Practical TFHE Ciphertext Sanitization for Oblivious Circuit Evaluation

Intak Hwang, Seonhong Min, Jinyeong Seo, and Yongsoo Song

Seoul National University
{intak.hwang, minsh, jinyeong.seo, y.song}@snu.ac.kr

Abstract. Homomorphic encryption (FHE) enables the computation of arbitrary circuits over encrypted data. A widespread application of HE is oblivious circuit evaluation, where a sender evaluates its private circuit over a receiver's encrypted data, covering scenarios such as oblivious inference and oblivious PRF protocols. However, while the security of HE guarantees the receiver's privacy against the sender, the privacy of the sender's circuit is not solely derived from the security of HE.

One effective solution to this problem is ciphertext sanitization, an algorithm that removes any information contained in a ciphertext except for the plaintext. Since its introduction by Ducas and Stehlé (Eurocrypt 2016), several approaches have been proposed for constructing sanitization algorithms for TFHE, but they remain highly impractical.

In this work, we present a novel sanitization algorithm for TFHE ciphertexts that is practically deployable. Unlike prior methods that introduce randomization throughout the entire bootstrapping procedure or require repeated bootstrappings, our approach involves only two lightweight randomization steps at the input and output of the original TFHE bootstrapping, without modifying its core operations. As a result, our algorithm achieves sanitization with a single bootstrapping and minimal randomization, fully leveraging the fast performance of TFHE bootstrapping.

In addition, we design a zero-knowledge argument of knowledge (ZKAoK) for TFHE ciphertexts and bootstrapping keys to address malicious receivers. To the best of our knowledge, this work is the first to construct a concrete ZKAoK that covers all receiver-sent materials, enabling a secure TFHE-based protocol against a malicious receiver.

We provide a proof-of-concept implementation to demonstrate the practicality of our solution. Our experiments show that it takes approximately 42.17 ms to sanitize a single TFHE ciphertext, achieving up to a 31× speedup compared to the state-of-the-art method by Kluczniak (CiC 2025).

Keywords: homomorphic encryption; TFHE; circuit privacy; sanitization; malicious security; zero-knowledge argument

1 Introduction

Homomorphic Encryption (FHE) is a cryptosystem that allows the evaluation of an arbitrary circuit over encrypted data, without any limitation on the circuit depth. Thanks to such functionality, it can be utilized to construct a low-communication two-party oblivious evaluation protocol for scenarios where the receiver owns private data and the sender owns a circuit. In this protocol, the receiver first encrypts its private data using an HE scheme and sends it to the sender. Then, the sender evaluates some circuit over the given ciphertexts and its own private data, and returns the output ciphertext to the receiver. Finally, the receiver decrypts the returned ciphertext and obtains the evaluation result on its data. By the security of the underlying HE scheme, it is guaranteed that the sender learns nothing about the private data of the receiver nor the evaluation result. However, generally the converse is not true: the receiver can learn some information about the sender's data, namely the circuit being evaluated or the sender's private inputs.

To address this problem, Gentry [Gen09] introduced the concept of circuit privacy. Roughly speaking, an HE scheme is circuit-private if the output ciphertext does not contain any information about the circuit after being evaluated. One common approach to achieving circuit privacy is noise flooding [Gen09], which erases the information left over in the ciphertext by adding an encryption of zero with an exponentially large noise term. Naturally, this leads to exponential overhead in the parameters, as one needs to guarantee correct decryption even with the added noise. In RLWE-based HE schemes such as BFV [Bra12, FV12], BGV [BGV12] and CKKS [CKKS17], this can be a minor problem, as the usual parameters of these schemes are already large, with ciphertext modulus often spanning hundreds of bits. However, some HE schemes, such as FHEW [DM15] and TFHE [CGGI16], use very compact parameters, making the noise flooding method unfeasible.

Alternatively, Ducas and Stehlé [DS16] proposed a circuit privacy solution based on ciphertext sanitization. Sanitization is an algorithm for HE ciphertexts that preserves the plaintext of the input while ensuring that any two ciphertexts encrypting the same plaintext become indistinguishable after sanitization. In other words, sanitization removes all information from the ciphertext other than the underlying plaintext. As a result, applying sanitization at the end of a homomorphic evaluation eliminates leakage about the sender's circuit, making it a sufficient condition for circuit privacy.

The method in [DS16] achieves sanitization by repeatedly applying two operations: adding moderately sized noise and performing bootstrapping. The core idea is that by successively injecting noise into two ciphertexts, their statistical distance gradually decreases. With a sufficient number of iterations, the ciphertexts eventually become statistically indistinguishable. The primary drawback of this approach is the need to perform multiple consecutive bootstrappings. For instance, under typical TFHE parameters, achieving a sufficiently low statistical distance requires approximately 5 to 10 cycles.

Recently, Bourse and Izabachène [BI22] and Kluczniak [Klu24] independently proposed alternative constructions for TFHE that address the overhead of repeated bootstrapping in [DS16]. The core idea in both works is to randomize the entire inner workings of TFHE bootstrapping. As a result, their constructions achieve sanitization with only a single invocation of bootstrapping. However, due to the overhead introduced by randomization, their approaches perform worse in practice than soak-spin-repeat sanitization. The main bottleneck is the extensive reliance on discrete Gaussian sampling, which is notoriously difficult to perform efficiently.

In addition, existing solutions for TFHE sanitization [DS16, BI22, Klu24] provide security only against semi-honest receivers, as they rely on the assumption that the receiver submits well-formed public keys and ciphertexts. A general approach to addressing this limitation is to attach a zero-knowledge argument of knowledge (ZKAoK) that proves the well-formedness of both public keys and ciphertexts. However, to the best of our knowledge, there exists no concrete construction of ZKAoKs that effectively validates the well-formedness of TFHE ciphertexts and public keys. As a result, developing a practical sanitization solution for TFHE and enhancing its security against malicious receivers remains an important open problem.

1.1 Our Contribution

In this work, we construct a novel sanitization algorithm for the TFHE scheme that addresses the aforementioned limitations. Unlike the approach in [DS16], our method requires only a single invocation of TFHE bootstrapping. Moreover, in contrast to [BI22, Klu24], our algorithm introduces randomization only at the beginning and end of the bootstrapping procedure, rather than randomizing the entire process. As a result, the performance of our sanitization algorithm is significantly faster than previous works, nearly matching the original TFHE bootstrapping. In addition, to address malicious receivers, we develop a novel ZKAoK for validating the well-formedness of TFHE public keys. To the best of our knowledge, this is the first construction that fully validates the well-formedness of TFHE public keys.

We implement our sanitization algorithm in Go and provide benchmark results. The experimental results show that the latency of sanitization is reduced from 7500ms of [BI22] and 1330ms of [Klu24] to 42.17ms, yielding a speedup with a factor of 177.85 and 31.53, respectively. Moreover, our method merely adds 0.81ms of overhead to the original TFHE bootstrapping, which is only 1.9% of the total runtime of bootstrapping. Finally, our implementation of the ZKAoK for TFHE public keys requires only 22.6 seconds, demonstrating practically deployable performance.

While our focus in this paper is on the TFHE scheme, our sanitization method is readily applicable to any AP-like cryptosystems [AP14] that employ monomial-based bootstrapping, such as FHEW [DM15] and LMKCDEY [LMK+23], with minimal modifications. We also emphasize that our sanitization algorithm introduces randomization only at the beginning and end of the TFHE bootstrapping

procedure, leaving all other subroutines unchanged. As a result, recent advances for blind rotation in TFHE bootstrapping [LMSS23, LY23, BCL+24] can be seamlessly integrated into our approach, further improving performance.

Finally, our sanitization method directly contributes to security enhancements in various applications of TFHE, such as post-quantum oblivious pseudorandom functions (OPRFs) and privacy-preserving machine learning (PPML). Specifically, Albrecht et al. [ADDG24] construct a post-quantum OPRF based on TFHE, where the sender evaluates the PRF on a user's input using its secret key. In this setting, circuit privacy is essential to protect the sender's key from leakage. Similarly, in PPML applications [CDPP22, BMMP18, LJ19] where a sender obliviously evaluates an inference model on a receiver's input, circuit privacy is critical for preserving the confidentiality of the model, which may reveal sensitive information about the training data. We note that by applying our sanitization algorithm to the output TFHE ciphertexts, the sender can address these privacy concerns in TFHE-based protocols. Furthermore, security can be strengthened against malicious adversaries by incorporating our ZKAoKs for TFHE.

1.2 Technical Overview

Oblivious Circuit Evaluation. To begin with, we formalize the goal of TFHE sanitization as the construction of an oblivious circuit evaluation (OCE) protocol. An OCE protocol abstracts a sender-receiver protocol, where the receiver holds a private input, the sender holds a private function, and the receiver learns the evaluation result while the sender learns nothing. This abstraction encompasses various cryptographic protocols, including oblivious transfer [Rab81], oblivious polynomial evaluation [NP06], and oblivious PRF [FIPR05]. We then formally prove that the existence of a sanitization algorithm implies the construction of a semi-honest secure OCE protocol. Furthermore, by incorporating ZKAoKs for ciphertexts and public keys, we show that the protocol can be extended to achieve security against malicious receivers.

New Sanitization Method. We then introduce a new sanitization method for TFHE that effectively addresses the semi-honest setting. As noted earlier, our approach minimizes the use of randomization within the TFHE bootstrapping procedure. The core idea is to incorporate an efficient ciphertext randomization technique from [BI22] at the beginning of bootstrapping and a circuit-private linear evaluation algorithm from [dCKK+24] at the end. As a result, the main body of the bootstrapping remains unchanged, leading to significant performance improvement.

ZKAoK for TFHE. Finally, to address malicious receivers, we design a ZKAoK for TFHE within the PIOP framework [CHM+20]. In particular, we leverage the efficient PIOPs for RLWE-based HE schemes proposed in [HLSS25]. To enable efficient proof generation, we adopt the packed public key generation technique from [KDE+23], which allows public keys to be generated in a compressed form.

This significantly reduces the overhead of verifying the well-formedness of public keys. In addition, we design a new OCE protocol optimized for TFHE that eliminates the need for ZKAoKs on ciphertexts. As a result, we succeed in constructing practical OCE protocols that are secure against malicious receivers.

2 Preliminaries

2.1 Notation

The set $\mathbb{B}$ is defined as $\{0,1\}$, representing the binary alphabet. For a positive integer q, we use the set $\mathbb{Z} \cap \left(-\frac{q}{2}, \frac{q}{2}\right]$ as a representative set for $\mathbb{Z}_q$, and denote the reduction of an integer a modulo q by $[a]_q$. Vectors over $\mathbb{Z}$ or $\mathbb{Z}_q$ are denoted by regular lowercase letters with arrows, such as $\vec{v}$, while matrices over $\mathbb{Z}$ or $\mathbb{Z}_q$ are represented by regular uppercase letters. For polynomials, we use bold lowercase letters to denote them e.g., $\boldsymbol{f}$. We treat all vectors as column vectors and use the symbol $\|$ to denote the concatenation of two vectors. The Hadamard product between vectors is denoted by $\circ$. For a positive definite matrix S, $\sqrt{S}$ denotes a matrix that satisfies $S = \sqrt{S}^\top \sqrt{S}$. For a vector $\vec{v} = (v_0, \ldots, v_{n-1}) \in \mathbb{Z}^n$, the L^p-norm ($p \geq 1$) and the L^∞-norm are defined as $\|\vec{v}\|_p := \sqrt[p]{\sum_{i=0}^{n-1} |v_i|^p}$ and $\|\vec{v}\|_\infty := \max_{0 \leq i < n} |v_i|$. For a polynomial $\boldsymbol{f}$, $\|\boldsymbol{f}\|_p$ is defined as the norm of the vector of its coefficients. For a matrix A, the matrix norm is defined as $\|A\|_2 := \max_{\vec{x} \neq \vec{0}} \frac{\|A\vec{x}\|_2}{\|\vec{x}\|_2}$.

Let N be a power of two. We define $R = \mathbb{Z}[X]/(X^N+1)$ as the ring of integers of the $2N$-th cyclotomic field, and $R_q = \mathbb{Z}_q[X]/(X^N+1)$ as the quotient ring of R modulo q. For a ring element $\boldsymbol{a} = \sum_{i=0}^{N-1} a_i X^i \in R$, its coefficient vector $\vec{a}$ and negacyclic matrix A are defined as

$$\vec{a} = (a_0, \ldots, a_{N-1}), \quad A = \begin{bmatrix} a_0 & -a_{N-1} & \cdots & -a_1 \\ a_1 & a_0 & \cdots & -a_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{N-1} & a_{N-2} & \cdots & a_0 \end{bmatrix}.$$

For a probability distribution $\mathcal{X}$, we denote sampling x from $\mathcal{X}$ as $x \leftarrow \mathcal{X}$. For sampling from a uniform distribution over a finite set S, we write $x \xleftarrow{\$} S$. For a polynomial $\boldsymbol{f}$ of degree d, we denote by $\boldsymbol{f} \leftarrow \mathcal{X}^d$ if each of its coefficients is sampled from $\mathcal{X}$. The statistical distance between two distributions $\mathcal{X}_1$ and $\mathcal{X}_2$ with the same support Ω is defined as $\Delta_{\mathrm{SD}}(\mathcal{X}_1, \mathcal{X}_2) := \frac{1}{2} \sum_{x \in \Omega} |\mathcal{X}_1(x) - \mathcal{X}_2(x)|$. We write $\mathcal{X}_1 \overset{\varepsilon}{\approx} \mathcal{X}_2$ when $\Delta_{\mathrm{SD}}(\mathcal{X}_1, \mathcal{X}_2) \leq \varepsilon$. We write $\mathcal{X}_1 \approx_s \mathcal{X}_2$ if two distributions are statistically indistinguishable, i.e., the statistical distance between them is negligible. Similarly, we write $\mathcal{X}_1 \approx_c \mathcal{X}_2$ if two distributions are computationally indistinguishable, i.e., no probabilistic polynomial-time (PPT) algorithm can distinguish between them.

2.2 Discrete Gaussian over Lattices

The n-dimensional Gaussian function $\rho_{\vec{c},\sqrt{\Sigma}}$ with center $\vec{c}$ and width parameter $\sqrt{\Sigma}$ is defined as $\rho_{\vec{c},\sqrt{\Sigma}}(\vec{x}) := \exp\left(-\pi(\vec{x}-\vec{c})^\top \Sigma^{-1}(\vec{x}-\vec{c})\right)$. We define the discrete Gaussian distribution over a coset of lattice $\vec{a}+\Lambda$ as $\mathcal{D}_{\vec{a}+\Lambda,\vec{c},\sqrt{\Sigma}}(\vec{x}) := \frac{\rho_{\vec{c},\sqrt{\Sigma}}(\vec{x})}{\sum_{\vec{v}\in\vec{a}+\Lambda}\rho_{\vec{c},\sqrt{\Sigma}}(\vec{v})}$, and the continuous Gaussian distribution over $\mathbb{R}^n$ as $\mathcal{D}_{\mathbb{R}^n,\vec{c},\sqrt{\Sigma}}(\vec{x}) := \frac{\rho_{\vec{c},\sqrt{\Sigma}}(\vec{x})}{\sqrt{\det\Sigma}}$. We also define the rounded Gaussian distribution as $\left\lfloor \mathcal{D}_{\mathbb{R}^n,\vec{c},\sqrt{\Sigma}} \right\rceil := \{\lfloor \vec{x} \rceil \mid \vec{x} \leftarrow \mathcal{D}_{\mathbb{R}^n,\vec{c},\sqrt{\Sigma}}\}$. We simplify the notation in special cases. When $\vec{c}=\vec{0}$, we omit $\vec{c}$. If $\Sigma = \sigma^2 I$ for some $\sigma > 0$, we denote σ instead of $\sqrt{\Sigma}$. For a polynomial $\boldsymbol{a}$ and its coefficient vector $\vec{a}$, we denote a coset of lattice $\vec{a}+\Lambda$ as $\boldsymbol{a}+\Lambda$. Below, we provide useful lemmas related to discrete Gaussian distributions.

Definition 1 ([MR07, Smoothing Parameter]). *For a n-dimensional lattice Λ and $\varepsilon > 0$, the smoothing parameter $\eta_\varepsilon(\Lambda)$ is defined as the smallest σ such that $\rho_{1/(\sigma\sqrt{2\pi})}(\Lambda^* \setminus \{\vec{0}\}) \le \varepsilon$. Let Σ be a positive-definite matrix. We say that $\sqrt{\Sigma} \ge \eta_\varepsilon(\Lambda)$ if $\eta_\varepsilon(\sqrt{\Sigma}^{-1}\cdot\Lambda) \le 1$, i.e., $\rho\left(\sqrt{\Sigma}^\top \cdot \Lambda^* \setminus \{\vec{0}\}\right) \le \varepsilon$.*

Lemma 1 ([MR07, Lemma 3.3]). *For a n-dimensional lattice Λ and $\varepsilon > 0$, it holds that $\eta_\varepsilon(\Lambda) \le \lambda_n(\Lambda)\cdot\sqrt{\frac{\ln(2n(1+1/\varepsilon))}{\pi}}$, where $\lambda_n(\Lambda)$ is the n-th successive minimum of Λ.*

Lemma 2 ([Pei10, Theorem 3.1]). *Let Σ_1, Σ_2 be positive definite matrices, and let $\Sigma^{-1} := \Sigma_1^{-1} + \Sigma_2^{-1}$. If $\eta_\varepsilon(\mathbb{Z}^n) \le \sqrt{\Sigma}$ for $0 < \varepsilon < 1$, then for any $\vec{c}_1, \vec{c}_2 \in \mathbb{R}^n$,*

$$\Delta_{\mathrm{SD}}\left(\mathcal{D}_{\mathbb{Z}^n,\vec{c}_1,\sqrt{\Sigma_1}} + \mathcal{D}_{\mathbb{Z}^n,\vec{c}_2,\sqrt{\Sigma_2}},\ \mathcal{D}_{\mathbb{Z}^n,\vec{c}_1+\vec{c}_2,\sqrt{\Sigma_1+\Sigma_2}}\right) \le 2\varepsilon.$$

Lemma 3 ([dCKK+24, Lemma 6]). *Let $A \in \mathbb{Z}^{m\times n}$ be a matrix, q be a positive integer, and $\sigma_1, \sigma_2 > 0$ be real numbers such that $\frac{1}{\sigma_1^2} + \frac{\|A\|_2^2}{\sigma_2^2} \le \frac{1}{\eta_\varepsilon(q\mathbb{Z}^n)^2}$ for $0 < \varepsilon \le 1/2$. Then, for any $\vec{c} \in \mathbb{Z}^n$,*

$$\Delta_{\mathrm{SD}}\left(A\cdot\mathcal{D}_{\vec{c}+q\mathbb{Z}^n,\sigma_1} + \mathcal{D}_{\mathbb{Z}^m,\sigma_2},\ \mathcal{D}_{\mathbb{Z}^m,\sqrt{\sigma_1^2 AA^\top + \sigma_2^2 I}}\right) \le 4\varepsilon.$$

2.3 LWE and RLWE

The security of lattice-based homomorphic encryption schemes relies on the hardness of the LWE problem [Reg09] and its ring variant, the RLWE problem [LPR10, SSTX09], which are defined as follows.

Definition 2 (LWE). *Let m, n, q be positive integers, and χ_s, χ_e be distributions over $\mathbb{Z}$. The decisional LWE problem $\mathsf{LWE}^{n,q}_{\chi_s,\chi_e}$ is to distinguish between the following two distributions: $\{(\vec{a}, \langle\vec{a},\vec{s}\rangle + e) \mid \vec{a} \xleftarrow{\$} \mathbb{Z}_q^n, \vec{s} \leftarrow \chi_s^n, e \leftarrow \chi_e\}$ and $\{(\vec{a}, u) \mid \vec{a} \xleftarrow{\$} \mathbb{Z}_q^n, u \xleftarrow{\$} \mathbb{Z}_q\}$. If χ_s or χ_e follow the discrete Gaussian $\mathcal{D}_{\mathbb{Z},\sigma}$, we simply denote them as σ in the subscript.*

Definition 3 (Ring-LWE). *Let $R = \mathbb{Z}[X]/(X^N + 1)$, q be a positive integer, and χ_s, χ_e be distributions over R. The decisional RLWE problem $\mathsf{RLWE}^{R,q}_{\chi_s,\chi_e}$ is to distinguish the following two distributions: $\{(\boldsymbol{a}, \boldsymbol{as} + \boldsymbol{e}) \mid \boldsymbol{a} \xleftarrow{\$} R_q, \boldsymbol{s} \leftarrow \chi_s, \boldsymbol{e} \leftarrow \chi_e\}$ and $\{(\boldsymbol{a}, \boldsymbol{u}) \mid \boldsymbol{a}, \boldsymbol{u} \xleftarrow{\$} R_q\}$. If χ_s or χ_e follow the discrete Gaussian distribution $\mathcal{D}^N_{\mathbb{Z},\sigma}$, we simply denote them as σ in the subscript.*

Under these hardness assumptions, encryption schemes based on LWE and RLWE are constructed. An LWE-based ciphertext has the form $(b, \vec{a}) \in \mathbb{Z}_q^{n+1}$, and its phase with respect to $\vec{s} \in \mathbb{Z}^n$ is defined as $\varphi_{\vec{s}}(b, \vec{a}) := b + \langle \vec{a}, \vec{s} \rangle$. Similarly, an RLWE-based ciphertext is of the form $(\boldsymbol{b}, \boldsymbol{a}) \in R_q^2$, and its phase with respect to $\boldsymbol{s} \in R$ is defined as $\varphi_{\boldsymbol{s}}(\boldsymbol{b}, \boldsymbol{a}) := \boldsymbol{b} + \boldsymbol{as}$. For both LWE and RLWE ciphertexts, we call the first component the *body* of the ciphertext, and the remaining components the *mask* of the ciphertext.

