IND-CPA$^{\mathsf{C}}$: A New Security Notion for Conditional Decryption in Fully Homomorphic Encryption

Bhuvnesh Chaturvedi[1], Anirban Chakraborty[2], Nimish Mishra[1], Ayantika Chatterjee[1], and Debdeep Mukhopadhyay[1]

[1] Indian Institute of Technology Kharagpur, India
[2] Max Planck Institute for Security and Privacy, Germany[⋆]
`bhuvneshchaturvedi2512@kgpian.iitkgp.ac.in`, `anirban.chakraborty@mpi-sp.org`, `nimish.mishra@kgpian.iitkgp.ac.in`, `ayantika@atdc.iitkgp.ac.in`, `debdeep@cse.iitkgp.ac.in`

Abstract. Fully Homomorphic Encryption (FHE) allows a server to perform computations directly over the encrypted data. In general FHE protocols, the client is tasked with decrypting the computation result using its secret key. However, certain FHE applications benefit from the server knowing this result, especially without the aid of the client. Providing the server with the secret key allows it to decrypt all the data, including the client's private input. Protocols such as Goldwasser et. al. (STOC'13) have shown that it is possible to provide the server with the capability of conditional decryption that allows it to decrypt the result of some pre-defined computation and nothing else. While beneficial to an *honest-but-curious* server to aid in providing better services, a *malicious* server may utilize this added advantage to perform active attacks on the overall FHE application to leak secret information. Existing security notions fail to capture this scenario since they assume that only the client has the ability to decrypt FHE ciphertexts. Therefore, in this paper, we propose a new security notion named IND-CPA$^{\mathsf{C}}$, that provides the adversary with access to a conditional decryption oracle. We then show that none of the practical exact FHE schemes are secure under this notion by demonstrating a generic attack that utilizes this restricted decryption capability to recover the underlying errors in the FHE ciphertexts. Given the security guarantee of the underlying (R)LWE hardness problem collapses with the leakage of these error values, we show a full key recovery attack. Finally, we propose a technique to convert any IND-CPA secure FHE scheme into an IND-CPA$^{\mathsf{C}}$ secure FHE scheme. Our technique utilizes Succinct Non-Interactive Argument of Knowledge, where the server is forced to generate a proof of an honest computation of the requested function along with the computation result. During decryption, the proof is verified, and the decryption proceeds only if the verification holds. Both the verification and decryption steps run inside a Garbled Circuit and thus are not observable or controllable by the server.

Keywords: FHE · Conditional Decryption · SNARKs.

[⋆] Work partially done while at IIT Kharagpur

1 Introduction

Fully Homomorphic Encryption [18] has brought a revolution in the cloud computation scenario as it supports computation directly over the encrypted data without the need to decrypt it first. This allows a resource-constrained client to encrypt its data before offloading it to a cloud server[1]. The general use cases of FHE utilize the server for homomorphic computations on encrypted data while the client is tasked with encrypting the messages and decrypting the ciphertexts. The server cannot decrypt the homomorphic computation output as it does not possess the decryption key, which remains in the possession of the client. The server cannot be handed over access to the decryption key, as this would allow the server to decrypt the input ciphertexts, thereby compromising the privacy of the client data. While this setting suffices for most applications, a large class of applications may benefit if the server is allowed the added capability of decrypting the output of some (pre-decided) homomorphic computation.

One such service in the context of email can be spam filtering, where the server hosting the email service can apply a spam filter on the received emails and, based on its outcome, can either forward the email to the intended recipient's mailbox or quarantine it. In the current scenario, if such an application has to be deployed in the cloud, the server would have to apply a spam filter (homomorphically) on the encrypted emails and then send the (still-encrypted) results of the spam filter to the client to decrypt them. The client now resends the decrypted result (spam or not spam) in the clear to the server, which aids it to classify each email as spam or otherwise. Quite clearly, this scheme suffers from inefficiency and network overhead. Furthermore, the client cannot provide the decryption service to the server as that would require providing the server with the secret key, which it can then use to decrypt the emails themselves, thereby compromising privacy. Therefore, a more practical implementation would require a protocol where the server can only decrypt the homomorphically computed output of a pre-defined function (such as a spam filter) and nothing else.

