authors employed a latticed-based zkSNARK [52] to prove HE evaluations, and they implemented their protocol for BGV. More concretely, they implemented their protocols for a two-layer homomorphic circuit as follows. Let n be a parameter. In the first layer, given ciphertexts $\{\mathsf{ct}_{0,i}\}_{i\in[n]}$ and plaintexts $\{a_{0,i,j}, b_{0,j}\}_{i\in[n],j\in[3]}$, the circuit computes $\mathsf{ct}_{1,j} = (\sum_{i=1}^{n} a_{0,i,j} \cdot \mathsf{ct}_{0,i} + b_{0,j})^2$ for $j \in [3]$; in the second layer, given previously computed ciphertexts $\{\mathsf{ct}_{1,i}\}_{i\in[3]}$ and plaintexts $\{a_{1,i}, b_1\}_{i\in[3]}$, the circuit computes $\mathsf{ct}_2 = (\sum_{i=1}^{3} a_{1,i} \cdot \mathsf{ct}_{1,i} + b_1)^2$ as output. This circuit approximates the feedforward evaluation of basic neural network layers.

To compare with ABPS protocol, we implement our protocols for the above two-layer homomorphic circuit and select the same parameters as in [5]: we set $N = 4096$, and we select the HE modulus as the product of three primes. We evaluate both ABPS and our protocol under a 1Gbps network with 2ms delay and present the results in Figure 10. As is shown, our protocol has significant advantages over ABPS protocol in running time. Take $n = 5$ as an example, our preprocessing time is roughly 497× faster, our prover time is roughly 443× faster and our verifier time is roughly 3× faster. Note that we use the 2PC approach to instantiate the preprocessing phase, while ABPS assume a TTP, which is a weaker threat model and easier to design than 2PC. Nevertheless, their preprocessing cost is still high because they need to prepare a latticed-based Common Reference String (CRS) whose size is roughly 11.6GB, and the CRS generation is very time-consuming.

Our proof size is larger than ABPS. ABPS has a constant proof size, i.e., 0.18MB. In contrast, our proof size ranges from 5.81-14.25MB. We note that, despite our larger proof size, our end-to-end performance is still much better than ABPS protocol due to our much faster running time.

Comparison with TW. Thibault and Walter employed a zkSNARK called Plonky2 [63] to prove HE evaluations, and they aimed for TFHE scheme [26]. More precisely, they implemented their protocols to prove a full programmable bootstrapping (a.k.a, PBS) operation. We evaluate their codes using our machine, and it turns out that their prover time is 2900.35s, their verifier time is 1.99s and their proof size is 0.18MB. We also evaluate a full PBS operation of TFHE in [2] using the same parameter as TW, and the computation time is 21.31ms. Thus, their prover time is roughly 136102× higher than the HE computation time.

We intuitively report the prover overhead and verifier overhead for us and TW in Table 3. Note that TFHE is very different from CKKS and BGV, and it might be unfair to compare with them directly. We pick TW just because there are rather few implemented vHE schemes available. It turns out that our protocols have advantages in both prover's overhead and verifier's overhead, even if TW considers multi-thread optimizations (we evaluate their codes using a machine with 16 vCPUs) while ours do not. Especially, our prover overhead is about three orders of magnitude lower than TW.

Table 3 only counts the online phase, and TW protocol does not require a preprocessing phase because Plonky2 has a transparent setup. By observing Table 1 and Table 2, we can infer that if the preprocessing time was taken into consideration, our end-to-end overhead will increase by about one order of magnitude. In this case our prover overhead is still more than two orders of magnitude lower than TW, but our verifier overhead will be several times higher.

7.3 Scalability Analysis

We have evaluated protocols across various workloads and input sizes in previous sections, and the prover overhead (resp. verifier overhead) consistently remains around 30× (resp. 20×) for CKKS. This indicates that our protocols exhibit strong scalability.

Table 3: Comparison with TW [68] for overhead (how many times slower than HE computation). TW considers multi-thread optimizations while this work ($\mathcal{Z}$HE) does not.

	HE Scheme	Scenario	Prover Overhead	Verifier Overhead
TW	TFHE	PBS	136102	93
$\mathcal{Z}$HE	BGV	MVM	32	20
		NN†	36	21
	CKKS	MVM	30	20
		NN	34	21
		LRA	27	17

† Refer to basic neural network layers approximation, as discussed in Section 7.2.2. Parameter n is set as 5.

8 Discussions

Here we provide some discussions about this work.

Noise flooding. In this work, we focus on ZKP for HE evaluations, which is the main bottleneck in vHEs. To develop an end-to-end application—such as secure neural network inference—the server must introduce additional noise to the computed results to hide the possible leakages (know as noise flooding in [32]). For BFV and BGV schemes, we could leverage established noise flooding techniques such as those in [13]. However, for TFHE and CKKS schemes, noise flooding remains an active area of ongoing research ([28,49]). Given the evolving nature of these methods, we have opted not to delve deeply into their specifics in this paper, reserving them as a subject for future exploration.

Bootstrapping. In this work, we mainly deal with CKKS and BGV's leveled nature that allows one doing at most L multiplications on a ciphertext. When unbounded multiplications are needed, a procedure named bootstrapping will be applied to obtain a fully HE. Bootstrapping achieves this by refreshing a low-level ciphertext, lifting it to a higher level to enable more multiplications. In Appendix C, we take CKKS as an example to illustrate that the bootstrapping can satisfy Definition 3, so our VOLE-ZKPs can be used to prove the correct execution of bootstrapping.

Limitations. Although we have made progresses in building fast ZKPs for HE evaluations, there are still some limitations that deserve further exploration, and we list them as follows.

Proof size. The main drawback of our scheme is its linear proof size, which is a common problem of most VOLE-ZKPs (e.g. [11, 31, 72, 73]). Therefore, our proof size is larger than ABPS and TW protocols, since they use zkSNARKs with sublinear proof sizes. However, a recent work [56] has shown that VOLE-ZKPs can achieve sublinear proof sizes. Even if we take communication cost into consideration, our end-to-end performance is still better than ABPS and TW protocols due to our fast prover time. Take the complex workload LRA in Table 1 for example, transmitting the 186MB proof could be done in tens of seconds, which would only increase our end-to-end time by about $1\times$. Furthermore, we argue that a relatively large proof is acceptable in some vHE scenarios, where the proof is stored in a cloud instance, and the client will download and check the proof only if he has doubts about the returned results.