2.4 Gadget Toolkit

We review the definition of the gadget toolkit, a widely used technique in lattice-based cryptography [MP12, CGGI16].

Definition 4 (Gadget Toolkit). *A vector $\vec{g} \in \mathbb{Z}^d$ is called a* gadget vector*, and a map $h : \mathbb{Z}_q \to \mathbb{Z}^d$ is called a* gadget decomposition *corresponding to $\vec{g}$ with accuracy parameters (ε, δ) if h satisfies $\|h(a)\|_\infty \leq \delta$ and $|\langle h(a), \vec{g} \rangle - a| \leq \varepsilon$ for any $a \in \mathbb{Z}_q$. As an abuse of notation, we often denote $h : R_q^k \to R^{kd}$ for some integer k, where each coefficient of the input polynomials is decomposed using the gadget decomposition $h : \mathbb{Z}_q \to \mathbb{Z}^d$.*

Note that the gadget toolkit can be easily extended to the elements over the ring R_q, by applying the gadget decomposition to each coefficient of the input ring element. For the gadget decomposition over the ring, we will abuse the same notation as it over the integers. A simple example of gadget decomposition is the *approximate digit decomposition*. For integers $B, d > 0$ such that $B^d < q$ and a vector $\vec{g} = (\lceil q/B^d \rceil \cdot B^i)_{0 \leq i < d}$, there exists a decomposition function h such that the accuracy parameters are given by $\delta = \frac{B}{2}$ and $\varepsilon = \frac{q}{2B^d}$.

2.5 Interactive Argument of Knowledge

We define an interactive argument of knowledge and its zero-knowledge version as follows.

Definition 5 (Interactive Argument of Knowledge). *Let $\Pi = (\mathtt{Setup}, \mathtt{P}, \mathtt{V})$ be an interactive protocol between a prover $\mathtt{P}$ and a verifier $\mathtt{V}$. Π is called an argument of knowledge for a relation R if it satisfies the following properties.*

Completeness. *For all PPT adversaries $\mathtt{A}$, the following holds.*

$$\Pr\left[\begin{array}{r|l} \langle \mathtt{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w}), \mathtt{V}(\mathsf{pp}, \mathsf{x}) \rangle = 1 \vee & \mathsf{pp} \leftarrow \mathtt{Setup}(1^\lambda) \\ (\mathsf{x}, \mathsf{w}) \notin \mathsf{R} & (\mathsf{x}, \mathsf{w}) \leftarrow \mathtt{A}(\mathsf{pp}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Soundness. *For every PPT adversary* $\mathtt{A} = (\mathtt{A}_1, \mathtt{A}_2)$*, the following holds.*

$$\Pr\left[\begin{array}{r|l} \langle \mathtt{A}_2(\mathsf{pp}, \mathsf{st}, \mathsf{x}), \mathtt{V}(\mathsf{pp}, \mathsf{x})\rangle = 1 \wedge & \mathsf{pp} \leftarrow \mathtt{Setup}(1^\lambda) \\ \mathsf{x} \notin \mathsf{L}(\mathsf{R}) & (\mathsf{st}, \mathsf{x}) \leftarrow \mathtt{A}_1(\mathsf{pp}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Knowledge Soundness. *For every PPT adversary* $\mathtt{A} = (\mathtt{A}_1, \mathtt{A}_2)$*, there exists a PPT extractor* $\mathtt{E}$ *such that, given oracle access to* $\mathtt{A}$*, the following holds.*

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

Π is called a zero-knowledge argument of knowledge if it additionally satisfies the following.

Honest Verifier Zero-knowledge. *For every PPT adversary* $\mathtt{A} = (\mathtt{A}_1, \mathtt{A}_2)$*, there exists a PPT simulator* $\mathtt{S}$ *such that the following holds, where* $\mathtt{View}$ *outputs the verifier's view.*

$$\left|\Pr\left[\begin{array}{r|l} \mathtt{A}_2(\mathsf{ck}, \mathsf{view}) = 1 \wedge & \mathsf{pp} \leftarrow \mathtt{Setup}(1^\lambda) \\ (\mathsf{x}, \mathsf{w}) \in \mathsf{R} & (\mathsf{x}, \mathsf{w}) \leftarrow \mathtt{A}_1(\mathsf{pp}) \\ & \mathsf{view} \leftarrow \mathtt{S}(\mathsf{pp}, \mathsf{x}) \end{array}\right]\right.$$
$$\left. - \Pr\left[\begin{array}{r|l} \mathtt{A}_2(\mathsf{ck}, \mathsf{view}) = 1 \wedge & \mathsf{pp} \leftarrow \mathtt{Setup}(1^\lambda) \\ (\mathsf{x}, \mathsf{w}) \in \mathsf{R} & (\mathsf{x}, \mathsf{w}) \leftarrow \mathtt{A}_1(\mathsf{pp}) \\ & \mathsf{view} \leftarrow \mathtt{View}\big(\mathtt{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w}), \mathtt{V}(\mathsf{pp}, \mathsf{x})\big) \end{array}\right]\right| \leq \mathsf{negl}(\lambda)$$

Finally, Π is called public coin if all messages from the honest verifier can be computed as a deterministic function of a random public input.

Next, we review the definition of a polynomial interactive oracle proof (PIOP) in [CHM+20], which is a special class of interactive oracle proof [BSCS16]. In the following definition, we restrict polynomials to be univariate, but it can be generalized to the multivariate case, as defined in [CBBZ23].

Definition 6 (Polynomial Interactive Oracle Proof). *Let* $\Pi = (\mathtt{Setup}, \mathtt{P}, \mathtt{V})$ *be an interactive oracle proof system for a relation* R*, and let* $\mathbb{F}$ *be a finite field. Π is called a polynomial interactive oracle proof (PIOP), which satisfies the following.*

- *Every message from the prover is a polynomial oracle* $([\![\boldsymbol{f}]\!], d)$*, where* $\boldsymbol{f} \in \mathbb{F}[X]$ *of degree* $< d$*.*
- *Every message from the verifier is a random challenge.*
- *The verifier has oracle access to evaluations of the prover's polynomials at arbitrary points.*

Π is called a zero-knowledge PIOP if it additionally satisfies the honest verifier zero-knowledge property.

A PIOP can be compiled into an interactive argument of knowledge with the aid of a polynomial commitment scheme. This framework enables a modular approach to designing proof systems: one first conceptually designs a proof system in PIOP form, and then concretely instantiates it using a polynomial commitment scheme. We refer to [CHM+20] for the details of PIOP compilation.

2.6 Homomorphic Encryption

We refine the definition of homomorphic encryption (HE) by incorporating a well-formedness criterion into its correctness conditions. In particular, we require that the well-formedness criterion be formulated as an NP language, enabling efficient verification via zero-knowledge arguments of knowledge. The refined definitions are provided below.

Definition 7 (Homomorphic Encryption). *A homomorphic encryption scheme is a tuple of PPT algorithms (*$\mathtt{Setup}$*,* $\mathtt{KeyGen}$*,* $\mathtt{Enc}$*,* $\mathtt{Dec}$*,* $\mathtt{Eval}$*) which satisfies the following.*

- $\underline{\mathtt{Setup}}(1^\lambda) \to \mathsf{pp}$*: Given a security parameter* λ*, output a public parameter* pp*, and define NP languages* L_{pk} *and* L_{ct} *for well-formed public keys and ciphertexts.*
- $\underline{\mathtt{KeyGen}}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$*: Given a public parameter* pp*, output a secret key* sk *and a evaluation key* pk*.*
- $\underline{\mathtt{Enc}}(\mathsf{ek}, \mathsf{pt}) \to \mathsf{ct}$*: Given an encryption key* $\mathsf{ek} \in \mathsf{pk}$ *and a plaintext* pt*, output a ciphertext* ct*.*
- $\underline{\mathtt{Dec}}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{pt}$*: Given a secret key* sk *and a ciphertext* ct*, output a plaintext* pt*.*
- $\underline{\mathtt{Eval}}(\mathsf{pk}, C, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}) \to \mathsf{ct}_{\mathsf{out}}$*: Give a evaluation key* pk*, a circuit* C *and ciphertexts* $\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}$*, output a ciphertext* $\mathsf{ct}_{\mathsf{out}}$*.*

Correctness. *For* $\mathsf{pp} \leftarrow \mathtt{Setup}(1^\lambda)$ *and* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{Setup}(\mathsf{pp})$*, the following properties hold with overwhelming probability in* λ*.*

- *For the evaluation key* pk*, it holds that* $\mathsf{pk} \in \mathsf{L}_{\mathsf{pk}}$*.*
- *For any plaintext* pt*, it holds that* $\mathsf{ct} \in \mathsf{L}_{\mathsf{ct}}$ *and* $\mathsf{pt} = \mathtt{Dec}(\mathsf{sk}, \mathsf{ct})$*, where* $\mathsf{ct} \leftarrow \mathtt{Enc}(\mathsf{pk}, \mathsf{pt})$*.*
- *For any* ℓ*-ary circuit* C*, it holds that* $\mathsf{ct}_{\mathsf{out}} \in \mathsf{L}_{\mathsf{ct}}$ *and* $\mathtt{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{out}}) = C(\mathsf{pt}_0, \ldots, \mathsf{pt}_{\ell-1})$*, where* $\mathsf{pk} \in \mathsf{L}_{\mathsf{pk}}$*,* $\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1} \in \mathsf{L}_{\mathsf{ct}}$*,* $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathtt{Eval}(\mathsf{pk}, C, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})$*, and* $\mathsf{pt}_i = \mathtt{Dec}(\mathsf{sk}, \mathsf{ct}_i)$ *for* $0 \le i < \ell$*.*

Security. *The encryption scheme* $(\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ *satisfies semantic security.*

3 Oblivious Circuit Evaluation from HE

In this section, we formalize the functionality and security of oblivious circuit evaluation protocols, which we aim to achieve through homomorphic encryption

(HE) schemes, and revisit several well-known results within the framework of oblivious circuit evaluation.

We first review the concept of circuit-private HE schemes and the trivial instantiation of an oblivious circuit evaluation protocol in the semi-honest model using circuit-private HE. Then, we review the concept of ciphertext sanitization and its implications for circuit-private homomorphic encryption. Finally, we show how to upgrade its security against malicious receivers by incorporating zero-knowledge arguments of knowledge (ZKAoK).

As a result, we reduce the problem of constructing an oblivious circuit evaluation protocol from HE to the construction of a ciphertext sanitization method and a ZKAoK for HE schemes.

3.1 Oblivious Circuit Evaluation

Oblivious evaluation of a sender's private function on a receiver's private input is a frequently appearing concept in various cryptographic and privacy-preserving protocols. For example, classical topics such as oblivious transfer [Rab81], oblivious polynomial evaluation [NP06], oblivious PRF [FIPR05], and oblivious neural network evaluation [CL01] all fall under this concept. Below, we generalize the notion of oblivious evaluation as the oblivious circuit evaluation functionality in Fig. 1 and define its security within the UC framework [Can01], assuming that the sender evaluates its private function in the form of a circuit.

Definition 8. *Let Π be a sender-receiver protocol. Π is said to be a secure oblivious circuit evaluation protocol against an adversary* $\mathtt{A}$ *if there exists a PPT simulator* $\mathtt{S}$ *such that* $\mathtt{EXEC}[\Pi, \mathtt{A}, \mathtt{Z}] \approx_c \mathtt{EXEC}[\mathsf{F}_{\mathsf{OCE}}, \mathtt{S}, \mathtt{Z}]$ *for all PPT environments* $\mathtt{Z}$.

$\mathsf{F}_{\mathsf{OCE}}$

$\mathsf{F}_{\mathsf{OCE}}$ runs with a sender and a receiver.

Input: The receiver has inputs $\mathsf{pt}_0, \ldots, \mathsf{pt}_{\ell-1}$, and the sender has a circuit C to be evaluated.

Output: The receiver outputs $\mathsf{pt}_{\mathsf{out}} = C(\mathsf{pt}_0, \ldots, \mathsf{pt}_{\ell-1})$, and the sender outputs nothing.

Fig. 1. Functionality of Oblivious Circuit Evaluation

3.2 Homomorphic Encryption based OCE

For the construction of OCE protocols in the semi-honest setting, homomorphic encryption (HE) appears to offer a trivial solution at first glance: the receiver first sends an encryption of its input using HE, the sender homomorphically

evaluates its circuit on the ciphertexts and returns the output ciphertexts to the receiver, and finally, the receiver obtains the result by decrypting it. However, the functionality of HE alone does not guarantee the security of this protocol, even in the semi-honest setting. The main problem arises from the output ciphertexts that the sender returns, as the ciphertext itself may leak partial information about the sender's circuit.

Circuit Privacy. To address this issue, the concept of circuit-private HE has been utilized since the first introduction of HE in [Gen09]. In a nutshell, it formalizes a sufficient condition for HE to imply secure oblivious circuit evaluation, i.e., a homomorphic evaluation algorithm does not leak any information about the circuit beyond the output. Below, we provide its formal definition.

Definition 9 ([Gen09, Circuit Privacy]). *A homomorphic encryption scheme is said to be circuit private for a class of circuits $\mathcal{C}$ if there exists a PPT algorithm* $\mathtt{Sim}$ *such that for any circuit $C \in \mathcal{C}$, any ciphertexts $\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1} \in \mathsf{L}_{\mathsf{ct}}$, and any evaluation key $\mathsf{pk} \in \mathsf{L}_{\mathsf{pk}}$, it holds that*

$$\begin{aligned}&\{(\mathsf{sk}, \mathsf{pk}, \mathsf{ct}_{\mathsf{out}}) \mid \mathsf{ct}_{\mathsf{out}} \leftarrow \mathtt{Eval}(\mathsf{pk}, C, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})\} \\ &\quad \approx_c \{(\mathsf{sk}, \mathsf{pk}, \mathsf{ct}'_{\mathsf{out}}) \mid \mathsf{ct}'_{\mathsf{out}} \leftarrow \mathtt{Sim}(\mathsf{pk}, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}, \mathsf{pt}_{\mathsf{out}})\}\end{aligned}$$

where $\mathsf{pt}_i \leftarrow \mathtt{Dec}(\mathsf{sk}, \mathsf{ct}_i)$ for $0 \le i < \ell$ and $\mathsf{pt}_{\mathsf{out}} = C(\mathsf{pt}_0, \ldots, \mathsf{pt}_{\ell-1})$.

Below, we prove that the circuit privacy condition implies a secure OCE protocol in the semi-honest setting.

Theorem 1. *Let Π_{OCE} be a sender-receiver protocol described in Fig. 2, and let* $(\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$ *be a circuit-private homomorphic encryption scheme. Then, Π_{OCE} is a secure OCE protocol against semi-honest senders and receivers.*

Π_{OCE}

$\Pi^{\mathsf{passive}}_{\mathsf{OCE}}$ runs with a sender and a receiver.

Input: The receiver has inputs $\mathsf{pt}_0, \ldots, \mathsf{pt}_{\ell-1}$, and the sender has a circuit C to be evaluated.

Output: The receiver outputs $\mathsf{pt}_{\mathsf{out}} \leftarrow \mathtt{Dec}(\mathsf{sk}, \overline{\mathsf{ct}}_{\mathsf{out}})$, and the sender outputs nothing.

Setup:

1. Both the sender and the receiver generate a public parameter $\mathsf{pp} \leftarrow \mathtt{Setup}(1^\lambda)$.
2. The receiver generates $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{KeyGen}(\mathsf{pp})$, and sends pk to the sender.

Protocol:

1. The receiver generates $\mathsf{ct}_i \leftarrow \mathtt{Enc}(\mathsf{pk}, \mathsf{pt}_i)$ for $0 \le i < \ell$. The receiver then sends $(\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})$ to the sender.

2. The sender computes $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathtt{Eval}(\mathsf{pk}, C, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})$ and sends $\mathsf{ct}_{\mathsf{out}}$ to the receiver.

Fig. 2. Secure HE-based OCE protocol against semi-honest adversaries

Ciphertext Sanitization. Since circuit privacy is a core concept in HE-based two-party computation, several approaches [Gen09, BDPMW16, DS16] have been proposed to achieve circuit privacy from non-circuit-private HE schemes. One general approach is to design a sanitization algorithm, which removes any meaningful information from ciphertexts except for their underlying plaintexts, as defined below.

Definition 10 ([DS16, Sanitization]). *A PPT algorithm* $\mathtt{Sanitize}$ *is called a sanitization for ciphertexts in* L_{ct} *if the following properties hold with overwhelming probability for* $\mathsf{ct} \in \mathsf{L}_{\mathsf{ct}}$ *and* $\mathsf{pk} \in \mathsf{L}_{\mathsf{pk}}$.

Correctness. *For* $\overline{\mathsf{ct}} \leftarrow \mathtt{Sanitize}(\mathsf{pk}, \mathsf{ct})$*, it holds that* $\mathtt{Dec}(\mathsf{sk}, \overline{\mathsf{ct}}) = \mathtt{Dec}(\mathsf{sk}, \mathsf{ct})$.

Security. *For any* $\mathsf{ct}' \in \mathsf{L}_{\mathsf{ct}}$ *such that* $\mathtt{Dec}(\mathsf{sk}, \mathsf{ct}) = \mathtt{Dec}(\mathsf{sk}, \mathsf{ct}')$*, it holds that*

$$\{(\mathsf{sk}, \mathsf{pk}, \overline{\mathsf{ct}}) \mid \overline{\mathsf{ct}} \leftarrow \mathtt{Sanitize}(\mathsf{pk}, \mathsf{ct})\}$$
$$\approx_c \{(\mathsf{sk}, \mathsf{pk}, \overline{\mathsf{ct}}') \mid \overline{\mathsf{ct}}' \leftarrow \mathtt{Sanitize}(\mathsf{pk}, \mathsf{ct}')\}.$$

As described above, any two sanitizable ciphertexts in L_{ct} that decrypt to the same plaintext become indistinguishable after sanitization, implying that the output of sanitization no longer contains meaningful information beyond the plaintext. Then, the existence of a sanitization algorithm implies circuit privacy, as shown below.

Theorem 2. *Let* $(\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$ *be a homomorphic encryption scheme with a sanitization algorithm* $\mathtt{Sanitize}$*. Then,* $(\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval}')$ *is a circuit-private homomorphic encryption scheme, where*

$$\mathtt{Eval}'(\mathsf{pk}, C, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}) := \mathtt{Sanitize}\Big(\mathsf{pk}, \mathtt{Eval}(\mathsf{pk}, C, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})\Big)$$

for any circuit C *such that* $\mathtt{Eval}(\mathsf{pk}, C, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}) \in \mathsf{L}_{\mathsf{ct}}$ *if* $\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1} \in \mathsf{L}_{\mathsf{ct}}$.

Malicious Threats and ZKAoK. Although circuit-private HE provides a simple solution for OCE, its security is guaranteed only in the semi-honest setting, where the corrupted sender or receiver is assumed to honestly follow the protocol. This is explicitly described in Definition 9, where it is assumed that the public key and input ciphertexts are well-formed, i.e., , they belong to the NP languages L_{pk} and L_{ct}.

However, in the malicious setting, there is no guarantee that a malicious receiver sends a well-formed public key and ciphertexts to the sender. This issue can be addressed with the aid of a zero-knowledge argument of knowledge (ZKAoK), whose ideal functionality is described below.

$\mathsf{F}_{\mathsf{ZKAoK}}(\mathsf{R})$

$\mathsf{F}_{\mathsf{ZKAoK}}$ runs with a prover and a verifier.

Input: The prover has an input (x, w), and the verifier has an input x.

Output: The verifier outputs 1 if $(\mathsf{x}, \mathsf{w}) \in \mathsf{R}$, otherwise, it outputs 0. The prover outputs nothing.

Fig. 3. Functionality of zero-knowledge argument of knowledge

The purpose of ZKAoK is to enforce a malicious sender to send a well-formed public key and well-formed ciphertexts, thereby ensuring circuit privacy even against a malicious receiver. Thus, incorporating ZKAoK together with circuit-private HE results in secure OCE protocols against malicious receivers, as described below.