Public verifiability. [68] is public verifiable (i.e., any participants could act as a verifier), while this work and [5] are designated-verifier (i.e., the prover can only prove to one verifier at a time). In [8], Baum et al. proposed a paradigm called "VOLE-in-the-head" and showed how to transform the designated-verifier VOLE-ZKP in [75] into a public-verifiable ZKP. We conjecture that our work could also be transformed into public-verifiable ZKPs by following this paradigm, and leave the formal treatment as a future work.

Comparison with 2PC. Given that our protocol may incur a substantial communication cost, one might wonder how it compares to a malicious 2PC protocol. The advantages of our approach are as follows: Compared to garbled-circuits-based protocols (e.g., [71]), our protocol typically requires much less communication. In contrast, secret-sharing-based protocols (e.g., [29]) generally cannot achieve constant communication rounds as our protocol does.

Towards IND-CCA1. Typically, (F)HE schemes satisfy IND-CPA security but not IND-CCA1 security [34], e.g. decrypting the bootstrapping key directly exposes the private key. Theoretically, one can build an IND-CCA1-secure (F)HE from an IND-CPA-secure (F)HE by applying the Naor-Yung paradigm [60] which uses NIZK [18]. However, we note that simply verifying the correctness of HE evaluations—as done in our work and [5, 68]—is not sufficient for this construction. We note that, [18] additionally requires NIZKs for proving the well-formness of (F)HE ciphertext, a task that demands significant additional effort.

9 Conclusion

This paper present novel and efficient VOLE-ZKPs for modulo and (I)NTT operations, and build customized VOLE-ZKP for HE evaluations (named $\mathcal{Z}$HE) on top of them. Our VOLE-ZKP is applicable to all RLWE-based HEs, e.g., BGV and CKKS. We implement our protocols and conduct extensive experiments to demonstrate our VOLE-ZKP is scalable and has a faster prover time. Particularly, compared to the SOTA works, our prover's overhead is 2-3 orders of magnitude lower.

Acknowledgement

We thank Peirong Wu for pointing out a bug that appeared in Section 4 of the proceeding version. Bingsheng Zhang is supported by the National Natural Science Foundation of China (Grant No. 62232002) and Input Output (iohk.io).

References

[1] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL.

[2] TFHE. https://github.com/tfhe/tfhe.

[3] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority mpc for malicious adversaries—breaking the 1 billion-gate per second barrier. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 843–862. IEEE, 2017.

[4] Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. HELIOPOLIS: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. Cryptology ePrint Archive, Paper 2023/1949, 2023.

[5] Shahla Atapoor, Karim Baghery, Hilder V. L. Pereira, and Jannik Spiessens. Verifiable FHE via lattice-based SNARKs. *IACR Communications in Cryptology*, 2024.

[6] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. META-BTS: Bootstrapping precision beyond the limit. In *ACM CCS 2022*.

[7] Jean-Claude Bajard, Julien Eynard, Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *SAC 2016*.

[8] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In *CRYPTO 2023*.

[9] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl. Moz$\mathbb{Z}_{2^k}$arella: Efficient vector-OLE and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. In *CRYPTO 2022*.

[10] Carsten Baum, Samuel Dittmer, Peter Scholl, and Xiao Wang. Sok: vector ole-based zero-knowledge protocols. *Designs, Codes and Cryptography*, 2023.

[11] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In *CRYPTO 2021*.

[12] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT 2011*.

[13] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. In *PKC 2021*.

[14] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *ACM CCS 2018*.

[15] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM TOCT*, 2014.

[16] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In *ICML*, 2019.

[17] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE FOCS 2001*.

[18] Ran Canetti, Srinivasan Raghuraman, Silas Richelson, and Vinod Vaikuntanathan. Chosen-ciphertext secure fully homomorphic encryption. In *PKC 2017*.

[19] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS 2017*.

[20] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. Veritas: Plaintext encoders for practical verifiable homomorphic encryption. In *ACM CCS 2024*.

[21] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In *EUROCRYPT 2019*.

[22] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT 2018*.

[23] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full rns variant of approximate homomorphic encryption. In *SAC 2018*.

[24] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT 2017*.

[25] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In *ASIACRYPT 2020*.

[26] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 2020.

[27] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *ACM CCS 2021*.

[28] Morten Dahl, Daniel Demmler, Sarah El Kazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P Smart, Samuel Tap, and Michael Walter. Noah's ark: Efficient threshold-fhe using noise flooding. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 35–46, 2023.

[29] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*.

[30] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Improving line-point zero knowledge: Two multiplications for the price of one. In *ACM CCS 2022*.

[31] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In *ITC 2021*.

[32] Léo Ducas and Damien Stehlé. Sanitization of fhe ciphertexts. In *EUROCRYPT 2016*.

[33] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[34] Prastudy Fauzi, Martha Norberg Hovd, and Håvard Raddum. On the ind-cca1 security of fhe schemes. *Cryptography*, 6(1):13, 2022.

[35] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO 1986*.

[36] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *ACM CCS 2014*.

[37] Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In *PKC 2020*.

[38] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 225–255. Springer, 2017.

[39] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-knowledge SNARKs as a service. In *USENIX Security 2023*.

[40] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In *CRYPTO 2024*.

[41] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO 2010*.

[42] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT 2013*.

[43] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *ACM STOC 2009*.

[44] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. {ZKBoo}: Faster {Zero-Knowledge} for boolean circuits. In *Usenix Security 2016*.

[45] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *ACM STOC 1987*.

[46] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *ACM STOC 2013*.

[47] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *ACM STOC 1985*.

[48] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016*.

[49] Qian Guo, Denis Nabokov, Elias Suvanto, and Thomas Johansson. Key recovery attacks on approximate homomorphic encryption with {Non-Worst-Case} noise flooding countermeasures. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 7447–7461, 2024.

[50] Shai Halevi and Victor Shoup. Algorithms in HElib. In *CRYPTO 2014*.

[51] Mi-Ying (Miryam) Huang, Baiyu Li, Xinyu Mao, and Jiapeng Zhang. Fully homomorphic encryption with efficient public verification. Cryptology ePrint Archive, Paper 2024/1764, 2024.

[52] Yuval Ishai, Hang Su, and David J. Wu. Shorter and faster post-quantum designated-verifier zkSNARKs from lattices. In *ACM CCS 2021*.

[53] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS 2018*.

[54] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 2018.

[55] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *ICML*, 2022.

[56] Fuchun Lin, Chaoping Xing, and Yizhou Yao. Interactive line-point zero-knowledge with sublinear communication and linear computation. In *ASIACRYPT 2024*.

[57] Fuchun Lin, Chaoping Xing, and Yizhou Yao. More efficient zero-knowledge protocols over $\mathbb{Z}_{2^k}$ via galois rings. In *CRYPTO 2024*.

[58] Jian Liu, Jingyu Li, Di Wu, and Kui Ren. PIRANA: Faster multi-query PIR via constant-weight codes. In *IEEE S&P 2024*.

[59] Tao Lu, Haoyu Wang, Wenjie Qu, Zonghui Wang, Jinye He, Tianyang Tao, Wenzhi Chen, and Jiaheng Zhang. An efficient and extensible zero-knowledge proof framework for neural networks. Cryptology ePrint Archive, Paper 2024/703, 2024.

[60] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *ACM STOC 1990*.

[61] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO 2011*.

[62] Lance Roy Peter Rindal. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[63] Ploygon. Plonky2. https://github.com/mir-protocol/plonky2.

[64] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In *CRYPTO 2023*.

[65] Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 1980.

[66] Victor Shoup. NTL: a library for doing numbery theory. https://github.com/libntl/ntl.

[67] Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 2014.

[68] Louis Tremblay Thibault and Michael Walter. Towards verifiable FHE in practice: Proving correct execution of TFHE's bootstrapping using plonky2. Cryptology ePrint Archive, Paper 2024/451, 2024.

[69] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. Verifiable fully homomorphic encryption, 2023.

[70] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit.

[71] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *ACM CCS 2017*.

[72] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *IEEE S&P 2021*.

[73] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In *USENIX Security 2021*.

[74] Chenkai Weng, Kang Yang, Zhaomin Yang, Xiang Xie, and Xiao Wang. AntMan: Interactive zero-knowledge proofs with sublinear communication. In *ACM CCS 2022*.

[75] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *ACM CCS 2021*.

[76] Yibin Yang, David Heath, Carmit Hazay, Vladimir Kolesnikov, and Muthuramakrishnan Venkitasubramaniam. Batchman and robin: Batched and non-batched branching for interactive zk. In *ACM CCS 2023*.

[77] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International symposium on symbolic and algebraic manipulation*, 1979.

A Security Proofs

A.1 Proof of Theorem 1

Theorem 1. *The protocol $\Pi_{\mathsf{ZK\text{-}Mod}}$ UC-realizes the* MOD *command of $\mathcal{F}_{\mathsf{authZK}}$ in the $\{\mathcal{F}_{\mathsf{T\text{-}Rand}}, \mathcal{F}_{\mathsf{authZK}}\}$-hybrid world (using only basic commands of $\mathcal{F}_{\mathsf{authZK}}$) with statistical error $\max\{(q_i^{-r_i})_{i\in\mathcal{M}}\}$ against a static malicious adversary.*

Proof. We consider two cases where the adversary A corrupts the prover or the verifier. In each case, we will construct a simulator S which runs the A internally as a subroutine, and S will emulate $\mathcal{F}_{\mathsf{authZK}}$ with only basic commands for A. In both case, we will prove that no environment Z will distinguish the real-world execution from the ideal-world execution.

Case I: Malicious Prover. In this case, we consider the following simulation strategy of S:

1. S emulates $\mathcal{F}_{\mathsf{T\text{-}Rand}}$ and the basic commands of $\mathcal{F}_{\mathsf{authZK}}$ for A, therefore S can learn all the information of $(\{[\![t^{(i)}]\!]_i\}_{i\in\mathcal{M}}, \{[\![t_i^{(\mathsf{pos})}]\!]_{\mathsf{B}}\}_{i\in[0,m-1]}, [\![r]\!]_{\mathsf{B}}, \{[\![r]\!]_i, [\![s^{(i)}]\!]_i\}_{i\in\mathcal{M}\setminus\{\mathsf{pos}\}})$ and $\{[\![x^{(i)}]\!]_i\}_{i\in\mathcal{M}}$.
2. S acts as the honest verifier to interact with A. If A can send the messages that make an honest verifier accept, S checks if $x^{(j)} \stackrel{?}{=} x^{(\mathsf{pos})} \mod q_j$ for each $j \in \mathcal{M}\setminus\{\mathsf{pos}\}$. If so, S forwards the malicious prover's shares of $\{[\![x^{(i)}]\!]_i\}_{i\in\mathcal{M}}$ to $\mathcal{F}_{\mathsf{authZK}}$; otherwise, S aborts.

Now we prove the indistinguishability. Notice that, since S acts as the honest verifier, so the environment can distinguish the real world from the ideal world if and only if the adversary cheats without being detected. We observe that, in $\Pi_{\mathsf{ZK\text{-}Mod}}$, the only way for the adversary to cheat without being detected is to successfully forge the MAC tags, which happens at the probability of $\max\{(q_i^{-r_i})_{i\in\mathcal{M}}\}$. In conclusion, in the case of the malicious prover, the real-world execution is statistically indistinguishable from the ideal-world execution.