Theorem 3. *Let $\Pi_{\mathsf{OCE}}^{\mathsf{ZKAoK}}$ be a sender-receiver protocol described in Fig. 4, and let* $(\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$ *be a circuit-private homomorphic encryption scheme. Then, $\Pi_{\mathsf{OCE}}^{\mathsf{ZKAoK}}$ is a secure OCE protocol against semi-honest senders and malicious receivers in the $\mathsf{F}_{\mathsf{ZKAoK}}(\mathsf{R}_{\mathsf{pk}})$, $\mathsf{F}_{\mathsf{ZKAoK}}(\mathsf{R}_{\mathsf{ct}})$-hybrid model.*

$\Pi_{\mathsf{OCE}}^{\mathsf{ZKAoK}}$

$\Pi_{\mathsf{OCE}}^{\mathsf{active}}$ runs with a sender and a receiver.

Input: The receiver has inputs $\mathsf{pt}_0, \ldots, \mathsf{pt}_{\ell-1}$, and the sender has a circuit C to be evaluated.

Output: The receiver obtains $\mathsf{pt}_{\mathsf{out}} \leftarrow \mathtt{Dec}(\mathsf{sk}, \overline{\mathsf{ct}}_{\mathsf{out}})$, and the sender obtains nothing.

Setup:

1. Both the sender and the receiver generate a public parameter $\mathsf{pp} \leftarrow \mathtt{Setup}(1^\lambda)$.
2. The receiver generates $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{KeyGen}(\mathsf{pp})$, and sends pk to the sender.
3. The receiver and the sender runs $\mathsf{F}_{\mathsf{ZKAoK}}(\mathsf{R}_{\mathsf{pk}})$, acting as a prover and a verifier. The sender aborts if its output is 0.

Protocol:

1. The receiver generates $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{KeyGen}(\mathsf{pp})$ and computes $\mathsf{ct}_i \leftarrow \mathtt{Enc}(\mathsf{pk}, \mathsf{pt}_i)$ for $0 \le i < \ell$. The receiver then sends $(\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})$ to the sender.
2. The receiver and the sender runs $\mathsf{F}_{\mathsf{ZKAoK}}(\mathsf{R}_{\mathsf{ct}})$, acting as a prover and a verifier. The sender aborts if its output is 0.
3. The sender computes $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathtt{Eval}(\mathsf{pk}, C, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})$ and sends $\mathsf{ct}_{\mathsf{out}}$ to the receiver.

Fig. 4. Secure HE-based OCE protocol against malicious receivers

The above protocol only ensures security against semi-honest senders. However, we note that considering a semi-honest sender is often sufficient in HE-based OCE protocols, because a sender cannot affect the generation of the public key or input ciphertexts. Thus, a malicious sender can at most attempt to damage correctness, while the receiver's input remains protected by the semantic security of HE. Therefore, in constructing HE-based OCE protocols, we primarily focus on the malicious receiver case.

4 New Sanitization Method for TFHE

In this section, we propose a new sanitization method for the TFHE scheme [CGGI16] to achieve circuit privacy. Previously, several approaches [DS16, BI22, Klu24] have been proposed for sanitizing TFHE ciphertexts. Although their core idea is similar in that they incorporate randomization during TFHE bootstrapping, the details differ, particularly regarding when and how the randomization is applied. For example, the sanitization method in [DS16] adds small noise at the end of bootstrapping. However, due to the limited entropy growth, it requires multiple repetitions of this process. In comparison, the sanitization methods in [BI22, Klu24] randomize the entire bootstrapping procedure. As a result, they achieve sanitization with a single bootstrapping but suffer from degraded performance due to heavy randomization.

Our sanitization method is also based on TFHE bootstrapping, but it introduces randomization only at the beginning and end of the bootstrapping procedure. Compared to previous approaches, our method requires only a single bootstrapping and avoids heavy randomization, effectively combining the advantages of both approaches and significantly improving performance. In the remainder of this section, we first review the TFHE scheme, then present our new sanitization method, and finally provide a detailed performance analysis compared to prior approaches.

4.1 The TFHE Scheme

We review the TFHE scheme [CGGI16], incorporating the adaptation in [BMMP18]. We note that the encryption algorithm is defined as symmetric encryption, following the convention in TFHE, but it can be extended to asymmetric encryption as presented in [Joy24].

Basic Algorithms.

– $\underline{\mathtt{Setup}}(1^\lambda) \to \mathsf{pp}$: Given a security parameter λ, generate a public parameter $\mathsf{pp} = (q, t, n, \chi, N, \boldsymbol{\chi}, \vec{g}_{\mathsf{brk}}, h_{\mathsf{brk}}, \vec{g}_{\mathsf{ksk}}, h_{\mathsf{ksk}})$, where q is the ciphertext modulus, t is the plaintext modulus such that $t \mid q$, n is the temporal LWE dimension, χ is the error distribution for LWE ciphertexts, N is the RLWE ring dimension, $\boldsymbol{\chi}$

is the error distribution for RLWE ciphertexts, $(\vec{g}_{\mathsf{brk}}, h_{\mathsf{brk}})$ is the gadget toolkit for bootstrapping keys with $\vec{g}_{\mathsf{brk}} \in \mathbb{Z}^{d_{\mathsf{brk}}}$, $(\vec{g}_{\mathsf{ksk}}, h_{\mathsf{ksk}})$ is the gadget toolkit for bootstrapping keys with $\vec{g}_{\mathsf{ksk}} \in \mathbb{Z}^{d_{\mathsf{ksk}}}$,

- $\underline{\mathtt{KeyGen}}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$: Given a public parameter pp, generate a key pair $(\mathsf{sk}, \mathsf{pk})$ as follows, where $\mathsf{pk} = (\mathsf{brk}, \mathsf{ksk}, \mathsf{ek})$.
  1. Sample $\boldsymbol{s} = \sum_{i=0}^{N-1} s_i \cdot X^i \xleftarrow{\$} \mathbb{B}^N$, $\vec{s}' = (s'_0, \ldots, s'_{n-1}) \xleftarrow{\$} \mathbb{B}^n$, and set $\mathsf{sk} = \boldsymbol{s}$.
  2. Sample $\vec{\boldsymbol{a}}_{\mathsf{brk},i,0}, \vec{\boldsymbol{a}}_{\mathsf{brk},i,1} \xleftarrow{\$} R_q^{d_{\mathsf{brk}}}$, $\vec{\boldsymbol{e}}_{\mathsf{brk},i,0}, \vec{\boldsymbol{e}}_{\mathsf{brk},i,1} \leftarrow \boldsymbol{\chi}^{d_{\mathsf{brk}}}$, then compute $\mathsf{brk}_{i,0} = (-\boldsymbol{s} \cdot \vec{\boldsymbol{a}}_{\mathsf{brk},i,0} + s'_i \cdot \vec{g}_{\mathsf{brk}} + \vec{\boldsymbol{e}}_{\mathsf{brk},i,0},\ \vec{\boldsymbol{a}}_{\mathsf{brk},i,0}) \in R_q^{2d_{\mathsf{brk}}}$, $\mathsf{brk}_{i,1} = (-\boldsymbol{s} \cdot \vec{\boldsymbol{a}}_{i,1} + s'_i \boldsymbol{s} \cdot \vec{g}_{\mathsf{brk}} + \vec{\boldsymbol{e}}_{i,1},\ \vec{\boldsymbol{a}}_{i,1}) \in R_q^{2d_{\mathsf{brk}}}$, and set the blind rotation key $\mathsf{brk} = \{(\mathsf{brk}_{i,0}, \mathsf{brk}_{i,1})\}_{0 \le i < n}$
  3. Sample $A_i \xleftarrow{\$} \mathbb{Z}_q^{d \times n}$, $\vec{e}_{\mathsf{ksk},i} \leftarrow \chi^{d_{\mathsf{ksk}}}$, then compute $\mathsf{ksk}_i = \left[-A_i \vec{s}' + s_i \cdot \vec{g}_{\mathsf{ksk}} + \vec{e}_{\mathsf{ksk},i} \middle| A_i \right]^\top \in \mathbb{Z}_q^{d_{\mathsf{ksk}} \times (n+1)}$ for $0 \le i < N$, and set $\mathsf{ksk} = \{\mathsf{ksk}_i\}_{0 \le i < N}$.
  4. Sample $\boldsymbol{a}_{\mathsf{ek}} \xleftarrow{\$} R_q$, $\boldsymbol{e}_{\mathsf{ek}} \leftarrow \boldsymbol{\chi}$, and set $\mathsf{ek} = (-\boldsymbol{a}_{\mathsf{ek}} \boldsymbol{s} + \boldsymbol{e}_{\mathsf{ek}}, \boldsymbol{a}_{\mathsf{ek}}) \in R_q^2$.
- $\underline{\mathtt{Enc}}(\mathsf{sk}, m) \to \vec{c}$: Given a secret key sk and a message $m \in \mathbb{Z}_t$, sample $\vec{a} \xleftarrow{\$} \mathbb{Z}_q^N$, $e \leftarrow \chi$, and return an LWE ciphertext $\vec{c} = (-\langle \vec{a}, \vec{s} \rangle + q/t \cdot m + e, \vec{a}) \in \mathbb{Z}_q^{N+1}$, where $\vec{s}$ is the coefficient vector of sk.
- $\underline{\mathtt{Dec}}(\mathsf{sk}, \vec{c}) \to m$: Given a secret key sk and an LWE ciphertext $\vec{c} \in \mathbb{Z}_q^{N+1}$, return $m = \lfloor \frac{t}{q} \langle \vec{c}, (1, \vec{s}) \rangle \rceil \pmod t$, where $\vec{s}$ is the coefficient vector of sk.

Derived from the above algorithms, the relation R_{pk} for well-formed public keys and the relation R_{ct} for well-formed ciphertexts are defined in Figs. 10 and 11. These relations primarily verify whether the keys and ciphertexts are generated from valid RLWE or LWE samples, ensuring that the secret key has binary coefficients and the error terms have small norms.

Helper Algorithms for Bootstrapping. Next, we review the helper algorithms used in TFHE bootstrapping, defined as follows.

- $\underline{\mathtt{KeySwitch}}(\mathsf{ksk}, \vec{c}) \to \vec{c}_{\mathsf{ks}}$: Given a key-switching key ksk and an LWE ciphertext $\vec{c} = (b, a_0, \ldots, a_{N-1}) \in \mathbb{Z}_q^{N+1}$, return an LWE ciphertext $\vec{c}_{\mathsf{ks}} = (b, 0) + \sum_{i=0}^{N-1} \mathsf{ksk}_i^\top \cdot h_{\mathsf{ksk}}(a_i) \in \mathbb{Z}_q^{n+1}$.
- $\underline{\mathtt{BlindRotate}}(\mathsf{brk}, f, \vec{c})$: Given a blind rotation key brk, a negacyclic function $f : \mathbb{Z}_t \to \mathbb{Z}_t$, and LWE ciphertext $\vec{c} \in \mathbb{Z}_q^{n+1}$, run Algorithm 1, and return the output RLWE ciphertext $\mathsf{ct}_{\mathsf{br}} \in R_q^2$.
- $\underline{\mathtt{SampleExtract}}(\mathsf{ct}) \to \vec{c}$: Given an RLWE ciphertext $\mathsf{ct} = (\sum_{i=0}^{N-1} b_i X^i, \sum_{i=0}^{N} a_i X^i)$, return an LWE ciphertext $\vec{c} = (b_0, a_0, -a_{N-1}, \ldots, -a_1) \in \mathbb{Z}_q^{N+1}$.

We briefly summarize the functionality of each algorithm. The key-switching algorithm reduces the dimension of an input LWE ciphertext from N to n while preserving the plaintext. This reduction decreases the computational cost of blind rotation from $O(N)$ to $O(n)$ during bootstrapping. The blind rotation algorithm takes an LWE ciphertext $\vec{c}$ and a negacyclic function $f : \mathbb{Z}_t \to \mathbb{Z}_t$, satisfying $f(x + \lfloor t/2 \rfloor) = -f(x)$, and outputs an RLWE ciphertext $\mathsf{ct}_{\mathsf{br}}$ whose

Algorithm 1 Blind Rotation

Input: Blind rotation key brk, negacyclic function f, LWE ciphertext $\vec{c}$
Output: RLWE ciphertext $\mathsf{ct}_{\mathsf{br}}$

1: $\vec{c}_{\mathsf{ms}} = (b_{\mathsf{ms}}, \vec{a}_{\mathsf{ms}}) := \lfloor \frac{2N}{q} \cdot \vec{c} \rceil \in \mathbb{Z}_{2N}^{n+1}$.
2: $\mathsf{tv} := (q/t) \cdot \sum_{j=1-N/t}^{(t-1)N/t} f(\lfloor \frac{t}{2N} j \rceil) \cdot X^{2N-j}$
3: $\mathsf{ct}_{\mathsf{br}} := (X^{b_{\mathsf{ms}}} \cdot \mathsf{tv}, 0) \in R_q^2$
4: **for** $0 \le i < n$ **do**
5: $\quad \mathsf{ct}_{\mathsf{br}} \leftarrow \mathsf{ct}_{\mathsf{br}} + (X^{a_{\mathsf{ms},i}} - 1) \cdot \left(\mathsf{brk}_{i,0}^{\top} \cdot h_{\mathsf{brk}}(\mathsf{ct}_{\mathsf{br}}),\ \mathsf{brk}_{i,1}^{\top} \cdot h_{\mathsf{brk}}(\mathsf{ct}_{\mathsf{br}}) \right)$
6: **end for**
7: **return** $\mathsf{ct}_{\mathsf{br}}$

constant term in the plaintext is $f(m)$, where $m = \mathtt{Dec}(\mathsf{sk}, \vec{c})$. The sample extraction algorithm converts an RLWE ciphertext into an LWE ciphertext, where the output LWE ciphertext decrypts to the constant term of the plaintext of the input RLWE ciphertext. For a more detailed analysis, we refer the reader to Appendix A.

Homomorphic Operations. We now describe the homomorphic operations of the TFHE scheme, defined as follows.

- $\underline{\mathtt{Add}}(\vec{c}_0, \vec{c}_1) \to \vec{c}_{\mathsf{add}}$: Given two LWE ciphertexts $\vec{c}_0, \vec{c}_1 \in \mathbb{Z}_q^{N+1}$, return $\vec{c}_{\mathsf{add}} = \vec{c}_0 + \vec{c}_1 \in \mathbb{Z}_q^{N+1}$.
- $\underline{\mathtt{Boot}}(\mathsf{pk}, f, \vec{c}) \to \vec{c}_{\mathsf{out}}$: Given a evaluation key pk, a negacyclic function $f : \mathbb{Z}_t \to \mathbb{Z}_t$, and an LWE ciphertext $\vec{c} \in \mathbb{Z}_q^{N+1}$, run Algorithm 2, and return the output LWE ciphertext $\vec{c}_{\mathsf{boot}} \in \mathbb{Z}_q^{N+1}$.

Algorithm 2 TFHE Bootstrapping

Input: Evaluation key pk, negacyclic function f, LWE ciphertext $\vec{c}$
Output: LWE ciphertext $\vec{c}_{\mathsf{boot}}$

1: Parse $\mathsf{pk} = (\mathsf{brk}, \mathsf{ksk}, \mathsf{pk})$
2: $\vec{c}_{\mathsf{ks}} \leftarrow \mathtt{KeySwitch}(\mathsf{ksk}, \vec{c})$
3: $\mathsf{ct}_{\mathsf{br}} \leftarrow \mathtt{BlindRotate}(\mathsf{brk}, f, \vec{c}_{\mathsf{ks}})$
4: $\vec{c}_{\mathsf{boot}} \leftarrow \mathtt{SampleExtract}(\mathsf{ct}_{\mathsf{br}})$
5: **return** $\vec{c}_{\mathsf{boot}}$

The TFHE scheme supports basic linear operations through homomorphic addition and the evaluation of arbitrary negacyclic functions through the bootstrapping operation. By combining these two operations, it can implement the homomorphic NAND operation, which is functionally complete. Thus, TFHE can theoretically evaluate any binary circuit. In practice, we select an appropriate function f to efficiently evaluate the target circuit. For more details on

homomorphic evaluation strategies in TFHE, we refer to applications of TFHE in various scenarios [ADDG24, BMMP18, LJ19].

Below, we summarize the core properties of TFHE bootstrapping. We note that the output of TFHE bootstrapping is always a well-formed ciphertext, regardless of the well-formedness of the input ciphertext.

Theorem 4 (TFHE Bootstrapping). *For any* $\vec{c} \in \mathbb{Z}_q^{N+1}$*, any* $\mathsf{pk} \in \mathsf{L}_{\mathsf{pk}}$*, any negacyclic function* f*, and* $\vec{c}_{\mathsf{boot}} \leftarrow \mathtt{Boot}(\mathsf{pk}, f, \vec{c})$*, it holds that* $\vec{c}_{\mathsf{boot}} \in \mathsf{L}_{\mathsf{ct}}$ *for* $B_{\mathsf{ct}} = 2n(N+1)\varepsilon_{\mathsf{brk}} + 4nNB_{\boldsymbol{\chi}}d_{\mathsf{brk}}$*. Moreover, if* $\vec{c} \in \mathsf{L}_{\mathsf{ct}}$*, then* $\mathtt{Dec}(\mathsf{sk}, \vec{c}_{\mathsf{boot}}) = f(\mathtt{Dec}(\mathsf{sk}, \vec{c}))$*.*

A major caveat of TFHE bootstrapping is that it can only evaluate negacyclic functions. A common way to solve this problem is to add a padding bit in front of the message. In other words, we assume that $m \in \mathbb{Z} \cap [0, t/2)$, as captured in L_{ct} (Fig. 11), and only evaluate functions from and to $\mathbb{Z} \cap [0, t/2)$. An alternative bootstrapping method called Full-Domain Functional Bootstrapping, which removes the negacyclic condition, is also been widely studied [KS23, MHW$^+$24]. While the concrete constructions differ, they usually use two or more TFHE bootstrappings, hence they share a similar characteristic as TFHE bootstrapping. In this work, we only use its functionality in an abstract way as below. We refer the readers to [KS23, MHW$^+$24] for more detail.

Theorem 5 (Full-Domain Functional Bootstrapping). *There exists an algorithm* $\mathtt{FDFB}$ *with the same signature as* $\mathtt{Boot}$ *such that, for any* $\vec{c} \in \mathbb{Z}_q^{N+1}$*, any* $\mathsf{pk} \in \mathsf{L}_{\mathsf{pk}}$*, any function* f*, and* $\vec{c}_{\mathsf{fdfb}} \leftarrow \mathtt{FDFB}(\mathsf{pk}, f, \vec{c})$*, it holds that* $\vec{c}_{\mathsf{fdfb}} \in \mathsf{L}_{\mathsf{ct}}$*. Moreover, if* $\vec{c} \in \mathsf{L}_{\mathsf{ct}}$*, then* $\mathtt{Dec}(\mathsf{sk}, \vec{c}_{\mathsf{fdfb}}) = f(\mathtt{Dec}(\mathsf{sk}, \vec{c}))$*.*

4.2 Our Sanitization Method

To design a sanitization algorithm, we modify the internal workings of TFHE bootstrapping, similar to prior works [Klu24, BI22], rather than following the approach in [DS16], which treats the entire bootstrapping procedure as a black box. The sanitization methods in [BI22, Klu24] primarily focus on randomizing the blind rotation step within bootstrapping. In particular, they leverage the randomized gadget decomposition technique from [AP14, BDPMW16] to achieve simulatability of the output ciphertexts. In contrast, our sanitization algorithm performs randomization only before key-switching and after blind rotation, thereby reducing the overhead introduced by randomization.

To be precise, we recall that the functionality of blind rotation corresponds to the homomorphic computation of $\mathsf{tv} \cdot X^{b_{\mathsf{ms}} + \sum_{i=0}^{n-1} a_{\mathsf{ms},i} \cdot s'_i}$, where tv is a public polynomial, referred to as the test vector, and $\vec{c}_{\mathsf{ms}} = (b_{\mathsf{ms}}, \vec{a}_{\mathsf{ms}})$ is a scaled LWE ciphertext, both of which appear in Algorithm 1. During blind rotation, the algorithm first multiplies tv by $X^{b_{\mathsf{ms}}}$ and then applies multiplication by $X^{a_{\mathsf{ms},i} \cdot s'_i}$ for each i, using the blind rotation keys. In this setting, the sender's private information lies in the ciphertext $\vec{c}_{\mathsf{ms}}$, and hence, prior works [Klu24, BI22] randomize the entire blind rotation procedure to ensure privacy.

To avoid heavy randomization during blind rotation, we first observe that the body part $\vec{a}_{\mathsf{ms}}$ of $\vec{c}_{\mathsf{ms}}$ can be efficiently randomized through adding encryption of zero, then $\vec{a}_{\mathsf{ms}}$ does not contain private information anymore, making it simulatable by the receiver. However, this does not solve the problem as the body b_{ms} still contains private information. Since blind rotation begins with multiplying $X^{b_{\mathsf{ms}}}$ to tv, then the result still contains private information, so the remaining multiplication of each $X^{a_{\mathsf{ms},i} \cdot s'_i}$ still should be randomized.