Case II: Malicious Verifier. In this case, we consider the following simulation strategy of S:

1. S emulates $\mathcal{F}_{\mathsf{T\text{-}Rand}}$ and the basic commands of $\mathcal{F}_{\mathsf{authZK}}$ for A, therefore S can learn all the information of $(\{[\![t^{(i)}]\!]_i\}_{i\in\mathcal{M}}, \{[\![t_i^{(\mathsf{pos})}]\!]_{\mathsf{B}}\}_{i\in[0,m-1]}, [\![r]\!]_{\mathsf{B}}, \{[\![r]\!]_i, [\![s^{(i)}]\!]_i\}_{i\in\mathcal{M}\setminus\{\mathsf{pos}\}})$ and $\{[\![x^{(i)}]\!]_i\}_{i\in\mathcal{M}}$. Then S forwards the malicious verifier's shares of $\{[\![x^{(i)}]\!]_i\}_{i\in\mathcal{M}}$ to $\mathcal{F}_{\mathsf{authZK}}$ on behalf of the corrupted dummy verifier in the ideal-world execution.
2. Upon receiving TRUE from $\mathcal{F}_{\mathsf{authZK}}$, S picks a random bit c and opens c to V^* at first. Then S opens $h^{(i)} = 0$ to V^* for every $i \in \mathcal{M}\setminus\{\mathsf{pos}\}$. Notice that, S is able to do that since $\mathcal{F}_{\mathsf{T\text{-}Rand}}$ and the basic commands of $\mathcal{F}_{\mathsf{authZK}}$ for A are emulated by S, so S knows all the MAC keys of V^*.
3. Upon receiving FALSE from $\mathcal{F}_{\mathsf{authZK}}$, S randomly picks an index $i \in \mathcal{M}\setminus\{\mathsf{pos}\}$ and opens $h^{(i)} = 1$ to the malicious verifier.

Now we prove the indistinguishability. Firstly, the distribution of the MAC tags are identical in both real-world execution and ideal-world execution. Secondly, the distribution of the opened messages are also identical in both real-world execution and ideal-world execution, since c is masked by a random bit in real-world execution and c is randomly picked in the ideal-world execution. Therefore, in the case of the malicious verifier, the real-world execution is perfectly indistinguishable from the ideal-world execution. □

A.2 Proof of Theorem 2

Theorem 2. *The protocol $\Pi_{\mathsf{ZK\text{-}NTT}}$ UC realizes the* NTT *command of the functionality $\mathcal{F}_{\mathsf{authZK}}$ in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid world (using only basic commands of $\mathcal{F}_{\mathsf{authZK}}$) with statistical error $\frac{N-1}{q_i}$ against a static malicious adversary.*

Proof. The proof is analogous to the proof of Theorem 1. We consider the following two cases.

Case I: Malicious Prover. In this case, we consider the following simulation strategy of S:

1. S emulates the basic commands of $\mathcal{F}_{\mathsf{authZK}}$ for A, so S knows the malicious prover's shares of $[\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!]$.
2. S acts as the honest verifier to interact with A.
3. S aborts, if $\boldsymbol{a}, \boldsymbol{b}$ do not correspond to the same polynomial but the verification checks pass; S forwards the prover's shares of $[\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!]$ to $\mathcal{F}_{\mathsf{authZK}}$ on behalf of the corrupted dummy prover in the ideal-world, otherwise.

Now we prove the indistinguishability. It is easy to see that the probability of the environment Z distinguishing the ideal-world execution from the real-world execution is the probability of the simulator S aborting. The simulator S aborts when (i) the adversary A is able to forge the valid MAC tag which happens at probability $p_i^{r_i}$; (ii) let $a(x)$ (resp. $b(x)$) be the polynomial that $\boldsymbol{a}$ (resp. $\boldsymbol{b}$) corresponds to, $a(x)$ and $b(x)$ are two distinct polynomials, but $a(s) = b(s)$, which happens at probability $\frac{N-1}{q_i}$ due to Schwartz-Zippel lemma [65,77].

Case II: Malicious Verifier. In this case, we consider the following simulation strategy of S:

1. S emulates the basic commands of $\mathcal{F}_{\mathsf{authZK}}$ for A, so S knows the malicious verifier's shares of $[\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!]$, i.e. the global MAC key Δ_i and the local MAC keys $\boldsymbol{k_a}, \boldsymbol{k_b}$.
2. S forwards $\Delta_i, \boldsymbol{k_a}, \boldsymbol{k_b}$ to $\mathcal{F}_{\mathsf{authZK}}$ on behalf of the corrupted dummy verifier in the ideal-world.

3. If $\mathcal{F}_{\mathsf{authZK}}$ returns TRUE, S forges the valid MAC tag of $[\![0]\!]$ and sends it to A since S knows $\Delta_i, \boldsymbol{k_a}, \boldsymbol{k_b}$; if $\mathcal{F}_{\mathsf{authZK}}$ returns FALSE, S samples a invalid MAC tag and sends it to A.

The perfect indistinguishability between the real-world execution and the ideal-world execution is trivial. We also provide the intuition that why malicious verifier cannot learn anything about the honest prover's witness. During the interaction, the honest prover only reveal $w = 0$ and the corresponding MAC tag m_w which is independent of $\boldsymbol{a}, \boldsymbol{b}$; thus, the malicious verifier cannot learn anything about $\boldsymbol{a}, \boldsymbol{b}$. □

B VOLE-ZKPs for More HE Evaluation

Here we take CKKS as an example to show how we prove key-switching, ciphertext-ciphertext multiplication and rotation of CKKS.

B.1 For Key-Switching

We also provide a VOLE-ZKP for key-switching of CKKS, i.e., given the witness $\mathsf{ct} \in \mathcal{R}^2_{Q_\ell}$, the public switching key $\mathsf{swk} = \{\mathsf{swk}_i\}_{i\in[0,L]}$ and the public output $\mathsf{ct}' \in \mathcal{R}^2_{Q_\ell}$, P wants to convince V that $\mathsf{ct}' := \mathsf{KeySwitch}(\mathsf{swk}, \mathsf{ct})$ without leaking anything about ct.

Formally, we put our protocol $\Pi_{\mathsf{ZK\text{-}KS}}$ in Figure 11 and its security is guaranteed by Theorem 3. Notice that, ModSwitch is contained in KeySwitch, so our protocol $\Pi_{\mathsf{ZK\text{-}KS}}$ will invoke $\Pi_{\mathsf{ZK\text{-}MS}}$ as a subroutine. We also note that, each public switching key swk_i is also stored in NTT form matrix, and we denote it as $\mathsf{NMat}(\mathsf{swk}_i)$. Similarly as in Section 6.2.1, in $\Pi_{\mathsf{ZK\text{-}KS}}$, we will describe KeySwitch step by step and demonstrate that how we prove KeySwitch are honestly executed in colored boxes.

B.2 For Ciphertext-Ciphertext Multiplication

We here provide a VOLE-ZKP for ciphertext-ciphertext multiplication of CKKS, i.e., given the witness $\mathsf{ct}, \mathsf{ct}' \in \mathcal{R}^2_{Q_\ell}$, the public relinearization keys rlk and the public output $\mathsf{ct}_{\mathsf{Mult}} \in \mathcal{R}^2_{Q_\ell}$, P wants to convince V that $\mathsf{ct}_{\mathsf{Mult}} = \mathsf{Mult}(\mathsf{rlk}, \mathsf{ct}, \mathsf{ct}')$ without leaking anything about the witness $\mathsf{ct}, \mathsf{ct}'$. Formally, we put the protocol $\Pi_{\mathsf{ZK\text{-}Mult}}$ in Figure 12. The security of $\Pi_{\mathsf{ZK\text{-}Mult}}$ is guaranteed by Theorem 3.

B.3 For Rotation

We here provide a VOLE-ZKP for rotation of CKKS, i.e., given the witness $\mathsf{ct} \in \mathcal{R}^2_{Q_\ell}$, the public rotation keys $\mathsf{rk}^{(k)}$ w.r.t. the rotation step k, and the public output $\mathsf{ct}' \in \mathcal{R}^2_{Q_\ell}$, P wants to convince V that $\mathsf{ct}' = \mathsf{Rotate}(\mathsf{rk}^{(k)}, \mathsf{ct})$ without leaking anything about the witness ct.

Notice that, given a rotation step $k \in [1, \frac{N}{2} - 1]$, Rotate will first performs a permutation $\mathsf{Perm}^{(k)}$ on input ct, where $\mathsf{Perm}^{(k)}$ can be viewed a linear transformation, so this can be processed by P and V locally. For simplicity, we ignore the details of $\mathsf{Perm}^{(k)}$ here. Then Rotate will perform KeySwitch on the part of permuted results; note that, we have already showed that how to prove KeySwitch in $\Pi_{\mathsf{ZK\text{-}KS}}$ in Figure 11. Given above, it is easy to prove Rotate. We provide the detailed protocol description $\Pi_{\mathsf{ZK\text{-}Rot}}$ in Figure 13 and its security is guaranteed by Theorem 3.

C Discussion on Proving Bootstrapping of CKKS

There are extensive study on bootstrapping of CKKS ([21,22], etc.). Although they vary in details, as pointed in [6], all of them consist of the following steps: conversion between slots and coefficients (StC and CtS), modulus raising (ModRaise) and the evaluation of modular reduction (EvalMod). In the following, we will show that our VOLE-ZKPs can be used to prove the correct execution of these steps; therefore, our VOLE-ZKPs can be used to prove bootstrapping of CKKS.

In terms of CtS and StC, they are linear transformations; as pointed in [6], the only non-trivial part here is rotation. As for EvalMod, it is an arithmetic circuit evaluating a polynomial function which approximates the operation of modular reduction, and the only non-trivial part in it is ciphertext-ciphertext multiplications. (Besides these, they only require additions or plaintext-ciphertext multiplications, which can be easily proven by VOLE-ZKP.) Considering we have already shown that our VOLE-ZKP can be used to prove rotation and ciphertext-ciphertext multiplication in Appendix B, our VOLE-ZKP can deal with CtS, StC and EvalMod.

For ModRaise, it is used to transforming a level-ℓ ciphertext consisting of elements in $\mathcal{R}_{Q_\ell}$ into one with elements in $\mathcal{R}_{Q_{\ell'}}$, where $\ell' > \ell$ so we have $Q_{\ell'} > Q_\ell$ (recall that, $Q_\ell = \Pi_{i=0}^{\ell} q_i$ and q_i is a prime). In many practically used settings of CKKS bootstrapping (e.g., [55]), ℓ is set as 0. In this case, we assume the underlying ciphertext is $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}^2_{q_0}$, and we need to compute $c'_{b,i} = c_b \mod q_i$ for each $b \in \{0,1\}, i \in [0,\ell]$ and then use CRT to compute $c'_b \in \mathcal{R}_{Q_{\ell'}}$ from $\{c'_{b,i}\}_{i\in[0,\ell]}$. Notice that, this computation is contained in KeySwitch, so our VOLE-ZKP can be used to prove it. In a more general case where $\ell > 0$, a

Protocol $\Pi_{\mathsf{ZK\text{-}KS}}$

Input: P and V hold the switching key $\{\mathsf{NMat}(\mathsf{swk}_i)\}_{i\in[0,\ell]}$, where $\mathsf{NMat}(\mathsf{swk}_i) := (\mathsf{NMat}(\tilde{b}_i), \mathsf{NMat}(\tilde{a}_i))$ for $i \in [0,\ell]$, as well as the public output $\mathsf{NMat}(\mathsf{ct}') := (\mathsf{NMat}(c'_0), \mathsf{NMat}(c'_1))$ where $\mathsf{ct}' \in \mathcal{R}^2_{Q_\ell}$. P privately holds $\mathsf{NMat}(\mathsf{ct}) := (\mathsf{NMat}(c_0), \mathsf{NMat}(c_1))$ where $\mathsf{ct} \in \mathcal{R}^2_{Q_\ell}$. For notation convenience, here we set $q_{\ell+1} := p_0$ and the description of p_0 can be found in Section 2.3.

Protocol:

(Generating authenticated ciphertexts.) P and V obtain $\langle\!\langle \mathsf{NMat}(\mathsf{ct}) \rangle\!\rangle$ using the same approach as in Figure 9.