We resolve this issue by modifying the evaluation order: we first multiply $X^{a_{\mathsf{ms},i} \cdot s'_i}$ to tv and defer the multiplication by $X^{b\mathsf{ms}}$ until the final step. With this change, the evaluation of $\mathsf{tv} \cdot X^{\sum_{i=0}^{n-1} a_{\mathsf{ms},i} \cdot s'_i}$ does not involve any private information, as $\vec{a}_{\mathsf{ms}}$ is randomized and both tv and the blind rotation keys are known to the receiver. Thus, this part of the computation does not require any additional randomization. The only component that still involves private data is the final multiplication by $X^{b_{\mathsf{ms}}}$, since b_{ms} remains private. To handle this, we leverage the randomized linear evaluation method from [dCKK+24], which enables the multiplication of ring elements with ciphertexts in a circuit-private manner. As a result, we achieve sanitization by introducing randomization only at the beginning and end of the blind rotation, while keeping the core algorithm unchanged.

In the remainder of this section, we first present the randomization algorithms used in our sanitization method. We then describe our sanitization that incorporates these algorithms, and finally, we provide a performance analysis of our method in comparison with prior works [BI22, Klu24].

Randomization Algorithms for Sanitization. Below, we provide the randomization algorithms used in our sanitization method.

- $\underline{\texttt{EncZero}}(\mathsf{pk}) \to \mathsf{ct}$: Given a public key pk, sample $\boldsymbol{e}_1, \boldsymbol{e}_2 \leftarrow \mathcal{D}_{\mathbb{Z}^N, \sigma_r}$ and $\boldsymbol{e}_0 \leftarrow \mathcal{D}_{\mathbb{Z}^N, \tau_r}$, and return a RLWE ciphertext $\mathsf{ct} = \boldsymbol{e}_0 \cdot \mathsf{pk} + (\boldsymbol{e}_1, \boldsymbol{e}_2) \in R_q^2$.
- $\underline{\texttt{ReRand}}(\mathsf{pk}, \vec{c}) \to \vec{c}_{\mathsf{rand}}$: Given a public key pk and an LWE ciphertext $\vec{c} \in \mathbb{Z}_q^{N+1}$, return an LWE ciphertext $\vec{c}_{\mathsf{rand}} \leftarrow \vec{c} + \texttt{SampleExtract}(\texttt{EncZero}(\mathsf{pk})) \in \mathbb{Z}_q^{N+1}$.
- $\underline{\texttt{RandLinEval}}(\mathsf{pk}, \mathsf{ct}, \boldsymbol{x}) \to \mathsf{ct}_{\mathsf{lin}}$: Given a public key pk, an RLWE ciphertext ct, and a ring element $\boldsymbol{x} \in R_t$, sample $\hat{\boldsymbol{x}} \leftarrow \mathcal{D}_{\boldsymbol{x}+t\mathbb{Z}^N, \sigma_\ell}$ and $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}^N, \tau_\ell}$, and return an RLWE ciphertext $\mathsf{ct}_{\mathsf{lin}} \leftarrow \hat{\boldsymbol{x}} \cdot \mathsf{ct} + (\boldsymbol{e}, 0) + \texttt{EncZero}(\mathsf{pk})$.

`EncZero` serves as the core source of randomization in our sanitization method, as it allows the sender to generate fresh encryptions of zero independently. We note that prior works [BI22, Klu24] also utilize random encryptions of zero to mask the output ciphertext produced by blind rotation. However, our method uses random encryptions of zero to randomize both the input and output ciphertexts of blind rotation. The `EncZero` algorithm is inspired by the construction in [BI22], which achieves computational indistinguishability from uniform by relying on a reduction from the Hint-RLWE [KLSS23, MKMS22] problem to the standard RLWE problem. This approach significantly reduces the parameter overhead compared to the leftover hash lemma–based technique used in [Klu24].

Lemma 4 (Distribution of `EncZero`). *Let $\sigma_r, \tau_r > 0$ be reals. Let $B > 0$ be an upper bound of $\| \left[E_{\mathsf{pk}} \; S \right] \|_2$ for any $\mathsf{pk} \in \mathsf{L}_{\mathsf{pk}}$, where E_{pk} and S are the negacyclic*

matrices of $\boldsymbol{e}_{\mathsf{pk}} := \mathtt{Err}(\mathsf{pk})$ *and* $\mathsf{sk} = \boldsymbol{s}$ *respectively. If for* $0 < \varepsilon \leq 1/2, \kappa^{-2} := \frac{1}{\sigma_r^2} + \frac{B^2}{\tau_r^2} \leq \eta_\varepsilon(\mathbb{Z}^{2N})^{-2}$ *and* $\mathsf{RLWE}^{1,N,q}_{\kappa/\sqrt{2},\kappa/\sqrt{2}}$ *is hard, then the following holds.*

$$\begin{aligned}&\left\{(\mathsf{sk}, \mathsf{pk}, \mathsf{ct}_r) \mid \mathsf{ct}_r \leftarrow \mathtt{EncZero}_{\sigma_r,\tau_r}(\mathsf{pk})\right\}\\ &\approx_c \left\{(\mathsf{sk}, \mathsf{pk}, (-\boldsymbol{us} + \boldsymbol{e}_r, \boldsymbol{u})) \mid \boldsymbol{u} \xleftarrow{\$} R_q, \boldsymbol{e}_r \leftarrow \mathcal{D}^{\sigma_r,\tau_r}_{\mathtt{EncZero}}(\boldsymbol{s}, \boldsymbol{e}_{\mathsf{pk}})\right\}\end{aligned}$$

where $\mathcal{D}^{\sigma_r,\tau_r}_{\mathtt{EncZero}}(\boldsymbol{s}, \boldsymbol{e}_{\mathsf{pk}}) := \left\{\boldsymbol{se}_1 + \boldsymbol{e}_{\mathsf{pk}}\boldsymbol{e}_2 + \boldsymbol{e}_0 \mid \boldsymbol{e}_1, \boldsymbol{e}_2 \leftarrow \mathcal{D}_{\mathbb{Z}^N,\sigma_r}, \boldsymbol{e}_0 \leftarrow \mathcal{D}_{\mathbb{Z}^N,\tau_r}\right\}$.

By utilizing `EncZero`, we can randomize the mask part of the input ciphertext using `ReRand`, whose security is analyzed as follows.

Lemma 5 (Mask Rerandomization). *Let* $\vec{c} = (b, \vec{a}) \in \mathbb{Z}_q^{N+1}$ *be an LWE ciphertext such that* $\varphi_{\vec{s}}(\vec{c}) = q/t \cdot m + e$*, and* $\mathsf{pk} \in \mathsf{L}_{\mathsf{pk}}$*. If* $\sigma_r, \tau_r > 0$ *satisfy the conditions of Lemma 4, then the following holds.*

$$\begin{aligned}&\left\{(\mathsf{sk}, \mathsf{pk}, \mathtt{ReRand}_{\sigma_r,\tau_r}(\mathsf{pk}, \vec{c}))\right\}\\ &\quad\approx_c \left\{(\mathsf{sk}, \mathsf{pk}, (-\langle \vec{u}, \vec{s}\rangle + (q/t)m + e + e_r) \mid \vec{u} \xleftarrow{\$} \mathbb{Z}_q^N, e_r \leftarrow \mathcal{D}^{\sigma_r,\tau_r}_{\mathtt{EncZero}}|_0\right\}\end{aligned}$$

Lastly, we introduce `RandLinEval`, which enables the multiplication of any ring element $\boldsymbol{x} \in R_t$ in a circuit-private manner, inspired by the technique in [dCKK+24]. The core idea is to randomize $\boldsymbol{x}$ by sampling $\hat{\boldsymbol{x}} \in R$ from a discrete Gaussian distribution defined over the coset $\boldsymbol{x} + t\mathbb{Z}^N$. This randomization preserves correctness, since $\hat{\boldsymbol{x}} \equiv \boldsymbol{x} \pmod t$, while ensuring that the error component follows a discrete Gaussian distribution. Leveraging this property, we can multiply any ring element without leaking partial information through the ciphertext error. Finally, the mask part of the ciphertext is further randomized by adding an encryption of zero using `EncZero`, making the output ciphertext simulatable by the receiver.

Lemma 6 (Randomized Linear Evaluation). *Let* $\sigma_r, \tau_r, \sigma_\ell, \tau_\ell > 0$ *be reals, and* $\mathsf{ct} = (-\boldsymbol{as} + q/t \cdot \boldsymbol{m} + \boldsymbol{e}, \boldsymbol{a}) \in R_q^2$ *be an RLWE ciphertext. Let* $B > 0$ *be an upper bound of* $\|E\|_2$ *where* E *is the negacyclic matrix of* $\boldsymbol{e}$*. If* σ_r, τ_r *satisfy the condition of Lemma 4 and* $\sigma_\ell, \tau_\ell > 0$ *satisfy* $\frac{1}{\sigma_\ell^2} + \frac{B^2}{\tau_\ell^2} \leq \frac{1}{\eta_\varepsilon(t\mathbb{Z}^n)^2}$ *for some* $0 < \varepsilon \leq 1/2$*, then for any* $\boldsymbol{x} \in R$*, the following holds.*

$$\begin{aligned}&\{(\mathsf{sk}, \mathsf{pk}, \mathtt{RandLinEval}(\mathsf{pk}, \mathsf{ct}, \boldsymbol{x}))\}\\ &\quad\approx_c \{(\mathsf{sk}, \mathsf{pk}, \mathtt{RandLinEval}(\mathsf{pk}, \mathsf{ct}, 0) + (q/t \cdot (\boldsymbol{mx}), 0))\}\end{aligned}$$

Ciphertext Sanitization. Below, we present our sanitization method.

- $\underline{\mathtt{Setup}_{\mathsf{san}}}(\mathsf{pp}) \to (\sigma_r, \sigma_\ell, \tau_r, \tau_\ell)$: Given a public parameter pp, return width parameters $\sigma_r, \sigma_\ell, \tau_r, \tau_\ell$ for `ReRand` and `RandLinEval`.
- $\underline{\mathtt{Sanitize}}(\mathsf{pk}, \vec{c}) \to \vec{c}_{\mathsf{san}}$: Given a evaluation key pk and an LWE ciphertext $\vec{c} \in \mathbb{Z}_q^{N+1}$, run Algorithm 3 and return the output ciphertext $\vec{c}_{\mathsf{san}} \in \mathbb{Z}_q^{N+1}$.

Algorithm 3 Our Sanitization Method

Input: Evaluation key pk, LWE ciphertext $\vec{c}$
Output: LWE ciphertext $\vec{c}_{\mathsf{san}}$

Parse $\mathsf{pk} = (\mathsf{brk}, \mathsf{ksk}, \mathsf{pk})$
$\vec{c}_{\mathsf{rand}} \leftarrow \mathtt{ReRand}(\mathsf{pk}, \vec{c})$
$\vec{c}_{\mathsf{ks}} = (b_{\mathsf{ks}}, \vec{a}_{\mathsf{ks}}) \leftarrow \mathtt{KeySwitch}(\mathsf{ksk}, \vec{c}_{\mathsf{rand}})$
$\mathsf{ct}_{\mathsf{br}} \leftarrow \mathtt{BlindRotate}(\mathsf{brk}, \mathsf{id}, (0, \vec{a}_{\mathsf{ks}}))$
$\mathsf{ct}_{\mathsf{lin}} \leftarrow \mathtt{RandLinEval}(\mathsf{pk}, \mathsf{ct}_{\mathsf{br}}, X^{\lfloor \frac{2N}{q} \cdot b_{\mathsf{ks}} \rceil})$
$\vec{c}_{\mathsf{san}} \leftarrow \mathtt{SampleExtract}(\mathsf{ct}_{\mathsf{lin}})$
return $\vec{c}_{\mathsf{san}}$

A brief sketch of the security of our method is as follows. Given an input ciphertext $\vec{c}$, we first randomize its mask part using `ReRand`, resulting in a ciphertext $\vec{c}_{\mathsf{rand}}$ whose mask is computationally indistinguishable from a uniform sample from the receiver's viewpoint. Next, we apply key-switching to $\vec{c}_{\mathsf{rand}}$, obtaining $\vec{c}_{\mathsf{ks}}$. The mask part of $\vec{c}_{\mathsf{ks}}$ remains simulatable, as the key-switching operation does not affect the mask and the key-switching keys are known to the receiver.

We then perform blind rotation using only the mask part of $\vec{c}_{\mathsf{ks}}$, resulting in a ciphertext $\mathsf{ct}_{\mathsf{br}}$. At this point, $\mathsf{ct}_{\mathsf{br}}$ remains simulatable by the receiver, as it is computed using simulatable components of $\vec{c}_{\mathsf{ks}}$ and blind rotation keys, which are also known to the receiver. Subsequently, we multiply the body part of $\vec{c}_{\mathsf{ks}}$ using `RandLinEval`, yielding an output ciphertext $\mathsf{ct}_{\mathsf{lin}}$. Due to the security guarantees of `RandLinEval`, $\mathsf{ct}_{\mathsf{lin}}$ is also simulatable.

Finally, we apply `SampleExtract` to obtain the LWE ciphertext $\vec{c}_{\mathsf{san}}$. Since `SampleExtract` does not involve any private information, the resulting ciphertext $\vec{c}_{\mathsf{san}}$ is simulatable by the receiver. Therefore, the overall sanitization procedure yields an output ciphertext that is simulatable from the receiver's perspective. Below, we provide a detailed security analysis of our sanitization method.

Theorem 6. *Let σ_r, τ_r and σ_ℓ, τ_ℓ be reals that satisfy the conditions of Lemma 4 and Lemma 6 respectively. For an LWE encryption $\vec{c} \in \mathbb{Z}_q^{N+1}$ such that $\vec{c} \in \mathsf{L}_{\mathsf{ct}}$ and $m = \mathtt{Dec}(\mathsf{sk}, \vec{c})$, let $\vec{c}_{\mathsf{san}} \leftarrow \mathtt{Sanitize}(\mathsf{pk}, f, \vec{c})$. Then, $\mathtt{Dec}(\mathsf{sk}, \vec{c}_{\mathsf{san}}) = m$ holds and there exists a PPT algorithm `Sim` such that $\{(\mathsf{sk}, \mathsf{pk}, \vec{c}_{\mathsf{san}})\} \approx_c \{(\mathsf{sk}, \mathsf{pk}, \vec{c}_{\mathsf{sim}})\}$ for $\vec{c}_{\mathsf{sim}} \leftarrow \mathtt{Sim}(\mathsf{pk}, m)$.*

Corollary 1. *Let $\sigma_r, \tau_r, \sigma_\ell, \tau_\ell$ be reals that satisfy the conditions of Theorem 6. Then, `Sanitize` is a sanitization algorithm for ciphertexts in L_{ct}.*

4.3 Performance Analysis

In this subsection, we analyze and compare the performance of our new sanitization method with previous works [BI22, Klu24] that utilize the randomized gadget decomposition technique in [AP14, BDPMW16]. For comparison, we count the number of Gaussian samples required for sanitization, as Gaussian sampling

is often the primary bottleneck in practical implementations. For instance, in previous works [BI22, Klu24], Gaussian sampling accounted for up to 80% and 60%, respectively, of the total running time.

In our sanitization method, Gaussian sampling is required in `ReRand` and `RandLinEval`. In `ReRand`, we need N Gaussian samples each for $\mathcal{D}_{\mathbb{Z}^N,\sigma_r}$ and $\mathcal{D}_{\mathbb{Z}^N,\tau_r}$ when calling `EncZero`. However, since we only use the extracted LWE ciphertext of `EncZero`, we only need to sample the constant term of the Gaussian distribution. This reduces the requirement to $N+1$ Gaussian samples. In `RandLinEval`, we need N Gaussian samples each for $\mathcal{D}_{X^b+p\mathbb{Z}^N,\sigma_\ell}$ and $\mathcal{D}_{\mathbb{Z}^N,\tau_\ell}$. Additionally, we need $2N$ Gaussian samples from `EncZero`, as described earlier. We note that in our sanitization algorithm (Algorithm 3), `SampleExtract` is invoked immediately after `RandLinEval`. This means that only the constant term of the Gaussian samples is needed, similar to `ReRand`. This results in $2N+2$ Gaussian samples. Therefore, our method requires a total of $3N+3$ Gaussian samples.

In contrast, the major source of Gaussian sampling in previous works stems from their heavy use of randomized gadget decomposition, which requires $2dN$ Gaussian samples. Since we need to decompose an RLWE ciphertext n times for a single blind rotation, the total amount of Gaussian samples needed is $2dnN$. Moreover, both works require mask rerandomization, so additional $N+1$ Gaussian samples are needed for [BI22], and $h+1$ samples are needed for [Klu24] where h is the length of the LWE public key. We summarize the number of Gaussian samples required in Table 1. In short, our method reduces the required samples by up to a factor of $O(dn)$.

Ours	[BI22]	[Klu24]
$3N+3$	$2dnN+N+1$	$2dnN+h+1$

Table 1. Number of Gaussian samples required.

4.4 Discussion

Other Schemes. Our sanitization method can be applied to other FHEW-like cryptosystems [DM15, LMK+23] or acceleration techniques for optimized TFHE blind rotation [LMSS23, LY23, BCL+24], without significant modifications. This is mainly because blind rotation is used as a black-box subroutine in our construction, and the correctness and security rely only on the functionality of the blind rotation. To be specific, we note that all FHEW-like cryptosystems share the same bootstrapping pipeline and subroutine used in Algorithm 2. Crucially, while the implementation of the blind rotation functionality differs by scheme, they all output an RLWE ciphertext of $\mathsf{tv} \cdot X^{\lfloor \frac{2N}{q} \cdot b \rceil + \langle \lfloor \frac{2N}{q} \cdot \vec{a} \rceil, s' \rangle}$ given an LWE ciphertext $(b, \vec{a})$, where s' is the corresponding LWE key and tv is some test vector.

Moreover, our two core randomization functionalities, `ReRand` and `RandLinEval`, work over any LWE and RLWE ciphertexts respectively, and hence are essentially scheme-agnostic. Therefore, we can adapt our proof for Theorem 6 to other FHEW-like schemes almost verbatim.

Functional Sanitization. Traditionally, sanitization requires the message preserving property, meaning that it needs to output an encryption of the same message as the input ciphertext. However, similar to functional bootstrapping, one can naturally extend this notion to the case where the sanitization algorithm outputs a ciphertext that encrypts an evaluation of some function f on the input message. Indeed, our sanitization algorithm can be easily modified to simultaneously compute a private, negacyclic function f during sanitization, using the intrinsic property of TFHE bootstrapping.

Naively, by computing $\mathsf{ct}_{\mathsf{br}} \leftarrow \mathtt{BlindRotate}(\mathsf{brk}, f, (0, \vec{a}_{\mathsf{ks}}))$ in line 4 of Algorithm 3 instead, we can easily see that the output ciphertext of our sanitization encrypts $f(\mathtt{Dec}(\vec{c}))$. However, this method does not hide f, as it is included as an input to blind rotation and hence affects the output ciphertext. To solve this problem, we decompose the test vector tv of f as $\mathsf{tv} = \mathsf{tv}' \cdot (q/t)$, where $\mathsf{tv}' \in R_t$. Then we modify Algorithm 3 as follows. First, we compute $\mathsf{ct}_{\mathsf{br}} \leftarrow \mathtt{BlindRotate}(\mathsf{brk}, q/t, (0, \vec{a}_{\mathsf{ks}}))$, i.e., evaluate blind rotation with the constant function q/t. Then, by the functionality of blind rotation, $\mathsf{ct}_{\mathsf{br}}$ now encrypts $(q/t) \cdot X^{\langle \lfloor \frac{2N}{q} \cdot \vec{a} \rceil, s' \rangle}$. To correct this, we compute $\mathsf{ct}_{\mathsf{lin}} \leftarrow \mathtt{RandLinEval}(\mathsf{pk}, \mathsf{ct}_{\mathsf{br}}, \mathsf{tv}' \cdot X^{\lfloor \frac{2N}{q} \cdot b \rceil})$. Now, the result is correct, and the output ciphertext does not contain any information about f as the constant q/t is public and any information about tv' is erased by `RandLinEval`.

Using this property, we can optimize our construction of the circuit-private evaluation algorithm in Theorem 2 by merging the last TFHE bootstrapping with sanitization. This removes the computational overhead of circuit-private evaluation for TFHE, since our sanitization has nearly negligible performance difference compared to the original TFHE bootstrapping.

5 ZKAoK for TFHE

In this section, we present a zero-knowledge argument of knowledge (ZKAoK) for the TFHE scheme. As noted earlier, the sanitization algorithm provides security only against semi-honest adversaries in oblivious circuit evaluation (OCE) protocols. To enhance security against malicious receivers, we require a ZKAoK that verifies the well-formedness of the ciphertexts and public keys provided by the receiver.

To design an efficient ZKAoK for TFHE, we employ the PIOPs for RLWE ciphertexts introduced in [HLSS25], which support the proof of arithmetic constraints and norm bounds over polynomial rings. However, directly applying these techniques to TFHE is intractable, as TFHE public keys include LWE ciphertexts and the well-formedness of blind rotation keys involves non-arithmetic

constraints over polynomial rings. To overcome this challenge, we adopt the packed key generation technique from recent works [KDE+23, XZW+24], where the receiver generates packed public keys for TFHE, and the sender reconstructs the standard public keys using a key unroll algorithm. Although originally proposed to reduce the communication cost of public keys, we repurpose this method for efficient ZKAoK construction. This is because the packed public keys consist solely of RLWE ciphertexts and their structure is defined entirely by arithmetic constraints, making them compatible with the PIOPs in [HLSS25]. By leveraging packed key generation, we resolve the key obstacles in constructing ZKAoKs for the well-formedness of TFHE public keys.

However, applying the PIOPs for RLWE ciphertexts to TFHE ciphertexts remains inefficient, as TFHE ciphertexts are based on the LWE structure. To address this, we design a new OCE protocol tailored specifically to the TFHE scheme. In our protocol, security against a malicious receiver is achieved without requiring a ZKAoK for ciphertexts, by enforcing the sender to perform bootstrapping on the input ciphertexts. Since the output of TFHE bootstrapping is always well-formed, regardless of the well-formedness of the input ciphertexts, the sender can ensure well-formedness without relying on ZKAoKs for TFHE ciphertexts. As a result, by incorporating these optimizations, we successfully design an efficient OCE protocol for TFHE that is secure against malicious receivers. In the remainder of this section, we first briefly review the PIOPs supported in [HLSS25] and the packed key generation method introduced in [KDE+23, XZW+24]. We then design a PIOP for verifying the well-formedness of packed public keys and present our optimized OCE protocol for TFHE.

5.1 PIOP Toolbox

We briefly review the PIOPs for RLWE ciphertexts introduced in [HLSS25]. For simplicity, we outline only their functionality here, for the full protocols, we refer to [HLSS25].

Vector Representations. In [HLSS25], the witness space is defined as the vector space $\mathbb{Z}_p^N$, where p is a prime such that $2N \mid p-1$. Thus, for elements in the polynomial ring R_p, we utilize the following two vector representations: the coefficient representation and the number theoretic transform (NTT) representation.

- $\underline{\texttt{Coeff}}(\boldsymbol{a}) \to \vec{a}$: Given a polynomial $\boldsymbol{a} = \sum_{i=0}^{N-1} a_i X^i$, outputs a vector $\vec{a} = (a_0, \ldots, a_{N-1})$.
- $\underline{\texttt{NTT}}(\boldsymbol{a}) \to \underline{\vec{a}}$: Given a polynomial $\boldsymbol{a} = \sum_{i=0}^{N-1} a_i X^i$, outputs a vector $\underline{\vec{a}} = \big(\boldsymbol{a}(\xi), \boldsymbol{a}(\xi^3), \ldots, \boldsymbol{a}(\xi^{2N-1})\big)$, where ξ is a $2N$-th root of unity in $\mathbb{Z}_p$.

We note that `Coeff` outputs the coefficient vector of the input polynomial, and therefore preserves the norm. In contrast, `NTT` outputs the evaluations of the polynomial at the $2N$-th roots of unity in $\mathbb{Z}_p$; as a result, it does not preserve the norm. However, since `NTT` is an isomorphism between R_p and $\mathbb{Z}_p^N$, it preserves arithmetic constraints.

Polynomial Encoding for Vectors. Since PIOPs admit only polynomial oracles for messages exchanged between the prover and verifier, the prover must encode witness vectors into polynomials in order to generate polynomial oracles. For this polynomial encoding, we use the following method.

- $\underline{\mathtt{Ecd}}(\vec{w}) \to \boldsymbol{w}$: Given a vector $\vec{w} \in \mathbb{Z}_p^N$, output the polynomial $\boldsymbol{w} \in \mathbb{Z}_p^{\leq 2N}[X]$ such that $\boldsymbol{w}(\xi^{2i}) = w_i$ for $0 \leq i < N$.
- $\underline{\mathtt{Dcd}}(\boldsymbol{w}) \to \vec{w}$: Given a polynomial $\boldsymbol{w} \in \mathbb{Z}_p[X]$, output a vector $\vec{w} = (\boldsymbol{w}(\xi^0), \ldots, \boldsymbol{w}(\xi^{2(N-1)}))$.

PIOP for Linear Relatinos. For witness vectors $\vec{a}_i, \vec{b}_i \in \mathbb{Z}_p^N$ satisfying the linear relations $\vec{b}_i = M_i\vec{a}_i$ for public matrices $M_i \in \mathbb{Z}_p^{N\times N}$, where $0 \leq i < k$, their validity can be proven using the following PIOP.

$$\Pi_{\mathsf{lc}}(M_0, \ldots, M_{k-1}; [\![\hat{\boldsymbol{a}}_0]\!], \ldots, [\![\hat{\boldsymbol{a}}_{k-1}]\!]; [\![\hat{\boldsymbol{b}}_0]\!], \ldots, [\![\hat{\boldsymbol{b}}_{k-1}]\!])$$

Instance: Matrices $M_i \in \mathbb{Z}_p^{N\times N}$ for $0 \leq i < k$.
Witness: Vectors $\vec{a}_i = \mathtt{Dcd}(\hat{\boldsymbol{a}}_i)$ and $\vec{b}_i = \mathtt{Dcd}(\hat{\boldsymbol{b}}_i)$ for $0 \leq i < k$.
Statement: $\vec{b}_i = M_i\vec{a}_i$ for $0 \leq i < k$.

Fig. 5. PIOP for batched linear relation constraints

This PIOP can be used to prove relations between the coefficient representation and the NTT representation of polynomials, as well as automorphism relations between vector representations. This is because the NTT can be represented as a Vandermonde matrix, and automorphisms correspond to permutation matrices.

PIOP for Arithmetic Constraints. An arithmetic constraint over $\mathbb{Z}_p^N$, where addition and multiplication are component-wise, can be represented as a multivariate polynomial $\vec{\mathsf{C}}$ over the product ring $\mathbb{Z}_p^N$. The satisfiability of $\vec{\mathsf{C}}$ for witness vectors in $\mathbb{Z}_p^N$ can be verified using the following PIOP.

$$\Pi_{\mathsf{ac}}(\vec{\mathsf{C}}; [\![\hat{\boldsymbol{a}}_0]\!], \ldots, [\![\hat{\boldsymbol{a}}_{k-1}]\!])$$

Instance: k-variate polynomial $\vec{\mathsf{C}} \in (\mathbb{Z}_p^N)[\vec{X}_0, \ldots, \vec{X}_{k-1}]$.
Witness: Vectors $\vec{a}_i = \mathtt{Dcd}(\boldsymbol{a}_i)$ for $0 \leq i < k$.
Statement: $\vec{\mathsf{C}}(\vec{a}_0, \ldots, \vec{a}_{k-1}) = 0$.

Fig. 6. Functionality of the PIOP for arithmetic constraints

This PIOP can be used to prove arithmetic relations between polynomials through their NTT vector representations, as the NTT is an isomorphism and therefore preserves arithmetic constraints.

PIOP for Norm Constraints. For witness vectors $\vec{a}_0, \ldots, \vec{a}_{k-1} \in \mathbb{Z}_p^N$, one can prove their L^∞-norm upper bound in a batched manner using the following PIOP.

$$\Pi_{\mathsf{nc}}(B; [\![\hat{\boldsymbol{a}}_0]\!], \ldots, [\![\hat{\boldsymbol{a}}_{k-1}]\!])$$

Instance: Norm bound B.
Witness: Vectors $\vec{a}_i = \mathtt{Dcd}(\hat{\boldsymbol{a}}_i)$ for $0 \le i < k$.
Statement: $\|\vec{a}_i\|_\infty \le B$ for $0 \le i < k$.

Fig. 7. Functionality of the PIOP for L^∞-norm constraints

This PIOP can be used to prove norm constraints of polynomials through their coefficient representations.

5.2 Packed Key Generation for TFHE

Below, we present the packed key generation algorithms from [KDE+23, XZW+24].

- $\underline{\mathtt{Setup}_{\mathsf{pkg}}}(\mathsf{pp}) \to (p, \boldsymbol{\chi}', \boldsymbol{\chi}_{\mathsf{pksk}}, \vec{g}_{\mathsf{atk}}, h_{\mathsf{atk}}, \vec{g}_{\mathsf{rlk}}, h_{\mathsf{rlk}})$: Given a public parameter pp, return a ciphertext modulus p for packed public keys, error distributions $\boldsymbol{\chi}', \boldsymbol{\chi}_{\mathsf{pksk}}$ for packed public keys, and gadget vectors and decompositions $(\vec{g}_{\mathsf{atk}}, h_{\mathsf{atk}})$ and $(\vec{g}_{\mathsf{rlk}}, h_{\mathsf{rlk}})$ for automorphism keys and relinearization keys.
- $\underline{\mathtt{PackedKeyGen}}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{ppk})$: Given a public parameter pp, generate a key pair $(\mathsf{sk}, \mathsf{ppk})$ as follows, where $\mathsf{ppk} = (\mathsf{atk}, \mathsf{rlk}, \mathsf{pbrk}, \mathsf{pksk}, \mathsf{ek}')$.
  1. Sample $\boldsymbol{s} = \sum_{i=0}^{N-1} s_i \cdot X^i \xleftarrow{\$} \mathbb{B}^N$, $\boldsymbol{s}' = \sum_{i=0}^{n-1} s'_i \cdot X^i \xleftarrow{\$} \mathbb{B}^n$, and set $\mathsf{sk} = \boldsymbol{s}$.
  2. Sample $\vec{\boldsymbol{a}}_{\mathsf{atk},i} \leftarrow R_p^{d_{\mathsf{atk}}}$ and $\vec{\boldsymbol{e}}_{\mathsf{atk},i} \leftarrow \boldsymbol{\chi}'^{d_{\mathsf{atk}}}$, then compute the automorphism keys $\mathsf{atk}_i = (-\boldsymbol{s} \cdot \vec{\boldsymbol{a}}_{\mathsf{atk},i} + \boldsymbol{s}(X^{2^{i+1}+1}) \cdot \vec{g}_{\mathsf{atk}} + \vec{\boldsymbol{e}}_{\mathsf{atk},i},\ \vec{\boldsymbol{a}}_{\mathsf{atk},i}) \in R_p^{d_{\mathsf{atk}} \times 2}$ for $0 \le i < \log N$, and set $\mathsf{atk} = \{\mathsf{atk}_i\}_{0 \le i < \log N}$.
  3. Sample $\vec{\boldsymbol{a}}_{\mathsf{rlk}} \leftarrow R_p^{d_{\mathsf{rlk}}}$ and $\vec{\boldsymbol{e}}_{\mathsf{rlk}} \leftarrow \boldsymbol{\chi}'^{d_{\mathsf{rlk}}}$, then compute the relinearization key $\mathsf{rlk} = (-\boldsymbol{s} \cdot \vec{\boldsymbol{a}}_{\mathsf{rlk}} + \boldsymbol{s}^2 \cdot \vec{g}_{\mathsf{rlk}} + \vec{\boldsymbol{e}}_{\mathsf{rlk}},\ \vec{\boldsymbol{a}}_{\mathsf{rlk}}) \in R_p^{d_{\mathsf{rlk}} \times 2}$.
  4. Sample $\vec{\boldsymbol{a}}_{\mathsf{pbrk}} \xleftarrow{\$} R_p^{d_{\mathsf{brk}}}$, $\vec{\boldsymbol{e}}_{\mathsf{brk}} \leftarrow \boldsymbol{\chi}'^{d_{\mathsf{brk}}}$, then compute the packed blind rotation key $\mathsf{pbrk} = (-\boldsymbol{s} \cdot \vec{\boldsymbol{a}}_{\mathsf{pbrk}} + [N^{-1}]_p \boldsymbol{s}' \cdot \lfloor p/q \cdot \vec{g}_{\mathsf{brk}} \rceil + \vec{\boldsymbol{e}}_{\mathsf{pbrk}},\ \vec{\boldsymbol{a}}_{\mathsf{pbrk}}) \in R_p^{d_{\mathsf{brk}} \times 2}$.
  5. Sample $\vec{\boldsymbol{a}}_{\mathsf{pksk}} \xleftarrow{\$} R_p^{d_{\mathsf{ksk}}}$, $\vec{\boldsymbol{e}}_{\mathsf{pksk}} \leftarrow \boldsymbol{\chi}_{\mathsf{pksk}}^{d_{\mathsf{ksk}}}$, then compute the packed key-switching key $\mathsf{pksk} = (-\boldsymbol{s}' \cdot \vec{\boldsymbol{a}}_{\mathsf{pksk}} + \boldsymbol{s} \cdot \lfloor p/q \cdot \vec{g}_{\mathsf{ksk}} \rceil + \vec{\boldsymbol{e}}_{\mathsf{pksk}},\ \vec{\boldsymbol{a}}_{\mathsf{pksk}}) \in R_p^{d_{\mathsf{ksk}} \times 2}$.
  6. Sample $\boldsymbol{a}_{\mathsf{ek}'} \xleftarrow{\$} R_p$, $\boldsymbol{e}_{\mathsf{ek}'} \leftarrow \boldsymbol{\chi}'$, then compute the temporal public key $\mathsf{ek}' = (-\boldsymbol{a}_{\mathsf{ek}'} \boldsymbol{s} + \boldsymbol{e}_{\mathsf{ek}'}, \boldsymbol{a}_{\mathsf{ek}'}) \in R_p^2$.
- $\underline{\mathtt{KeyUnroll}}(\mathsf{ppk}) \to \mathsf{pk}$: Given a packed public key $\mathsf{ppk} = (\mathsf{atk}, \mathsf{rlk}, \mathsf{pbrk}, \mathsf{pksk}, \mathsf{ek}')$, unroll the packed public key as follows.

1. Compute a temporal blind rotation key brk' in R_p using Algorithm 10 of [KDE+23] with input $(\mathsf{atk}, \mathsf{rlk}, \mathsf{pbrk})$.
2. Compute a temporal key-switching key ksk' in R_p using Lemma 11 of [XZW+24] with input pksk.
3. Set $\mathsf{brk} = \lfloor \frac{q}{p} \cdot \mathsf{brk}' \rceil$, $\mathsf{ksk} = \lfloor \frac{q}{p} \cdot \mathsf{ksk}' \rceil$, and $\mathsf{pk} = \lfloor \frac{q}{p} \cdot \mathsf{pk}' \rceil$.

We note that packed public keys are generated over the ring R_p for some prime $p \approx 2^\lambda$, which is typically much larger than the ciphertext modulus q used in TFHE. This design choice is motivated by two main reasons: to manage noise growth during the key unroll process and to achieve negligible soundness error in the PIOP.

First, the key unroll algorithm can be viewed as performing homomorphic operations on the packed public keys. If the unrolling is performed directly over modulus q, the resulting noise becomes significantly larger than that of ordinary TFHE public keys. However, by unrolling the keys over a larger modulus $p \gg q$ and subsequently reducing the modulus from p to q, the noise growth introduced by the key unroll algorithm can be effectively mitigated.

Second, the PIOP for RLWE ciphertexts operates over the ring R_p, where p is a prime. Since the soundness error of the PIOP is roughly proportional to $1/p$, generating proofs over a larger modulus p substantially reduces the soundness error. Therefore, using a separate, larger modulus for packed public key generation not only helps manage noise growth but also improves the soundness guarantees of the PIOP without incurring additional proof overhead that would arise from working directly over q.

5.3 ZKAoK for Packed Public Keys

We now design a ZKAoK for packed public keys within the PIOP framework as follows.

Theorem 7. *An interactive protocol Π_{ppk} described in Fig. 12 is an HVZK PIOP for the relation* Rppk *defined in Fig. 8.*

Finally, we show that proving the well-formedness of a packed public key implies the well-formedness of the corresponding decompressed public keys obtained via the key unroll algorithm, which is described below. Therefore, the sender can ensure the well-formedness of the public keys by verifying the well-formedness of the packed public key.

$$
\mathsf{R}_{\mathsf{ppk}} = \left\{ (\mathsf{x}_{\mathsf{ppk}}, \mathsf{w}_{\mathsf{ppk}}) \;\middle|\; \begin{array}{r}
\mathsf{x}_{\mathsf{ppk}} = \mathsf{ppk} = \left(\left\{ \mathsf{atk}_i \right\}_{0 \le i < \log N}, \mathsf{rlk}, \mathsf{pbrk}, \mathsf{pksk}, \mathsf{ek}' \right), \\
\mathsf{w}_{\mathsf{pk}} = \left(\boldsymbol{s}, \boldsymbol{s}', \left\{ \vec{\boldsymbol{e}}_{\mathsf{atk},i} \right\}_{0 \le i < \log N}, \vec{\boldsymbol{e}}_{\mathsf{rlk}}, \vec{\boldsymbol{e}}_{\mathsf{pbrk}}, \vec{\boldsymbol{e}}_{\mathsf{pksk}}, \boldsymbol{e}_{\mathsf{ek}'} \right), \\
\boldsymbol{s} \in \{0,1\}^N,\ \boldsymbol{s}' \in \{0,1\}^n,\ \|\vec{\boldsymbol{e}}_{\mathsf{pksk}}\|_\infty \le B_{\boldsymbol{\chi}_{\mathsf{pksk}}} \\
\|\vec{\boldsymbol{e}}_{\mathsf{atk},i}\|_\infty, \|\vec{\boldsymbol{e}}_{\mathsf{rlk}}\|_\infty, \|\vec{\boldsymbol{e}}_{\mathsf{pbrk}}\|_\infty, \|\vec{\boldsymbol{e}}_{\mathsf{ek}'}\|_\infty \le B_{\boldsymbol{\chi}'}, \\
\vec{\boldsymbol{b}}_{\mathsf{atk},i} = -\boldsymbol{s} \cdot \vec{\boldsymbol{a}}_{\mathsf{atk},i} + \boldsymbol{s}(X^{2^{i+1}+1}) \cdot \vec{g}_{\mathsf{atk}} + \vec{\boldsymbol{e}}_{\mathsf{atk},i}, \\
\text{where } \mathsf{atk}_i = (\vec{\boldsymbol{b}}_{\mathsf{atk},i}, \vec{\boldsymbol{a}}_{\mathsf{atk},i}) \text{ for } 0 \le i < \log N \\
\vec{\boldsymbol{b}}_{\mathsf{rlk}} = -\boldsymbol{s} \cdot \vec{\boldsymbol{a}}_{\mathsf{rlk}} + \boldsymbol{s}^2 \cdot \vec{g}_{\mathsf{rlk}} + \vec{\boldsymbol{e}}_{\mathsf{rlk}}, \\
\text{where } \mathsf{rlk} = (\vec{\boldsymbol{b}}_{\mathsf{rlk}}, \vec{\boldsymbol{a}}_{\mathsf{rlk}}) \\
\vec{\boldsymbol{b}}_{\mathsf{pbrk}} = -\boldsymbol{s} \cdot \vec{\boldsymbol{a}}_{\mathsf{pbrk}} + \boldsymbol{s}' \cdot \vec{g}_{\mathsf{pbrk}} + \vec{\boldsymbol{e}}_{\mathsf{pbrk}}, \\
\text{where } \mathsf{pbrk} = (\vec{\boldsymbol{b}}_{\mathsf{pbrk}}, \vec{\boldsymbol{a}}_{\mathsf{pbrk}}) \\
\vec{\boldsymbol{b}}_{\mathsf{pksk}} = -\boldsymbol{s}' \cdot \vec{\boldsymbol{a}}_{\mathsf{pksk}} + \boldsymbol{s} \cdot \vec{g}_{\mathsf{pksk}} + \vec{\boldsymbol{e}}_{\mathsf{pksk}}, \\
\text{where } \mathsf{pksk} = (\vec{\boldsymbol{b}}_{\mathsf{pksk}}, \vec{\boldsymbol{a}}_{\mathsf{pksk}}) \\
\boldsymbol{b}_{\mathsf{ek}'} = -\boldsymbol{a}_{\mathsf{ek}'}\boldsymbol{s} + \boldsymbol{e}_{\mathsf{ek}'} \text{ where } \mathsf{ek}' = (\boldsymbol{b}_{\mathsf{ek}'}, \boldsymbol{a}_{\mathsf{ek}'})
\end{array} \right\}
$$

Fig. 8. Relation for well-formed packed public keys

Theorem 8. *Let* $\mathsf{ppk} \in \mathsf{R}_{\mathsf{ppk}}$ *with norm bounds* $B_{\boldsymbol{\chi}'}$ *and* B_{pksk}*. Then, for* $\mathsf{pk} \leftarrow \mathtt{KeyUnroll}(\mathsf{ppk})$*, we have* $\mathsf{pk} \in \mathsf{R}_{\mathsf{pk}}$ *with norm bounds* $B_{\mathsf{brk}} = \frac{N+1}{2} + \frac{q}{p} \cdot (N^2 \varepsilon_{\mathsf{atk}} + dB_{\mathsf{atk}} N^2 \delta_{\mathsf{atk}})$*,* $B_{\mathsf{ksk}} = \frac{q}{p} B_{\mathsf{ksk}} + \frac{n+1}{2}$*, and* $B_{\mathsf{ek}} = \frac{q}{p} B_{\mathsf{ek}'} + \frac{N+1}{2}$*.*

5.4 Optimized OCE for TFHE

We present an optimized OCE protocol for TFHE that eliminates the need for ZKAoKs for ciphertexts. The core idea relies on a key property of TFHE bootstrapping: its output is always a well-formed ciphertext, regardless of the well-formedness of the input, as described in Theorems 4 and 5. By leveraging this property, the sender can ensure that input ciphertexts are well-formed by performing bootstrapping on them prior to circuit evaluation. The detailed protocol and its security analysis are presented below.