(Proving KeySwitch *has been executed correctly.)* P performs KeySwitch on $\mathsf{NMat}(\mathsf{ct})$, and P proves that KeySwitch has been performed correctly as follows:

1. For each $i \in [0,\ell]$: Let $\boldsymbol{s}^{(i)} \in \mathbb{F}_{q_i}^N$ be the i-th row of $\mathsf{NMat}(c_1)$, and P computes $\boldsymbol{t}^{(i)} := \mathsf{NTT}_i(\boldsymbol{s}^{(i)}) \in \mathbb{F}_{q_i}^N$.

   P proves above as follows. For each $i \in [0,\ell]$: Let $[\![\boldsymbol{s}^{(i)}]\!]_\ell$ be the i-th row of the matrix $\langle\!\langle \mathsf{NMat}(c_1) \rangle\!\rangle$. P invokes INPUT command of $\mathcal{F}_{\mathsf{authZK}}$ using $\boldsymbol{t}^{(i)}$ so that both parties can obtain $[\![\boldsymbol{t}^{(i)}]\!]_i$. P and V invoke NTT command of $\mathcal{F}_{\mathsf{authZK}}$ using $[\![\boldsymbol{t}^{(i)}]\!]_i$, $[\![\boldsymbol{s}^{(i)}]\!]_i$ so that V can be convinced that $[\![\boldsymbol{t}^{(i)}]\!]_i$ is correctly computed.

2. For each $i \in [0,\ell]$ and $j \in [0,\ell+1]$: If $i = j$, P sets $\boldsymbol{u}^{(i,i)} := \boldsymbol{t}^{(i)}$; else, P computes $\boldsymbol{u}^{(i,j)} := \boldsymbol{t}^{(i)} \mod q_j$.

   P proves above as follows. For each $i \in [0,\ell]$ and $j \in [0,\ell+1]$: If $i = j$, P and V set $[\![\boldsymbol{u}^{(i,i)}]\!]_i := [\![\boldsymbol{t}^{(i)}]\!]_i$. Else, P invokes INPUT command of $\mathcal{F}_{\mathsf{authZK}}$ using $\boldsymbol{u}^{(i,j)}$ so both parties can obtain $[\![\boldsymbol{u}^{(i,j)}]\!]_j$, then P and V invoke MOD command of $\mathcal{F}_{\mathsf{authZK}}$ using $[\![\boldsymbol{t}^{(i)}]\!]_i$ and $[\![\boldsymbol{u}^{(i,j)}]\!]_j$ so V can be convinced that $[\![\boldsymbol{u}^{(i,j)}]\!]_j$ is correctly computed.

3. For each $i \in [0,\ell]$ and $j \in [0,\ell+1]$: P computes $\boldsymbol{h}^{(i,j)} := \mathsf{NTT}_j(\boldsymbol{u}^{(i,j)}) \in \mathbb{F}_{q_j}^N$.

   P proves above as follows. For each $i \in [0,\ell]$ and $j \in [0,\ell+1]$: P invokes INPUT command of $\mathcal{F}_{\mathsf{authZK}}$ using $\boldsymbol{h}^{(i,j)}$ so both parties can obtain $[\![\boldsymbol{h}^{(i,j)}]\!]_i$, then P and V invoke NTT command of $\mathcal{F}_{\mathsf{authZK}}$ using $[\![\boldsymbol{u}^{(i,j)}]\!]_j$ and $[\![\boldsymbol{h}^{(i,j)}]\!]_j$ so V can be convinced that $[\![\boldsymbol{h}^{(i,j)}]\!]_j$ is correctly computed.

4. For each $i \in [0,\ell]$: P constructs a new matrix $\boldsymbol{H}^{(i)} \in \mathbb{Z}^{(\ell+2)\times N}$ by setting the j-th row of $\boldsymbol{H}^{(i)}$ as $\boldsymbol{h}^{(i,j)}$ for each $j \in [0,\ell+1]$. Notice that, $\boldsymbol{H}^{(i)}$ can be viewed as a NTT form matrix $\mathsf{NMat}(h_i)$ of a ring element $h_i \in \mathcal{R}_{Q_{\ell+1}}$, where $Q_{\ell+1} = p_0 Q_\ell$.

   For each $i \in [0,\ell]$: P and V locally construct $\langle\!\langle \mathsf{NMat}(h_i) \rangle\!\rangle$ by setting the j-th row of $\langle\!\langle \mathsf{NMat}(h_i) \rangle\!\rangle$ as $[\![\boldsymbol{h}^{(i,j)}]\!]_j$ for $j \in [0,\ell+1]$.

5. P locally constructs $\mathsf{NMat}(b'_i), \mathsf{NMat}(a'_i) \in \mathbb{Z}^{(\ell+1)\times N}$ from $\mathsf{NMat}(\tilde{b}_i), \mathsf{NMat}(\tilde{a}_i) \in \mathbb{Z}^{(L+1)\times N}$ as follows. For $i \in [0,\ell+1]$, P sets the i-th rows of $\mathsf{NMat}(b'_i), \mathsf{NMat}(a'_i)$ as the rows of $\mathsf{NMat}(\tilde{b}_i), \mathsf{NMat}(\tilde{a}_i)$ over $\mathbb{F}_{q_i}$, respectively.

   V locally constructs $\mathsf{NMat}(b'_i), \mathsf{NMat}(a'_i) \in \mathbb{Z}^{(\ell+1)\times N}$ from $\mathsf{NMat}(\tilde{b}_i), \mathsf{NMat}(\tilde{a}_i) \in \mathbb{Z}^{(L+1)\times N}$ as above.

6. For each $i \in [0,\ell]$: P computes $\mathsf{NMat}(g'_{0,i}) := \mathsf{NMat}(h_i) \odot \mathsf{NMat}(b'_i)$, $\mathsf{NMat}(g'_{1,i}) := \mathsf{NMat}(h_i) \odot \mathsf{NMat}(a'_i)$.

   For each $i \in [0,\ell]$: P and V locally compute $\langle\!\langle \mathsf{NMat}(g'_{0,i}) \rangle\!\rangle := \mathsf{NMat}(b'_i) \odot \langle\!\langle \mathsf{NMat}(h_i) \rangle\!\rangle$ and $\langle\!\langle \mathsf{NMat}(g'_{1,i}) \rangle\!\rangle := \mathsf{NMat}(a'_i) \odot \langle\!\langle \mathsf{NMat}(h_i) \rangle\!\rangle$.