Theorem 9. *Let* $\Pi_{\mathsf{OCE}}^{\mathsf{TFHE}}$ *be a sender-receiver protocol described in Fig. 9. Then* $\Pi_{\mathsf{OCE}}^{\mathsf{TFHE}}$ *is a secure OCE protocol against semi-honest senders and malicious receivers in the* $\mathsf{F}_{\mathsf{ZKAoK}}(\mathsf{R}_{\mathsf{ppk}})$*-hybrid model.*

$\Pi^{\mathsf{TFHE}}_{\mathsf{OCE}}$

$\Pi^{\mathsf{TFHE}}_{\mathsf{OCE}}$ runs with a sender and a receiver.

Input: The receiver has inputs $\mathsf{pt}_0, \ldots, \mathsf{pt}_{\ell-1}$, and the sender has a circuit C to be evaluated.

Output: The receiver obtains $\mathsf{pt}_{\mathsf{out}} \leftarrow \mathtt{Dec}(\mathsf{sk}, \overline{\mathsf{ct}}_{\mathsf{out}})$, and the sender obtains nothing.

Setup:

1. Both the sender and the receiver generate a public parameter $\mathsf{pp} \leftarrow \mathtt{Setup}(1^\lambda)$.
2. The receiver generates $(\mathsf{sk}, \mathsf{ppk}) \leftarrow \mathtt{PackedKeyGen}(\mathsf{pp})$. The receiver then sends ppk to the sender.
3. The receiver and the sender runs $\mathsf{F}_{\mathsf{ZKAoK}}(\mathsf{R}_{\mathsf{ppk}})$, acting as a prover and a verifier, respectively. If the output is 1, the sender computes a evaluation key $\mathsf{pk} \leftarrow \mathtt{KeyUnroll}(\mathsf{ppk})$. Otherwise, the sender aborts.

Protocol:

1. The receiver generates $\mathsf{ct}_i \leftarrow \mathtt{Enc}(\mathsf{pk}, \mathsf{pt}_i)$ for $0 \le i < \ell$. The receiver then sends $(\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})$ to the sender.
2. The sender computes $\mathsf{ct}'_i \leftarrow \mathtt{FDFB}(\mathsf{pk}, \eta, \mathsf{ct}_i)$ for $0 \le i < \ell$ where $\eta(x) = x \mod \lfloor t/2 \rfloor$, $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathtt{Eval}(\mathsf{pk}, C, \mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\ell-1})$ and $\overline{\mathsf{ct}}_{\mathsf{out}} \leftarrow \mathtt{Sanitize}(\mathsf{pk}, \mathsf{ct}_{\mathsf{out}})$. The sender then sends $\overline{\mathsf{ct}}_{\mathsf{out}}$ to the receiver.

Fig. 9. Actively secure oblivious circuit evaluation protocol from TFHE

In the above protocol, we implicitly assume that the inputs and outputs of C lies in $\mathbb{Z} \cap [0, t/2)$. Thus, the sender computes $\eta(x)$ to ensure the plaintext lies in this range. We note that other constructions are also possible without this restriction. For instance, the sender might instead apply `FDFB` with identity function at the start, and also run `Eval` with `FDFB`. Additionally, if the entire circuit C is negacyclic, we can replace `FDFB` at start with the usual TFHE bootstrapping with identity function.

6 Experiments

In this section, we present the experimental results of our sanitization algorithm from our proof-of-concept implementation and compare them with those of previous works.

6.1 Optimization Techniques

Gaussian Sampling. Like previous works [BI22, Klu24], our randomization heavily depends on the properties of discrete Gaussian distributions. Unfortunately, sampling discrete Gaussian distributions is highly inefficient, which was the main bottleneck of [BI22]. Kluczniak [Klu24] instead used a rounded Gaussian distribution, which is much more efficient to sample, but their change was only based on heuristics.

In our implementation, we also replace the discrete Gaussian distribution with its rounded counterpart, but we give a rigorous proof for this change. To be specific, we prove the following lemma, which can be directly used in our security proofs, replacing Lemma 3.

Lemma 7 (Rounded Gaussian Lemma). *Let $A \in \mathbb{Z}^{m\times n}$ be a matrix, $t > 0$ be an integer, and $\sigma, \tau > 0$ be reals such that $\frac{1}{\sigma^2} + \frac{\|A\|_2^2}{\tau^2} \leq \frac{1}{\eta_\varepsilon(t\mathbb{Z}^n)^2}$ for $0 < \varepsilon < 1/2$. Then, for any $\mathbf{a} \in \mathbb{Z}^n$, the following holds.*

$$\Delta_{\mathrm{SD}}\left(A\mathcal{D}_{\mathbf{a}+t\mathbb{Z}^n,\sigma} + \lfloor \mathcal{D}_{\mathbb{R}^m,\tau} \rceil, \left\lfloor \mathcal{D}_{\mathbb{R}^m,\sqrt{\sigma^2\mathbf{E}\mathbf{E}^\top + \tau^2\mathbf{I}}} \right\rceil\right) \leq 4\varepsilon$$

Even with the above lemma, we still need to sample from $\mathcal{D}_{\mathbb{Z}^N,\sigma_r}$ and $\mathcal{D}_{X^b+t\mathbb{Z}^N,\sigma_\ell}$. In these cases, we use an inversion sampler with precomputed cumulative distribution tables. For sampling from the coset $X^b + t\mathbb{Z}^N$, we use the fact that $\mathcal{D}_{X^b+t\mathbb{Z}^N,\sigma_\ell} \equiv \mathbf{X}_b\mathcal{D}_{1+t\mathbb{Z}^N,\sigma_\ell}$, where $\mathbf{X}_b$ is the negacyclic matrix of X^b. Thus, we can sample the random variable for arbitrary b with samples from $\mathcal{D}_{1+t\mathbb{Z},\sigma_\ell}$ and $\mathcal{D}_{t\mathbb{Z},\sigma_\ell}$.

Exact Polynomial Multiplication. In most TFHE implementations, the Fast Fourier Transform (FFT) is used for fast polynomial multiplication. However, since FFT is computed over floating-point numbers, it can only provide an approximate result, especially when the coefficients of the polynomials are large. In the original TFHE scheme, this error was tolerable because all RLWE operations are inherently noisy. However, when implementing randomized algorithms, this can be problematic, as the additional FFT error is not simulatable. Therefore, to securely implement the previous TFHE sanitization methods with randomizations [BI22, Klu24], much slower exact polynomial multiplication algorithms, such as the Number Theoretic Transform (NTT) must be used, or parameters must be heavily restricted.

On the other hand, our sanitization algorithm uses the blind rotation of the original TFHE bootstrapping as a black box, freeing us from such restrictions. In our method, exact polynomial multiplication is only required in the `ReRand` and `RandLinEval` steps, which we use the split-FFT trick [Hwa23, WHS+25].

6.2 Benchmark Results

We present the benchmark results of our and previous sanitization algorithms in Table 2, along with the base TFHE bootstrapping performance. For the parameters selection, we refer to Appendix B. We implemented our sanitization

algorithm in Go, based on the TFHE-go library [Hwa23].[1] All benchmark results of our implementation were measured on a sender machine equipped with an Intel Xeon Platinum 8268 CPU.

For the sanitization algorithms from [BI22, Klu24], we directly take the performance numbers of their algorithms and the base TFHE bootstrapping from the respective papers. Kluczniak [Klu24] provides two implementations, one based on NTT and the other on FFT. In the table, we denote them as (Int) and (Double) respectively.

	Ours	Ours (Malicious)	[BI22]	[Klu24] (Int)	[Klu24] (Double)
Sanitization	42.17	112.1	7500	1360	1330
Bootstrap	41.36	110.4	420	140	270

Table 2. Performance of sanitization algorithms (in miliseconds). All experiments were run in single-thread.

As we can clearly see, our sanitization method offers significant speedups compared to previous works. Specifically, it is approximately 177.85 and 31.53 times faster than [BI22, Klu24] in the semi-honest receiver setting, respectively. When compared to the original TFHE bootstrapping with the same parameters, our method is only 1.9% slower, with a 0.81 ms difference in performance. In contrast, [BI22] and [Klu24] are 17.85 and 4.92 times slower than the original TFHE bootstrapping, respectively.

We also implement our ZKAoKs for TFHE (Fig. 12) in Go, and present benchmark results in Table 3. To the best of our knowledge, we are the first concrete construction for the TFHE scheme, so we omit the comparison with other works. Still, the benchmark results show that our ZKAoKs offer practically deployable performance.

Prove (1 thread)	Prove (16 threads)	Verify
196	22.63	24.47

Table 3. Performance of ZKAoK for TFHE (in seconds).

Acknowledgements

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2023-00211649) and the

[1] Our code is available at `https://github.com/SNUCP/tfhe-sanitization`.

Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. RS-2024-00399491, Development of Privacy-Preserving Multiparty Computation Techniques for Secure Multiparty Data Integration).

References

ADDG24. Martin R Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. Crypto dark matter on the torus: Oblivious prfs from shallow prfs and tfhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 447–476, 2024.

AP14. Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*, pages 297–314. Springer, 2014.

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

BCL+24. Loris Bergerat, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, Adeline Roux-Langlois, and Samuel Tap. New secret keys for enhanced performance in (t) fhe. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 2547–2561, 2024.

BDPMW16. Florian Bourse, Rafaël Del Pino, Michele Minelli, and Hoeteck Wee. Fhe circuit privacy almost for free. In *Annual International Cryptology Conference*, pages 62–89. Springer, 2016.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325, 2012.

BI22. Florian Bourse and Malika Izabachène. Plug-and-play sanitization for TFHE. Cryptology ePrint Archive, Paper 2022/1438, 2022. `https://eprint.iacr.org/2022/1438`.

BMMP18. Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, pages 483–512, 2018.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

BSCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Proceedings, Part II, of the 14th International Conference on Theory of Cryptography-Volume 9986*, pages 31–60, 2016.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

CBBZ23. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 499–530. Springer, 2023.

CDPP22. Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder VL Pereira. Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 563–577, 2022.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *international conference on the theory and application of cryptology and information security*, pages 3–33. Springer, 2016.

CHM+20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 738–768, 2020.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

CL01. Yan-Cheng Chang and Chi-Jen Lu. Oblivious polynomial evaluation and oblivious neural learning. In *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7*, pages 369–384. Springer, 2001.

dCKK+24. Leo de Castro, Duhyeong Kim, Miran Kim, Keewoo Lee, Seonhong Min, and Yongsoo Song. More efficient lattice-based OLE from circuit-private linear HE with polynomial overhead. Cryptology ePrint Archive, Paper 2024/1534, 2024.

DM15. Léo Ducas and Daniele Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640. Springer, 2015.

DS16. Léo Ducas and Damien Stehlé. Sanitization of fhe ciphertexts. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I 35*, pages 294–310. Springer, 2016.

FIPR05. Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*, pages 303–324. Springer, 2005.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

HLSS25. Intak Hwang, Hyeonbum Lee, Jinyeong Seo, and Yongsoo Song. Practical zero-knowledge PIOP for maliciously secure multiparty homomorphic encryption. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2025.

HMS25. Intak Hwang, Seonhong Min, and Yongsoo Song. Ciphertext-simulatable HE from BFV with randomized evaluation. Cryptology ePrint Archive, Paper 2025/203, 2025.

Hwa23. Intak Hwang. TFHE-go. Online: `https://github.com/sp301415/tfhe-go`, 2023.

Joy24. Marc Joye. Tfhe public-key encryption revisited. In *Cryptographers' Track at the RSA Conference*, pages 277–291. Springer, 2024.

KDE+23. Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for rlwe-based homomorphic encryption. *IEEE Transactions on Computers*, 73(1):86–96, 2023.

KLSS23. Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-mlwe. In *Annual International Cryptology Conference*, pages 549–580. Springer, 2023.

Klu24. Kamil Kluczniak. Circuit privacy for FHEW/TFHE-style fully homomorphic encryption in practice. *IACR Communications in Cryptology (CiC)*, 1(4):33, 2024.

KS23. Kamil Kluczniak and Leonard Schild. FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):501–537, 2023.

LJ19. Qian Lou and Lei Jiang. She: a fast and accurate deep neural network for encrypted data. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 10035–10043, 2019.

LMK+23. Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 227–256. Springer, 2023.

LMSS23. Changmin Lee, Seonhong Min, Jinyeong Seo, and Yongsoo Song. Faster tfhe bootstrapping with block binary keys. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, pages 2–13, 2023.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*, pages 1–23. Springer, 2010.

LY23. Kang Hoon Lee and Ji Won Yoon. Discretization error reduction for high precision torus fully homomorphic encryption. In *IACR International Conference on Public-Key Cryptography*, pages 33–62. Springer, 2023.

MHW+24. Shihe Ma, Tairong Huang, Anyu Wang, Qixian Zhou, and Xiaoyun Wang. Fast and accurate: Efficient full-domain functional bootstrap and digit decomposition for homomorphic computation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(1):592–616, 2024.

MKMS22. Jose Maria Bermudo Mera, Angshuman Karmakar, Tilen Marc, and Azam Soleimanian. Efficient lattice-based inner-product functional encryption. In *IACR International Conference on Public-Key Cryptography*, pages 163–193. Springer, 2022.

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 700–718. Springer, 2012.

MR07. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007.

NP06. Moni Naor and Benny Pinkas. Oblivious polynomial evaluation. *SIAM Journal on Computing*, 35(5):1254–1281, 2006.

Pei10. Chris Peikert. An efficient and parallel gaussian sampler for lattices. In *Annual Cryptology Conference*, pages 80–97. Springer, 2010.

Rab81. Michael O Rabin. How to exchange secrets by oblivious transfer. *Tech. Memo TR-81, Aiken Computation Laboratory, Harvard University*, 1981.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

SSTX09. Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 617–635. Springer, 2009.

WHS+25. Ruida Wang, Jincheol Ha, Xuan Shen, Xianhui Lu, Chunling Chen, Kunpeng Wang, and Jooyoung Lee. Refined TFHE leveled homomorphic evaluation and its application. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2025.

XZW+24. Binwu Xiang, Jiang Zhang, Kaixing Wang, Yi Deng, and Dengguo Feng. Ntru-based bootstrapping for mk-fhes without using overstretched parameters. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 241–270, 2024.

A Noise Analysis

Lemma 8 (Key Switching). *For an LWE ciphertext $\vec{c} \in \mathbb{Z}_q^{N+1}$, the variance of the noise of $\vec{c}_{\mathsf{ks}} \leftarrow \mathtt{KeySwitch}(\mathsf{ksk}, \vec{c})$ is bounded by*

$$\mathtt{Var}(\mathtt{Err}_{s'}(\mathbf{c}_{\mathsf{ks}})) \leq \mathtt{Var}(\mathtt{Err}_{s}(\mathbf{c})) + V_{\mathtt{KeySwitch}}$$

for

$$V_{\mathtt{KeySwitch}} := \frac{1}{3}\varepsilon_{ksk}^2 N + \frac{1}{3}\alpha^2 d_{ksk} N \delta_{ksk}^2.$$

Proof. Let $\vec{c} = (b, a_0, \ldots, a_{N-1})$ and $\vec{e}_i$ denote the encryption noise vector of i-th key-switching key ksk_i for $0 \leq i < N$. Then, we have

$$\begin{aligned}\varphi_{s'}(\vec{c}_{\mathsf{ks}}) - \varphi_s(\vec{c}) &= b + \sum_{i=0}^{N-1} h_{\mathsf{ksk}}(a_i)^\top \varphi_{s'}(\mathsf{ksk}_i) - (b + \sum_{i=0}^{N-1} a_i \cdot s_i) \\ &= \sum_{i=0}^{N-1} (\langle h_{\mathsf{ksk}}(a_i), \vec{g}_{\mathsf{ksk}} \cdot s_i + \mathbf{e}_i \rangle - a_i s_i)\end{aligned}$$

$$= \sum_{i=0}^{N-1} s_i \cdot (\langle h_{\mathsf{ksk}}(a_i), \vec{g}_{\mathsf{ksk}} \rangle - a_i) + \sum_{i=0}^{N-1} \langle h_{\mathsf{ksk}}(a_i), \vec{e}_i \rangle$$

from the definition. Then, since each s_i is either zero or one, the variance of the final error term is bounded by $\frac{1}{3}\varepsilon_{ksk}^2 N + \frac{1}{3}\alpha^2 d_{ksk} N \delta_{ksk}^2$. This concludes the proof.

Lemma 9 (Modulus Switching). *For an LWE ciphertext $\vec{c} = (b, a_0, \ldots, a_{n-1}) \in \mathbb{Z}_q^{n+1}$, the variance of the noise of $\vec{c}_{\mathsf{ms}} = (\lfloor \frac{2N}{q} \cdot b \rceil, \lfloor \frac{2N}{q} \cdot a_0 \rceil, \ldots, \lfloor \frac{2N}{q} \cdot a_{n-1} \rceil) \in \mathbb{Z}_{2N}$ is bounded by*

$$\mathtt{Var}(\mathtt{Err}_{s'}(\vec{c}_{\mathsf{ms}})) \leq \left(\frac{2N}{q}\right)^2 \mathtt{Var}(\mathtt{Err}_{s'}(\vec{c})) + V_{\mathtt{ModSwitch}}$$

for $V_{\mathtt{ModSwitch}} := \frac{n+1}{12}$.

Proof. We have

$$\varphi_{s'}(\vec{c}_{\mathsf{ms}}) - \frac{2N}{q}\varphi_{s'}(\vec{c}) = (\lfloor \frac{2N}{q} \cdot b \rceil - \frac{2N}{q} \cdot b) + \sum_{i=0}^{n-1} (\lfloor \frac{2N}{q} \cdot a_i \rceil - \frac{2N}{q} \cdot a_i) \cdot s'_i.$$

Similar to the previous proofs, we assume that the ciphertext follows the uniform distribution. Therefore, $\lfloor \frac{2N}{q} \cdot b \rceil - \frac{2N}{q} \cdot b$ and $\lfloor \frac{2N}{q} \cdot a_i \rceil - \frac{2N}{q} \cdot a_i$ follow a uniform distribution over $\frac{1}{q}\mathbb{Z} \cap [-\frac{1}{2}, \frac{1}{2}]$ for $0 \leq i < n$. Moreover, as s_i is either one or zero, the variance of the term above is bounded by $\frac{n+1}{12}$, respectively.

Lemma 10 (Blind Rotation). *For an LWE ciphertext $\vec{c} \in \mathbb{Z}_q^{n+1}$, the variance of the noise of $\mathsf{ct}_{\mathsf{br}} \leftarrow \mathtt{BlindRotate}(\mathsf{brk}, \vec{c})$ is bounded by*

$$\mathtt{Var}(\mathtt{Err}_s(\mathsf{ct}_{\mathsf{br}})) \leq V_{\mathtt{BlindRotate}} := \frac{2}{3} n(N+1)\varepsilon_{brk}^2 + \frac{4}{3} d_{\mathsf{brk}} n N \beta^2 \delta_{brk}^2.$$

Proof. Let acc_i be the value of the accumulator before i-th iteration of the for loop in Algorithm 1, and acc_n denote the output of Algorithm 1. Also, let us follow the same notations from the algorithm description. Then, by definition, it follows that

$$\begin{aligned}
&\varphi_{\boldsymbol{s}}(\mathsf{acc}_{i+1}) \\
&= \varphi_{\boldsymbol{s}}(\mathsf{acc}_i) + (X^{\tilde{a}_i} - 1) \cdot \varphi_{\boldsymbol{s}}(\mathsf{acc}_i \boxdot \mathsf{brk}_i) \\
&= \varphi_{\boldsymbol{s}}(\mathsf{acc}_i) + (X^{\tilde{a}_i} - 1) \cdot \langle h_{\mathsf{brk}}(\mathsf{acc}_i), \vec{g}_{\mathsf{brk}} \cdot \begin{bmatrix} 1 \\ s \end{bmatrix} s'_i + \mathbf{e}_i \rangle \\
&= X^{\tilde{a}_i s'_i} \varphi_{\boldsymbol{s}}(\mathsf{acc}_i) + (X^{\tilde{a}_i} - 1) \cdot (s'_i \left(\langle h_{\mathsf{brk}}(\mathsf{acc}_i), \vec{g}_{\mathsf{brk}} \rangle - \mathsf{acc}_i\right) \cdot \begin{bmatrix} 1 \\ t \end{bmatrix} \\
&\quad + \langle h_{\mathsf{brk}}(\mathsf{acc}_i), \mathbf{e}_i \rangle)
\end{aligned}$$

where $\vec{e}_i$ denotes the noise vector of i-th blind rotation key brk_i. As s'_i is a binary value and $\boldsymbol{s}$ is a polynomial with binary coefficients, the variance of the introduced error can be bounded by

$$\frac{2}{3}(N+1)\varepsilon_{brk}^2 + \frac{4}{3}d_{\mathsf{brk}}N\beta^2\delta_{brk}^2,$$

respectively. Note that each line introduces an additional error, and monomial multiplication only rotates the coefficient vector (up to sign). So the upper bound of the variance for the final noise term is given by

$$\frac{2}{3}n(N+1)\varepsilon_{brk}^2 + \frac{4}{3}d_{\mathsf{brk}}nN\beta^2\delta_{brk}^2.$$

B Parameter Selection

Our parameters for the base TFHE scheme are given in Table 4. We set $t = 2^2$, which are common parameters for instantiating binary TFHE. We choose the LWE parameters n, α and the RLWE parameters N, β to achieve 128-bit security, as verified by the lattice estimator [APS15]. All parameters support bootstrapping failure rate less than 2^{-128}.

For the error analysis, we used the average-case analysis from Appendix A. Note that usually in the malicious receiver setting, we cannot blindly use average case analysis as the receiver may maliciously generate the keys and ciphertexts. In particular, our ZKAoKs only prove the bounds of the witnesses, not their exact distributions. Therefore, for malicious receiver case, we only use the average-case analysis for noise term related to the mask, as they can be guaranteed to be uniform in the CRS model. For all other terms, we use worst-case analysis.

Receiver Type	n	N	q	α	β
Semi-Honest	690	2048	2^{64}	$2^{49.67}$	$2^{13.97}$
Malicious	850	4096	$65340^8 + 1$	$2^{109.46}$	$2^{23.65}$

Table 4. Base parameters for TFHE in the semi-honest and malicious receiver setting.

B_{brk}	d_{brk}	B_{ksk}	d_{ksk}	$B_{\mathsf{rlk}}, B_{\mathsf{atk}}$	$d_{\mathsf{rlk}}, d_{\mathsf{atk}}$
2^{13}	3	2^3	5	2^{32}	4

Table 5. Gadget decomposition parameters for TFHE.

Now we provide the Gaussian parameters $\sigma_r, \tau_r, \sigma_\ell, \tau_\ell$. We selected them to minimize the output error variance while satisfying the conditions of Lemma 4 and Lemma 6. All parameters support sanitization failure rate less than 2^{-128}.

Receiver Type	σ_r	τ_r	σ_ℓ	τ_ℓ
Semi-Honest	$2^{11.99}$	$2^{34.74}$	$2^{9.19}$	$2^{56.20}$
Malicious	$2^{10.39}$	$2^{29.60}$	$2^{9.44}$	$2^{56.16}$

Table 6. Gaussian parameters for `Sanitize` in the semi-honest and malicious receiver setting.

C Deferred Relations

$$
\mathsf{R}_{\mathsf{pk}} = \left\{ (\mathsf{x}_{\mathsf{pk}}, \mathsf{w}_{\mathsf{pk}}) \;\middle|\; \begin{array}{r}
\mathsf{x}_{\mathsf{pk}} = \Big(\big\{ (\mathsf{brk}_{i,0}, \mathsf{brk}_{i,1}) \big\}_{0 \le i < n}, \{\mathsf{ksk}_i\}_{0 \le i < N}, \mathsf{pk} \Big), \\
\mathsf{w}_{\mathsf{pk}} = \Big(\boldsymbol{s}, \vec{s}', \big\{ (\vec{\boldsymbol{e}}_{\mathsf{brk},i,0}, \vec{\boldsymbol{e}}_{\mathsf{brk},i,1}) \big\}_{0 \le i < n}, \big\{ \vec{e}_{\mathsf{ksk},i} \big\}_{0 \le i < N}, \boldsymbol{e}_{\mathsf{ek}} \Big), \\
\\
\boldsymbol{s} \in \{0,1\}^N,\ \vec{s}' \in \{0,1\}^n,\ \|\vec{e}_{\mathsf{ksk},i}\|_\infty \le B_\chi \\
\|\vec{\boldsymbol{e}}_{\mathsf{brk},i,0}\|_\infty, \|\vec{\boldsymbol{e}}_{\mathsf{brk},i,1}\|_\infty, \|\vec{\boldsymbol{e}}_{\mathsf{ek}}\|_\infty \le B_{\boldsymbol{\chi}}, \\
\\
\vec{\boldsymbol{b}}_{\mathsf{brk},i,0} = -\boldsymbol{s} \cdot \vec{\boldsymbol{a}}_{i,0} + s'_i \cdot \vec{g}_{\mathsf{brk}} + \vec{\boldsymbol{e}}_{\mathsf{brk},i,0} \text{ for } 0 \le i < n, \\
\vec{\boldsymbol{b}}_{\mathsf{brk},i,1} = -\boldsymbol{s} \cdot \vec{\boldsymbol{a}}_{i,1} + s'_i \boldsymbol{s} \cdot \vec{g}_{\mathsf{brk}} + \vec{\boldsymbol{e}}_{\mathsf{brk},i,1} \text{ for } 0 \le i < n, \\
\text{where } \mathsf{brk}_{i,0} = (\vec{\boldsymbol{b}}_{\mathsf{brk},i,0}, \vec{\boldsymbol{a}}_{\mathsf{brk},i,0}), \mathsf{brk}_{i,1} = (\vec{\boldsymbol{b}}_{\mathsf{brk},i,1}, \vec{\boldsymbol{a}}_{\mathsf{brk},i,1}), \\
\\
\vec{b}_{\mathsf{ksk},i} = -A_{\mathsf{ksk},i} \vec{s}' + s_i \cdot \vec{g}_{\mathsf{ksk}} + \vec{e}_{\mathsf{ksk},i} \text{ for } 0 \le i < N, \\
\text{where } \mathsf{ksk}_i = (\vec{b}_{\mathsf{ksk},i}, -A_{\mathsf{ksk},i}) \\
\\
\boldsymbol{b}_{\mathsf{ek}} = -\boldsymbol{a}_{\mathsf{ek}} \boldsymbol{s} + \boldsymbol{e}_{\mathsf{ek}} \text{ where } \mathsf{pk} = (\boldsymbol{b}_{\mathsf{ek}}, \boldsymbol{a}_{\mathsf{ek}})
\end{array} \right\}
$$

Fig. 10. Relation for well-formed TFHE public keys

$$\mathsf{R}_{\mathsf{ct}} = \left\{ (\mathsf{x}_{\mathsf{ct}}, \mathsf{w}_{\mathsf{ct}}) \;\middle|\; \begin{array}{r} \mathsf{x}_{\mathsf{ct}} = \vec{c} \in \mathbb{Z}_q^{N+1},\ \mathsf{w}_{\mathsf{ct}} = (m, \vec{s}, e, \boldsymbol{e}_{\mathsf{pk}}) \in \mathbb{Z}_t \times \mathbb{Z}_q^N \times \mathbb{Z}_q \times R_q, \\ \vec{s} \in \{0,1\}^N,\ |e| \le B_{\mathsf{ct}},\ \|\boldsymbol{e}_{\mathsf{pk}}\|_\infty \le B_{\boldsymbol{\chi}}, m \in \mathbb{Z} \cap [0, t/2) \\ b = -\langle \vec{a}, \vec{s} \rangle + q/t \cdot m + e,\ \text{where } \vec{c} = (b, \vec{a}) \\ \boldsymbol{b}_{\mathsf{pk}} = -\boldsymbol{a}_{\mathsf{pk}}\boldsymbol{s} + \boldsymbol{e}_{\mathsf{pk}}, \\ \text{where } \mathsf{pk} = (\boldsymbol{b}_{\mathsf{pk}}, \boldsymbol{a}_{\mathsf{pk}}) \text{ and } \boldsymbol{s} = \sum_{i=0}^{N-1} s_i X^i \end{array} \right\}$$

Fig. 11. Relation for well-formed TFHE ciphertexts

D Deferred Protocols

Π_{ppk}

Instance: Packed public key $\mathsf{ppk} = (\mathsf{atk}, \mathsf{rlk}, \mathsf{pbrk}, \mathsf{pksk}, \mathsf{ek}')$.
Witness: $\boldsymbol{s}, \boldsymbol{s}', \boldsymbol{e}_{\mathsf{ek}'} \in R_p$, $\vec{\boldsymbol{e}}_{\mathsf{atk},i} \in R_p^{d_{\mathsf{atk}}}$, $\vec{\boldsymbol{e}}_{\mathsf{rlk}} \in R_p^{d_{\mathsf{rlk}}}$, $\vec{\boldsymbol{e}}_{\mathsf{pbrk}} \in R_p^{d_{\mathsf{brk}}}$, and $\vec{\boldsymbol{e}}_{\mathsf{pksk}} \in R_p^{d_{\mathsf{ksk}}}$.
Statement: $\mathsf{pk} \in \mathsf{L}_{\mathsf{pk}}$.

1. The prover $\mathtt{P}$ computes the following.

$$\hat{\boldsymbol{s}} \leftarrow \mathtt{Ecd}(\mathtt{Coeff}(\boldsymbol{s})),\ \underline{\hat{\boldsymbol{s}}} \leftarrow \mathtt{Ecd}(\mathtt{NTT}(\boldsymbol{s})),$$
$$\hat{\boldsymbol{s}}' \leftarrow \mathtt{Ecd}(\mathtt{Coeff}(\boldsymbol{s}')),\ \underline{\hat{\boldsymbol{s}}}' \leftarrow \mathtt{Ecd}(\mathtt{NTT}(\boldsymbol{s}'))$$
$$\hat{\boldsymbol{e}}_{\mathsf{ek}'} \leftarrow \mathtt{Ecd}(\mathtt{Coeff}(\boldsymbol{e}_{\mathsf{ek}'})),\ \underline{\hat{\boldsymbol{e}}}_{\mathsf{ek}'} \leftarrow \mathtt{Ecd}(\mathtt{NTT}(\boldsymbol{e}_{\mathsf{ek}'}))$$
For $0 \le i < \log N$:
$$\underline{\hat{\boldsymbol{s}}}_{\mathsf{atk},i} \leftarrow \mathtt{Ecd}\Big(\mathtt{NTT}\big(\boldsymbol{s}(X^{2^{i+1}+1})\big)\Big)$$
For $0 \le i < \log N$ and $0 \le j < d_{\mathsf{atk}}$:
$$\hat{\boldsymbol{e}}_{\mathsf{atk},i,j} \leftarrow \mathtt{Ecd}(\mathtt{Coeff}(\boldsymbol{e}_{\mathsf{atk},i,j})),\ \underline{\hat{\boldsymbol{e}}}_{\mathsf{atk},i,j} \leftarrow \mathtt{Ecd}(\mathtt{NTT}(\boldsymbol{e}_{\mathsf{atk},i,j}))$$
For $0 \le j < d_{\mathsf{rlk}}$:
$$\hat{\boldsymbol{e}}_{\mathsf{rlk},j} \leftarrow \mathtt{Ecd}(\mathtt{Coeff}(\boldsymbol{e}_{\mathsf{rlk},j})),\ \underline{\hat{\boldsymbol{e}}}_{\mathsf{rlk},j} \leftarrow \mathtt{Ecd}(\mathtt{NTT}(\boldsymbol{e}_{\mathsf{rlk},j}))$$
For $0 \le j < d_{\mathsf{brk}}$:
$$\hat{\boldsymbol{e}}_{\mathsf{pbrk},j} \leftarrow \mathtt{Ecd}(\mathtt{Coeff}(\boldsymbol{e}_{\mathsf{pbrk},j})),\ \underline{\hat{\boldsymbol{e}}}_{\mathsf{pbrk},j} \leftarrow \mathtt{Ecd}(\mathtt{NTT}(\boldsymbol{e}_{\mathsf{pbrk},j}))$$
For $0 \le j < d_{\mathsf{ksk}}$:
$$\hat{\boldsymbol{e}}_{\mathsf{pksk},j} \leftarrow \mathtt{Ecd}(\mathtt{Coeff}(\boldsymbol{e}_{\mathsf{pksk},j})),\ \underline{\hat{\boldsymbol{e}}}_{\mathsf{pksk},j} \leftarrow \mathtt{Ecd}(\mathtt{NTT}(\boldsymbol{e}_{\mathsf{pksk},j}))$$

Then, the prover sends polynomial oracles $[\![\hat{\boldsymbol{s}}]\!]$, $[\![\underline{\hat{\boldsymbol{s}}}]\!]$, $[\![\hat{\boldsymbol{s}}']\!]$, $[\![\underline{\hat{\boldsymbol{s}}}']\!]$, $[\![\hat{\boldsymbol{e}}_{\mathsf{ek}'}]\!]$, $[\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{ek}'}]\!]$, $\{[\![\underline{\hat{\boldsymbol{s}}}_{\mathsf{atk},\mathrm{i}}]\!]\}$, $\{[\![\hat{\boldsymbol{e}}_{\mathsf{atk},i,j}]\!]$, $[\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{atk},i,j}]\!]\}$, $\{[\![\hat{\boldsymbol{e}}_{\mathsf{rlk},j}]\!]$, $[\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{rlk},j}]\!]\}$, $\{[\![\hat{\boldsymbol{e}}_{\mathsf{pbrk},j}]\!]$, $[\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{pbrk},j}]\!]\}$, $\{[\![\hat{\boldsymbol{e}}_{\mathsf{pksk},j}]\!]$, $[\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{pksk},j}]\!]\}$, of degree $< L$ to the verifier $\mathtt{V}$.
2. The verifier $\mathtt{V}$ sends a random challenge $\delta \leftarrow \mathcal{U}(\mathbb{Z}_p \setminus \mathbb{H})$.
3. For the binarity of secret keys, the prover $\mathtt{P}$ and the verifier $\mathtt{V}$ invoke the following PIOP, where $\vec{b} = (1, \ldots, 1, 0, \ldots, 0)$ with the first n components equal to 1.

$$\Pi_{\mathsf{ac}}\Big(\vec{X}_0 \circ (\vec{X}_0 - \vec{1}) + \delta \cdot \vec{X}_1 \circ (\vec{X}_1 - \vec{b}); [\![\hat{\boldsymbol{s}}]\!], [\![\hat{\boldsymbol{s}}']\!]\Big)$$

4. For the smallness of errors, the prover $\mathtt{P}$ and the verifier $\mathtt{V}$ invoke the following PIOP, where $B_{\boldsymbol{\chi}'}$ is a norm bound of $\boldsymbol{\chi}'$ and $B_{\boldsymbol{\chi}_{\mathsf{pksk}}}$ is a norm bound of $\boldsymbol{\chi}_{\mathsf{pksk}}$.

$$\Pi_{\mathsf{nc}}\Big(B_{\boldsymbol{\chi}'}; [\![\hat{\boldsymbol{e}}_{\mathsf{ek}'}]\!], \{[\![\hat{\boldsymbol{e}}_{\mathsf{atk},i,j}]\!]\}, \{[\![\hat{\boldsymbol{e}}_{\mathsf{rlk},j}]\!]\}, \{[\![\hat{\boldsymbol{e}}_{\mathsf{pbrk},j}]\!]\}\Big)$$

$$\Pi_{\mathsf{nc}}\Big(B_{\boldsymbol{\chi}_{\mathsf{pksk}}}; \{[\![\hat{\boldsymbol{e}}_{\mathsf{pksk},j}]\!]\}\Big)$$

5. For the validity of NTT representations, the prover $\mathtt{P}$ and the verifier $\mathtt{V}$ invoke the following PIOP, where T is the Vandermonde matrix for $\mathtt{NTT}$.

$$\Pi_{\mathsf{lc}}\Big(T; [\![\hat{\boldsymbol{s}}]\!], [\![\hat{\boldsymbol{s}}']\!], [\![\hat{\boldsymbol{e}}_{\mathsf{ek}'}]\!], \{[\![\hat{\boldsymbol{e}}_{\mathsf{atk},i,j}]\!]\}, \{[\![\hat{\boldsymbol{e}}_{\mathsf{rlk},j}]\!]\}, \{[\![\hat{\boldsymbol{e}}_{\mathsf{pbrk},j}]\!]\}, \{[\![\hat{\boldsymbol{e}}_{\mathsf{pksk},j}]\!]\},$$
$$[\![\underline{\hat{\boldsymbol{s}}}]\!], [\![\underline{\hat{\boldsymbol{s}}}']\!], [\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{ek}'}]\!], \{[\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{atk},i,j}]\!]\}, \{[\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{rlk},j}]\!]\}, \{[\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{pbrk},j}]\!]\}, \{[\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{pksk},j}]\!]\}\Big)$$

6. For the validity of the automorphism keys, the prover $\mathtt{P}$ and the verifier $\mathtt{V}$ invoke the following PIOPs, where $\mathsf{atk}_i = (\vec{\boldsymbol{b}}_{\mathsf{atk},i}, \vec{\boldsymbol{a}}_{\mathsf{atk},i})$, $\underline{\vec{a}}_{\mathsf{atk},i,j} = \mathtt{NTT}(\boldsymbol{a}_{\mathsf{atk},i,j})$, $\underline{\vec{b}}_{\mathsf{atk},i,j} = \mathtt{NTT}(\boldsymbol{b}_{\mathsf{atk},i,j})$, and P_i is the permutation matrix for the automorphisms $X \mapsto X^{2^{i+1}+1}$ for $0 \le i < \log N$ and $0 \le j < d_{\mathsf{atk}}$.

$$\Pi_{\mathsf{lc}}\Big(\{P_i\}; \{[\![\underline{\hat{\boldsymbol{s}}}]\!]\}; \{[\![\underline{\hat{\boldsymbol{s}}}_{\mathsf{atk},i}]\!]\}\Big)$$
$$\Pi_{\mathsf{ac}}\Bigg(\sum_{i=0}^{\log N-1}\sum_{j=0}^{d_{\mathsf{atk}}-1} \delta^{j+i\cdot d_{\mathsf{atk}}} \cdot (\underline{\vec{b}}_{\mathsf{atk},i,j} + \underline{\vec{a}}_{\mathsf{atk},i,j} \circ \vec{X}_0 - \lfloor p/B_{\mathsf{atk}}^{j+1} \rceil \cdot \vec{X}_{i+1} - \vec{X}_{j+i\cdot d_{\mathsf{atk}}+\log N+1})$$
$$; [\![\underline{\hat{\boldsymbol{s}}}]\!], \{[\![\underline{\hat{\boldsymbol{s}}}_{\mathsf{atk},i}]\!]\}, \{[\![\underline{\hat{\boldsymbol{e}}}_{\mathsf{atk},i,j}]\!]\}\Bigg)$$

7. For the validity of the relinearization key, the prover $\mathtt{P}$ and the verifier $\mathtt{V}$ invoke the following PIOPs, where $\mathsf{rlk} = (\vec{\boldsymbol{b}}_{\mathsf{rlk}}, \vec{\boldsymbol{a}}_{\mathsf{rlk}})$, $\underline{\vec{a}}_{\mathsf{rlk},j} = \mathtt{NTT}(\boldsymbol{a}_{\mathsf{rlk},j})$, and $\underline{\vec{b}}_{\mathsf{rlk},j} = \mathtt{NTT}(\boldsymbol{b}_{\mathsf{rlk},j})$ for $0 \le j < d_{\mathsf{rlk}}$.

$$\Pi_{\mathsf{ac}}\Bigg(\sum_{j=0}^{d_{\mathsf{rlk}}-1} \delta^j \cdot (\underline{\vec{b}}_{\mathsf{rlk},j} + \underline{\vec{a}}_{\mathsf{rlk},j} \circ \vec{X}_0 - \lfloor p/B_{\mathsf{rlk}}^{j+1} \rceil \cdot \vec{X}_0^2 - \vec{X}_{j+1})$$

$$; [\![\hat{\underline{\boldsymbol{s}}}]\!], \{[\![\hat{\underline{\boldsymbol{e}}}_{\mathsf{rlk},j}]\!]\}\Big)$$

8. For the validity of the packed blind rotation key, the prover $\mathtt{P}$ and the verifier $\mathtt{V}$ invoke the following PIOPs, where $\mathsf{pbrk} = (\vec{\boldsymbol{b}}_{\mathsf{pbrk}}, \vec{\boldsymbol{a}}_{\mathsf{pbrk}})$, $\vec{\underline{a}}_{\mathsf{pbrk},j} = \mathtt{NTT}(\boldsymbol{a}_{\mathsf{pbrk},j})$, and $\vec{\underline{b}}_{\mathsf{pbrk},j} = \mathtt{NTT}(\boldsymbol{b}_{\mathsf{pbrk},j})$ for $0 \le j < d_{\mathsf{brk}}$.