7. P compute $(\mathsf{NMat}(g_0), \mathsf{NMat}(g_1)) := \sum_{i\in[0,\ell]} (\mathsf{NMat}(g'_{0,i}), \mathsf{NMat}(g'_{1,i}))$.

   P and V locally compute $(\langle\!\langle \mathsf{NMat}(g_0) \rangle\!\rangle, \langle\!\langle \mathsf{NMat}(g_1) \rangle\!\rangle) := \sum_{i\in[0,\ell]} (\langle\!\langle \mathsf{NMat}(g'_{0,i}) \rangle\!\rangle, \langle\!\langle \mathsf{NMat}(g'_{1,i}) \rangle\!\rangle)$.

8. P performs ModSwitch on input $(\mathsf{NMat}(g_0), \mathsf{NMat}(g_1), p_0 Q_\ell, Q_\ell)$ and gets the output $(\mathsf{NMat}(z_0), \mathsf{NMat}(z_1))$, where $(z_0, z_1) \in \mathcal{R}^2_{Q_\ell}$.

   P proves above by executing the step 1-7 of $\Pi_{\mathsf{ZK\text{-}MS}}$ in Figure 9 with V so both parties can obtain $\langle\!\langle \mathsf{NMat}(z_0) \rangle\!\rangle$ and $\langle\!\langle \mathsf{NMat}(z_1) \rangle\!\rangle$.

9. P computes $\mathsf{NMat}(z_0) := \mathsf{NMat}(c_0) + \mathsf{NMat}(z_0)$.

   P and V locally compute $\langle\!\langle \mathsf{NMat}(z_0) \rangle\!\rangle := \langle\!\langle \mathsf{NMat}(c_0) \rangle\!\rangle + \langle\!\langle \mathsf{NMat}(z_0) \rangle\!\rangle$.

10. If P honestly executes the above steps, then $\mathsf{NMat}(z_b) = \mathsf{NMat}(c'_b)$ for $b \in \{0,1\}$ should hold.

    P proves above as follows. For each $b \in \{0,1\}$: P invokes OUTPUT of $\mathcal{F}_{\mathsf{authZK}}$ to reveal $\mathsf{NMat}(z_b)$ to V, who checks if $\mathsf{NMat}(z_b) = \mathsf{NMat}(c'_b)$ holds. If so, V outputs TRUE; else, V outputs FALSE.

Figure 11: Protocol for key-switching of CKKS $\Pi_{\mathsf{ZK\text{-}KS}}$

Protocol $\Pi_{\mathsf{ZK\text{-}Mult}}$

Input: P and V hold the output $\mathsf{NMat}(\mathsf{ct}_{\mathsf{Mult}})$ and the relinearization key $\{\mathsf{NMat}(\mathsf{rlk}_i)\}_{i\in[0,\ell]}$. P privately holds $\mathsf{NMat}(\mathsf{ct}) := (\mathsf{NMat}(c_0), \mathsf{NMat}(c_1))$ and $\mathsf{NMat}(\mathsf{ct}') := (\mathsf{NMat}(c'_0), \mathsf{NMat}(c'_1))$.

Protocol:

(Generating authenticated ciphertexts.) P and V obtain $\langle\!\langle \mathsf{NMat}(\mathsf{ct}) \rangle\!\rangle$ and $\langle\!\langle \mathsf{NMat}(\mathsf{ct}') \rangle\!\rangle$ using the same approach as in Figure 9.

(Proving Mult *has been performed correctly.)* P performs Mult on $\mathsf{NMat}(\mathsf{ct}), \mathsf{NMat}(\mathsf{ct}')$, and P proves that the Mult has been performed correctly as follows:

1. P computes $\mathsf{NMat}(d_0) := \mathsf{NMat}(c_0) \odot \mathsf{NMat}(c_1)$, $\mathsf{NMat}(d_{1,0}) := \mathsf{NMat}(c_0) \odot \mathsf{NMat}(c'_1)$, $\mathsf{NMat}(d_{1,1}) := \mathsf{NMat}(c_1) \odot \mathsf{NMat}(c'_0)$, $\mathsf{NMat}(d_2) := \mathsf{NMat}(c'_0) \odot \mathsf{NMat}(c'_1)$ and $\mathsf{NMat}(d_1) := \mathsf{NMat}(d_{1,0}) + \mathsf{NMat}(d_{1,1})$.

   P proves above as follows. P invokes INPUT command of $\mathcal{F}_{\mathsf{authZK}}$ so both parties can obtain $\langle\!\langle \mathsf{NMat}(d_0) \rangle\!\rangle$, $\langle\!\langle \mathsf{NMat}(d_{1,0}) \rangle\!\rangle$, $\langle\!\langle \mathsf{NMat}(d_{1,1}) \rangle\!\rangle$, and $\langle\!\langle \mathsf{NMat}(d_2) \rangle\!\rangle$. Then P and V invoke MULT command of $\mathcal{F}_{\mathsf{authZK}}$ using $\langle\!\langle \mathsf{NMat}(c_0) \rangle\!\rangle$, $\langle\!\langle \mathsf{NMat}(c'_0) \rangle\!\rangle$ and $\langle\!\langle \mathsf{NMat}(d_0) \rangle\!\rangle$ so V can be convinced that $\langle\!\langle \mathsf{NMat}(d_0) \rangle\!\rangle$ is correctly computed; similarly, the correctness of $\langle\!\langle \mathsf{NMat}(d_{1,0}) \rangle\!\rangle$, $\langle\!\langle \mathsf{NMat}(d_{1,1}) \rangle\!\rangle$, and $\langle\!\langle \mathsf{NMat}(d_2) \rangle\!\rangle$ can also be checked by V. Finally, both parties locally compute $\langle\!\langle \mathsf{NMat}(d_1) \rangle\!\rangle := \langle\!\langle \mathsf{NMat}(d_{1,0}) \rangle\!\rangle + \langle\!\langle \mathsf{NMat}(d_{1,1}) \rangle\!\rangle$.

2. P performs KeySwitch on $(\{\mathsf{NMat}(\mathsf{rlk}_i)\}_{i\in[0,\ell]}, 0, \mathsf{NMat}(d_2))$ and gets the output $\mathsf{NMat}(\tilde{c}_0)$, $\mathsf{NMat}(\tilde{c}_1)$ where $(\tilde{c}_0, \tilde{c}_1) \in \mathcal{R}_{Q_\ell}$.

   P proves above by executing the step 1-9 of $\Pi_{\mathsf{ZK\text{-}KS}}$ in Figure 11 with V so both parties can obtain $\langle\!\langle \mathsf{NMat}(\tilde{c}_0) \rangle\!\rangle$ and $\langle\!\langle \mathsf{NMat}(\tilde{c}_1) \rangle\!\rangle$.

3. P computes $\mathsf{NMat}(\mathsf{ct}'_{\mathsf{Mult}}) := (\mathsf{NMat}(d_0), \mathsf{NMat}(d_1)) + (\mathsf{NMat}(\tilde{c}_0), \mathsf{NMat}(\tilde{c}_1))$.

   Parties compute $\langle\!\langle \mathsf{NMat}(\mathsf{ct}'_{\mathsf{Mult}}) \rangle\!\rangle := (\langle\!\langle \mathsf{NMat}(d_0) \rangle\!\rangle, \langle\!\langle \mathsf{NMat}(d_1) \rangle\!\rangle) + (\langle\!\langle \mathsf{NMat}(\tilde{c}_0) \rangle\!\rangle, \langle\!\langle \mathsf{NMat}(\tilde{c}_1) \rangle\!\rangle)$.

4. If P honestly executes the above steps, then $\mathsf{NMat}(\mathsf{ct}'_{\mathsf{Mult}}) = \mathsf{NMat}(\mathsf{ct}_{\mathsf{Mult}})$ holds.

   P proves above by invoking OUTPUT command of $\mathcal{F}_{\mathsf{authZK}}$ to reveal $\mathsf{ct}'_{\mathsf{Mult}}$ to V, who checks if $\mathsf{NMat}(\mathsf{ct}'_{\mathsf{Mult}}) = \mathsf{NMat}(\mathsf{ct}_{\mathsf{Mult}})$ holds. If so, V outputs TRUE; else, V outputs FALSE.

Figure 12: Protocol $\Pi_{\mathsf{ZK\text{-}Mult}}$

Protocol $\Pi_{\mathsf{ZK\text{-}Rot}}$

Input: P and V hold the public output $\mathsf{NMat}(\mathsf{ct}') := (\mathsf{NMat}(c'_0), \mathsf{NMat}(c'_1))$, the public rotation key $\{\mathsf{NMat}(\mathsf{rk}_i^{(k)})\}_{i\in[0,\ell]}$ w.r.t. the rotation step k. P privately holds $\mathsf{NMat}(\mathsf{ct}) := (\mathsf{NMat}(c_0), \mathsf{NMat}(c_1))$.

Protocol:

(Generating authenticated ciphertexts.) P and V obtain $\langle\!\langle \mathsf{NMat}(\mathsf{ct}) \rangle\!\rangle$ using the same approach as in Figure 9.

(Proving Rotate *has been performed correctly.)* P performs Rotate on $\mathsf{NMat}(\mathsf{ct})$, and P proves that the Rotate has been performed correctly as follows:

1. P computes $\mathsf{NMat}(\mathsf{ct}) := \mathsf{Perm}^{(k)}(\mathsf{NMat}(\mathsf{ct}))$.

   P and V locally compute $\langle\!\langle \mathsf{NMat}(\mathsf{ct}) \rangle\!\rangle := \mathsf{Perm}^{(k)}(\langle\!\langle \mathsf{NMat}(\mathsf{ct}) \rangle\!\rangle)$.

2. P performs KeySwitch on $(\{\mathsf{NMat}(\mathsf{rk}_i^{(k)})\}_{i\in[0,\ell]}, 0, \mathsf{NMat}(c_1))$ and gets the output $\mathsf{NMat}(\tilde{c}_0)$, $\mathsf{NMat}(\tilde{c}_1)$ where $(\tilde{c}_0, \tilde{c}_1) \in \mathcal{R}_{Q_\ell}$.

   P proves above by executing the step 1-9 of $\Pi_{\mathsf{ZK\text{-}KS}}$ in Figure 11 with V so both parties can obtain $\langle\!\langle \mathsf{NMat}(\tilde{c}_0) \rangle\!\rangle$ and $\langle\!\langle \mathsf{NMat}(\tilde{c}_1) \rangle\!\rangle$.

3. P computes $\mathsf{NMat}(\tilde{c}_0) := \mathsf{NMat}(c_0) + \mathsf{NMat}(\tilde{c}_0)$.

   P and V compute $\langle\!\langle \mathsf{NMat}(\tilde{c}_0) \rangle\!\rangle := \langle\!\langle \mathsf{NMat}(c_0) \rangle\!\rangle + \langle\!\langle \mathsf{NMat}(\tilde{c}_0) \rangle\!\rangle$.

4. If P honestly executes the above steps, then $\mathsf{NMat}(c'_b) = \mathsf{NMat}(\tilde{c}_b)$ holds for each $b \in \{0, 1\}$.

   P proves above by invoking OUTPUT command of $\mathcal{F}_{\mathsf{authZK}}$ to reveal $\tilde{c}_0, \tilde{c}_1$ to V, who checks if $\mathsf{NMat}(\tilde{c}_b) = \mathsf{NMat}(c'_b)$ holds for each $b \in \{0, 1\}$. If so, V outputs TRUE; else, V outputs FALSE.

Figure 13: Protocol $\Pi_{\mathsf{ZK\text{-}Rot}}$

procedure called fast base extension [7] is used, and this procedure can be decomposed into additions, modulo operations and (I)NTT operations; thus, our VOLE-ZKP can still support this general case. In a word, our VOLE-ZKP can deal with ModRaise.