$$\Pi_{\mathsf{ac}}\Bigg(\sum_{j=0}^{d_{\mathsf{brk}}-1} \delta^j \cdot (\vec{\underline{b}}_{\mathsf{pbrk},j} + \vec{\underline{a}}_{\mathsf{pbrk},j} \circ \vec{X}_0 - \lfloor p/B_{\mathsf{brk}}^{j+1} \rceil \cdot \vec{X}_1 - \vec{X}_{j+2}) ; [\![\hat{\underline{\boldsymbol{s}}}]\!], [\![\hat{\underline{\boldsymbol{s}}}']\!], \{[\![\hat{\underline{\boldsymbol{e}}}_{\mathsf{pbrk},j}]\!]\}\Bigg)$$

9. For the validity of the packed key-switching key, the prover $\mathtt{P}$ and the verifier $\mathtt{V}$ invoke the following PIOPs, where $\mathsf{pksk} = (\vec{\boldsymbol{b}}_{\mathsf{pksk}}, \vec{\boldsymbol{a}}_{\mathsf{pksk}})$, $\vec{\underline{a}}_{\mathsf{pksk},j} = \mathtt{NTT}(\boldsymbol{a}_{\mathsf{pksk},j})$, and $\vec{\underline{b}}_{\mathsf{pksk},j} = \mathtt{NTT}(\boldsymbol{b}_{\mathsf{pksk},j})$ for $0 \le j < d_{\mathsf{ksk}}$.

$$\Pi_{\mathsf{ac}}\Bigg(\sum_{j=0}^{d_{\mathsf{ksk}}-1} \delta^j \cdot (\vec{\underline{b}}_{\mathsf{pksk},j} + \vec{\underline{a}}_{\mathsf{pksk},j} \circ \vec{X}_0 - \lfloor p/B_{\mathsf{ksk}}^{j+1} \rceil \cdot \vec{X}_1 - \vec{X}_{j+2}) ; [\![\hat{\underline{\boldsymbol{s}}}']\!], [\![\hat{\underline{\boldsymbol{s}}}]\!], \{[\![\hat{\underline{\boldsymbol{e}}}_{\mathsf{pksk},j}]\!]\}\Bigg)$$

10. For the validity of the public key, the prover $\mathtt{P}$ and the verifier $\mathtt{V}$ invoke the following PIOP, where $\mathsf{ek}' = (\boldsymbol{b}_{\mathsf{ek}'}, \boldsymbol{a}_{\mathsf{ek}'})$, $\vec{\underline{b}}_{\mathsf{ek}'} = \mathtt{NTT}(\boldsymbol{b}_{\mathsf{ek}'})$, and $\vec{\underline{a}}_{\mathsf{ek}'} = \mathtt{NTT}(\boldsymbol{a}_{\mathsf{ek}'})$.

$$\Pi_{\mathsf{ac}}\left(\vec{\underline{b}}_{\mathsf{ek}'} + \vec{\underline{a}}_{\mathsf{ek}'} \circ \vec{X}_0 - \vec{X}_1; [\![\hat{\underline{\boldsymbol{s}}}]\!], [\![\hat{\underline{\boldsymbol{e}}}_{\mathsf{ek}'}]\!]\right)$$

Fig. 12. PIOP for TFHE public keys

E Deferred Proofs

E.1 Proof of Theorem 1

Proof. Let $\mathtt{A}_{\mathsf{S}}$ be a semi-honest adversary that controls the sender. Then, we build a simulator $\mathtt{S}_{\mathsf{S}}$ as follows. First, $\mathtt{S}_{\mathsf{S}}$ generates pk by executing $\mathtt{KeyGen}(\mathsf{pp})$, and sets pk to $\mathtt{Z}$ as the received message. Then, it simulates $\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}$ by computing $\mathtt{Enc}(\mathsf{pk}, 0)$, and sends them to $\mathtt{Z}$ as the received message. By the semantic security of $(\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$, the execution with $\mathtt{S}_{\mathsf{S}}$ in the ideal world is computationally indistinguishable from the execution with $\mathtt{A}_{\mathsf{S}}$ in the real world.

On the other hand, let $\mathtt{A}_{\mathsf{R}}$ be a semi-honest adversary that controls the receiver. Then, we build a simulator $\mathtt{S}_{\mathsf{R}}$ as follows. When $\mathtt{Z}$ sends inputs $\mathsf{pt}_0, \ldots, \mathsf{pt}_{\ell-1}$, $\mathtt{S}_{\mathsf{R}}$ executes $\mathtt{F}_{\mathsf{OCE}}$ with the given inputs and receives $\mathsf{pt}_{\mathsf{out}}$. Finally, it sends $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathtt{Sim}(\mathsf{pk}, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}, \mathsf{pt}_{\mathsf{out}})$ to $\mathtt{Z}$, where $\mathtt{Sim}$ is the circuit privacy simulator for $\mathtt{Eval}$ defined in Definition 9. By Definition 9, the execution with $\mathtt{S}_{\mathsf{R}}$ in the ideal world is computationally indistinguishable from the execution with $\mathtt{A}_{\mathsf{R}}$ in the real world.

E.2 Proof of Theorem 2

Proof. Let C be a circuit that takes ℓ inputs. We construct a PPT simulator $\mathtt{Sim}$ for $\mathtt{Eval}'$ with respect to C as follows.

$$\mathtt{Sim}(\mathsf{pk}, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}, \mathsf{pt}_{\mathsf{out}}) := \mathtt{Sanitize}(\mathsf{pk}, \mathtt{Enc}(\mathsf{pk}, \mathsf{pt}_{\mathsf{out}}))$$

For $\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1} \in \mathsf{L}_{\mathsf{ct}}$, $\mathsf{pk} \in \mathsf{L}_{\mathsf{pk}}$, $\mathsf{pt}_i = \mathtt{Dec}(\mathsf{sk}, \mathsf{ct}_i)$, and $\mathsf{pt}_{\mathsf{out}} = C(\mathsf{pt}_0, \ldots, \mathsf{pt}_{\ell-1})$, let $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathtt{Eval}(\mathsf{pk}, C, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})$, $\mathsf{ct}'_{\mathsf{out}} \leftarrow \mathtt{Enc}(\mathsf{pk}, \mathsf{pt}_{\mathsf{out}})$, $\overline{\mathsf{ct}}_{\mathsf{out}} \leftarrow \mathtt{Eval}'(\mathsf{pk}, C, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})$, and $\overline{\mathsf{ct}}'_{\mathsf{out}} \leftarrow \mathtt{Sim}(\mathsf{sk}, \mathsf{pk}, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}, \mathsf{pt}_{\mathsf{out}})$. Then, we have the following since $\mathsf{ct}_{\mathsf{out}}, \mathsf{ct}'_{\mathsf{out}} \in \mathsf{L}_{\mathsf{ct}}$ and $\mathtt{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{out}}) = \mathtt{Dec}(\mathsf{sk}, \mathsf{ct}'_{\mathsf{out}}) = \mathsf{pt}_{\mathsf{out}}$ due to the correctness condition of homomorphic encryption.

$$\begin{aligned}(\mathsf{sk}, \mathsf{pk}, \overline{\mathsf{ct}}_{\mathsf{out}}) &\equiv \left(\mathsf{sk}, \mathsf{pk}, \mathtt{Sanitize}(\mathsf{pk}, \mathsf{ct}_{\mathsf{out}})\right) \\ &\approx_c \left(\mathsf{sk}, \mathsf{pk}, \mathtt{Sanitize}(\mathsf{pk}, \mathsf{ct}'_{\mathsf{out}})\right) \equiv (\mathsf{sk}, \mathsf{pk}, \overline{\mathsf{ct}}'_{\mathsf{out}})\end{aligned}$$

Thus, the output ciphertexts from $\mathtt{Eval}'$ and the output ciphertexts from $\mathtt{Sim}$ are computationally indistinguishable due to the definition of sanitization.

E.3 Proof of Theorem 3

Proof. Let $\mathtt{A}_{\mathsf{S}}$ be a semi-honest adversary that controls the sender. Then, we can build a simulator $\mathtt{S}_{\mathsf{S}}$ just as in proof of Theorem 1, and additionally set the response from $\mathtt{F}_{\mathsf{ZKAoK}}(\mathsf{R}_{\mathsf{pk}})$ and $\mathtt{F}_{\mathsf{ZKAoK}}(\mathsf{R}_{\mathsf{ct}})$ as 1. By the semantic security of $(\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$, the execution with $\mathtt{S}_{\mathsf{S}}$ in the ideal world is computationally indistinguishable from the execution with $\mathtt{A}_{\mathsf{S}}$ in the real world.

On the other hand, let $\mathtt{A}_{\mathsf{R}}$ be a malicious adversary that controls the receiver. In a standard manner, we assume that $\mathtt{A}_{\mathsf{R}}$ is trivial. Then, we build a simulator $\mathtt{S}_{\mathsf{R}}$ for $\mathtt{A}_{\mathsf{R}}$ as follows:

1. When $\mathtt{Z}$ sends pk, $\mathtt{S}_{\mathsf{R}}$ retrieves it.
2. When $\mathtt{Z}$ sends $(\mathsf{w}_{\mathsf{pk}}, \mathsf{x}_{\mathsf{pk}})$, then $\mathtt{S}_{\mathsf{R}}$ checks if $(\mathsf{w}_{\mathsf{pk}}, \mathsf{x}_{\mathsf{pk}}) \in \mathsf{R}_{\mathsf{pk}}$. If false, then it aborts.
3. When $\mathtt{Z}$ sends $\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}$, $\mathtt{S}_{\mathsf{R}}$ retrieves it.
4. When $\mathtt{Z}$ sends $(\mathsf{x}_{\mathsf{ct}_i}, \mathsf{w}_{\mathsf{ct}_i}) \in \mathsf{R}_{\mathsf{ct}}$ for $0 \leq i < \ell$, $\mathtt{S}_{\mathsf{R}}$ checks if $(\mathsf{x}_{\mathsf{ct}_i}, \mathsf{w}_{\mathsf{ct}_i}) \in \mathsf{R}_{\mathsf{ct}}$. If false, then it aborts.

5. Finally, $\mathtt{S_R}$ executes $\mathtt{F_{OCE}}$ with $\mathsf{pt}_0, \ldots, \mathsf{pt}_{\ell-1}$ and retrieves $\mathsf{pt}_{\mathsf{out}}$. Then, it sends $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathtt{Sim}(\mathsf{pk}, \mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1}, \mathsf{pt}_{\mathsf{out}})$ to $\mathtt{Z}$, where $\mathtt{Sim}$ is the circuit privacy simulator for $\mathtt{Eval}$ defined in Definition 9.

Note that $\mathtt{S_R}$ only aborts exactly when $\mathtt{A_R}$ aborts. Therefore, by Definition 9, the execution with $\mathtt{S_R}$ in the ideal world is computationally indistinguishable from the execution with $\mathtt{A_R}$ in the real world.

E.4 Proof of Theorem 4

Proof. We refer to noise analysis in [CGGI16].

E.5 Proof of Lemma 5

Proof. We refer to [HMS25, Lemma 4] and [BI22, Lemma 15].

E.6 Proof of Lemma 6

Proof. We refer to [HMS25, Lemma 8].

E.7 Proof of Theorem 6

Proof. The correctness naturally follows from the correctness of the original TFHE bootstrapping and Lemmas 5, 6 and 8 to 10.

Now, let $\vec{c}_{\mathsf{in}} = (b_{\mathsf{in}} = -\langle \vec{a}_{\mathsf{in}}, \vec{s} \rangle + (q/t)m + e_{\mathsf{in}}, \vec{a}_{\mathsf{in}})$. To prove the existence of the simulator, we alternatively show that the distribution of the output ciphertext of $\mathtt{Sanitize}$ only depends on pk and m. We proceed using the hybrid argument, where $\mathbf{Hyb}_0$ corresponds to the output of $\mathtt{Sanitize}(\mathsf{evk}, \vec{c}_{\mathsf{in}})$.

Hybrid 1. In this hybrid argument, we replace $\vec{c}_{\mathsf{rand}}$ with a ciphertext with uniform mask. By Lemma 5, the distribution of $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ are computationally indistinguishable.

$\mathbf{Hyb}_1(\boldsymbol{s}, \mathsf{pk}, m, e_{\mathsf{in}})$

$\vec{a}_{\mathsf{rand}} \xleftarrow{\$} \mathbb{Z}_q^N, e_r \leftarrow \mathcal{D}^{\sigma_r, \tau_r}_{\mathtt{EncZero}}|_0$

$\vec{c}_{\mathsf{rand}} \leftarrow (b_{\mathsf{rand}} := -\langle \vec{a}_{\mathsf{rand}}, \vec{s} \rangle + q/t \cdot m + e_{\mathsf{in}} + e_r, \vec{a}_{\mathsf{rand}})$

$\vec{c}_{\mathsf{ks}} := (b_{\mathsf{ks}}, \vec{a}_{\mathsf{ks}}) \leftarrow \mathsf{KeySwitch}(\mathsf{ksk}, \vec{c}_{\mathsf{rand}})$

$\mathsf{ct}_{\mathsf{br}} \leftarrow \mathsf{BlindRotate}(\mathsf{brk}, (0, \vec{a}_{\mathsf{ks}}))$

$\mathsf{ct}_{\mathsf{lin}} \leftarrow \mathsf{RandLinEval}(\mathsf{pk}, \mathsf{ct}_{\mathsf{br}}, X^{\lfloor 2N/q \cdot b_{\mathsf{ks}} \rceil})$

$\vec{c}_{\mathsf{san}} \leftarrow \mathsf{SampleExtract}(\mathsf{ct}_{\mathsf{lin}})$

return $\vec{c}_{\mathsf{san}}$

$\mathbf{Hyb}_2$. In this hybrid, we replace $\mathsf{ct}_{\mathsf{lin}}$. Note that by the correctness of $\mathtt{BlindRotate}$, $\mathsf{ct}_{\mathsf{br}}$ encrypts $\mathsf{tv} \cdot X^{\sum_{i=0}^{n-1} \lfloor 2N/q \cdot a_{\mathsf{ks},i} \rceil s'_i}$, where tv is defined as in Algorithm 1 and $\vec{a}_{\mathsf{ks}} = (a_{\mathsf{ks},0}, \ldots, a_{\mathsf{ks},n-1})$. Therefore, by Lemma 6, the distribution of $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_2$ are computationally indistinguishable.

$\mathbf{Hyb}_2(\boldsymbol{s}, \mathsf{pk}, m, e_{\mathrm{in}})$

$\vec{a}_{\mathsf{rand}} \xleftarrow{\$} \mathbb{Z}_q^N, e_r \leftarrow \mathcal{D}^{\sigma_r,\tau_r}_{\mathtt{EncZero}}|_0$
$\vec{c}_{\mathsf{rand}} \leftarrow (b_{\mathsf{rand}} := -\langle \vec{a}_{\mathsf{rand}}, \vec{s} \rangle + q/t \cdot m + e_{\mathsf{in}} + e_r, \vec{a}_{\mathsf{rand}})$
$\vec{c}_{\mathsf{ks}} := (b_{\mathsf{ks}}, \vec{a}_{\mathsf{ks}}) \leftarrow \mathsf{KeySwitch}(\mathsf{ksk}, \vec{c}_{\mathsf{rand}})$
$\mathsf{ct}_{\mathsf{br}} \leftarrow \mathsf{BlindRotate}(\mathsf{brk}, (0, \vec{a}_{\mathsf{ks}}))$
$m' \leftarrow \mathsf{tv} \cdot X^{\lfloor 2N/q \cdot b_{\mathsf{ks}} \rceil + \sum_{i=0}^{n-1} \lfloor 2N/q \cdot a_{\mathsf{ks},i} \rceil s'_i}$
$\mathsf{ct}_{\mathsf{lin}} \leftarrow \mathsf{RandLinEval}(\mathsf{pk}, \mathsf{ct}_{\mathsf{br}}, 0) + (m', 0)$
$\vec{c}_{\mathsf{san}} \leftarrow \mathsf{SampleExtract}(\mathsf{ct}_{\mathsf{lin}})$
return $\vec{c}_{\mathsf{san}}$

$\mathbf{Hyb}_2$. In this hybrid, we replace m' with $q/t \cdot m$. Note that the output ciphertext $\vec{c}_{\mathsf{san}}$ only encrypts the constant term of $m' = \mathsf{tv} \cdot X^{\lfloor 2N/q \cdot b_{\mathsf{ks}} \rceil + \sum_{i=0}^{n-1} \lfloor 2N/q \cdot a_{\mathsf{ks},i} \rceil s'_i}$, which is $q/t \cdot m$ by the correctness of bootstrapping. Therefore, this change is only syntactic, and the distribution of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_3$ are identical.

$\mathbf{Hyb}_3(\boldsymbol{s}, \mathsf{pk}, m, e_{\mathrm{in}})$

$\vec{a}_{\mathsf{rand}} \xleftarrow{\$} \mathbb{Z}_q^N, e_r \leftarrow \mathcal{D}^{\sigma_r,\tau_r}_{\mathtt{EncZero}}|_0$
$\vec{c}_{\mathsf{rand}} \leftarrow (b_{\mathsf{rand}} := -\langle \vec{a}_{\mathsf{rand}}, \vec{s} \rangle + q/t \cdot m + e_{\mathsf{in}} + e_r, \vec{a}_{\mathsf{rand}})$
$\vec{c}_{\mathsf{ks}} := (b_{\mathsf{ks}}, \vec{a}_{\mathsf{ks}}) \leftarrow \mathsf{KeySwitch}(\mathsf{ksk}, \vec{c}_{\mathsf{rand}})$
$\mathsf{ct}_{\mathsf{br}} \leftarrow \mathsf{BlindRotate}(\mathsf{brk}, (0, \vec{a}_{\mathsf{ks}}))$
$m' \leftarrow \mathsf{tv} \cdot X^{\lfloor 2N/q \cdot b_{\mathsf{ks}} \rceil + \sum_{i=0}^{n-1} \lfloor 2N/q \cdot a_{\mathsf{ks},i} \rceil s'_i}$
$\mathsf{ct}_{\mathsf{lin}} \leftarrow \mathsf{RandLinEval}(\mathsf{pk}, \mathsf{ct}_{\mathsf{br}}, 0) + (q/t \cdot m, 0)$
$\vec{c}_{\mathsf{san}} \leftarrow \mathsf{SampleExtract}(\mathsf{ct}_{\mathsf{lin}})$
return $\vec{c}_{\mathsf{san}}$

$\mathbf{Hyb}_4$. In this hybrid, we remove b_{ks} entirely. This change is only syntactical, as b_{ks} is not used anywhere in $\mathbf{Hyb}_3$. Note that we also remove b_{rand}, since

$$\mathtt{KeySwitch}(\mathsf{ksk}, (b_{\mathsf{rand}}, \vec{a}_{\mathsf{rand}})) = \mathtt{KeySwitch}(\mathsf{ksk}, (0, \vec{a}_{\mathsf{rand}})) + (b_{\mathsf{rand}}, 0)$$

by definition. Therefore, the distribution of $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_4$ are identical.

$\mathbf{Hyb}_4(\mathsf{pk}, m)$

$\vec{a}_{\mathsf{rand}} \xleftarrow{\$} \mathbb{Z}_q^N$
$(\ \cdot\ , \vec{a}_{\mathsf{ks}}) \leftarrow \mathsf{KeySwitch}(\mathsf{ksk}, (0, \vec{a}_{\mathsf{rand}}))$
$\mathsf{ct}_{\mathsf{br}} \leftarrow \mathsf{BlindRotate}(\mathsf{brk}, (0, \vec{a}_{\mathsf{ks}}))$
$m' \leftarrow \mathsf{tv} \cdot X^{\lfloor 2N/q \cdot b_{\mathsf{ks}} \rceil + \sum_{i=0}^{n-1} \lfloor 2N/q \cdot a_{\mathsf{ks},i} \rceil s'_i}$
$\mathsf{ct}_{\mathsf{lin}} \leftarrow \mathsf{RandLinEval}(\mathsf{pk}, \mathsf{ct}_{\mathsf{br}}, 0) + (q/t \cdot m, 0)$
$\vec{c}_{\mathsf{san}} \leftarrow \mathsf{SampleExtract}(\mathsf{ct}_{\mathsf{lin}})$
return $\vec{c}_{\mathsf{san}}$

We conclude that the output of $\texttt{Sanitize}(\mathsf{pk}, \mathbf{c}_{\mathsf{in}})$ is computationally indistinguishable from $\mathbf{Hyb}_4$. Moreover, $\mathbf{Hyb}_4$ only depends on pk, m as desired.

E.8 Proof of Theorem 7

Proof. Our protocol Π_{ppk} consists of Π_{ac}, Π_{lc}, and Π_{nc}, all of which are already proven to be HVZK PIOPs in [HLSS25]. Therefore, our protocol Π_{ppk} is also an HVZK PIOP for the relation $\mathsf{R}_{\mathsf{ppk}}$.

E.9 Proof of Theorem 8

Proof. We refer to noise analysis in [KDE+23, XZW+24].

E.10 Proof of Theorem 9

Proof. This naturally follows from Theorem 4 and Theorem 3.