Existing FHE-based protocols assume the server to be *honest-but-curious*, which is expected to abide by the protocol specification of a requested computation. However, in a real-world scenario where these schemes are supposed to be deployed, a question that needs to be pondered upon is: *What if the server is malicious*? Such a server is free to deviate arbitrarily from the protocol specification, which includes tampering with the data and the associated computation. Indeed, the trust bestowed upon the third-party server to respect the laid-down protocols can be a far-reaching assumption in the present world. The potential impact of the existence of a malicious server in practice has led to the concept of Trusted Execution Environments (TEEs) such as Intel SGX [32], Intel TDX [24], AMD SEV [38] and ARM Trustzone [23], which prevents the processes running inside a trusted enclave from being tampered with from the outside untrusted system. Given that these environments assume the server to be actively malicious and the operating system to be compromised, the entire security guarantees are

[1] We use cloud and server interchangeably throughout the paper.

bestowed upon hardware, leaving very little security responsibility on the software. However, given the size of the data and the computational complexity associated with FHE applications, storing the entire FHE data or running the entire FHE computations inside the TEEs is not practical. On the other hand, moving some of the data or computation outside the TEEs makes it vulnerable to malicious perturbations to the data. As FHE ciphertexts are malleable and do not provide data integrity guarantees [14], the possible threat of intentional data tampering by a malicious server is a practical one.

The security of all the practically existing FHE schemes [6,7,12,13,15,16] relies on the underlying (Ring) Learning With Errors or (R)LWE [29,36] hardness problem to provide robust and efficient homomorphic computations on the cloud. While the implementations and message-space domains of these FHE schemes differ, all of them rely on the (R)LWE assumptions and consider the secret key s and the ephemeral error e added to each ciphertext during encryption as private to the client. The intractability of (R)LWE equations comes from the fact that the system of approximate equations is hard to solve without knowledge of the error terms. Therefore, if the error terms corresponding to each ciphertext are leaked to the adversary, it can transform the system of approximate equations to exact equations and then solve it to retrieve the secret key.

The security of FHE schemes has been studied under various security notions, such as IND-CPA [20], IND-CCA1 [33], IND-CCA2 [35], IND-CPA$^{\mathsf{D}}$ [27], IND-CPA$^{\mathsf{rD}}$ [30], and Func-CPA [1], where each notion allows an adversary access to black-box oracles, which it can query on inputs of its choice. These notions assume that only the client can perform decryption using the decryption key that it possesses. Thus, they fail to capture the aforementioned scenario where the server is granted the additional capability to decrypt the result of some fixed computations and nothing else. Moreover, providing access to such functionality can grant additional powers to a malicious server that can abuse it to perform attacks on the underlying FHE schemes. For example, the server can add crafted perturbations into the resultant ciphertexts of the homomorphic operation. It can then locally decrypt these tampered ciphertexts to observe the effect of the added perturbations on the decrypted result. Therefore, it is necessary to evaluate the security of such constructions under a separate notion. Throughout this paper, we refer to this notion as IND-CPA$^{\mathsf{C}}$ to encompass the ability of the adversary to perform conditional decryptions. Given that a variety of FHE schemes exist in practice that differ due to the underlying mathematical hard problem and their choice of parameters, their robustness varies against an adversary with the capability of conditional decryptions. In Sect. 3.3, we introduce the IND-CPA$^{\mathsf{C}}$ notion in detail and highlight how it can be used to compare the security of practical FHE schemes.

Conditional Decryption in Practice. As observed by Goldwasser et al. [19], the problem of granting the server the ability of conditional decryption can be solved in practice using Functional Encryption (FE) [5]. Given an encryption $\hat{x}$ of some message x and a secret key fsk_f corresponding to some function

f, FE allows the server to obtain the value of $f(x)$ in the clear and nothing else. In terms of FHE applications, this is akin to decrypting the output of a homomorphic computation function for a particular input ciphertext without leaking any information about the input. However, FE schemes suffer from the limitation that the number of secret keys increases linearly with the number of functions to be evaluated. The authors of [19] overcame this limitation by using an FHE scheme, along with an Attribute-based Encryption (ABE) scheme [21] and a Garbled Circuit (GC) scheme [4, 41], to construct a *succinct functional encryption scheme* . The central idea is to provide the server with the FHE decryption function inside a garbled circuit, where the client's secret key is embedded within the circuit itself. The underlying FHE scheme ensures that the inputs x (say the email) upon which a function f (say a spam filter) is to be evaluated remain private, while the ABE scheme ensures that the server can only decrypt encryption of $f(x)$ (spam or not spam), say $\widehat{f(x)}$, and nothing else. The security of this succinct FE scheme builds upon the security guarantees of FHE, ABE, and GC and proves to be secure if the underlying FHE, ABE, and GC schemes are secure. However, while the security guarantees of this construction hold against a *semi-honest* server, it falters in the presence of a *malicious* server that can deviate arbitrarily from the protocol specification. The reason is that while the black-box primitives used to construct the overall protocol remain secure, combining them opens up new avenues for exploitation.

1.1 Contributions

In this paper, we make the following contributions:

1. **Introducing IND-CPA$^{\mathbf{C}}$ security notion:** We introduce a new security notion for FHE schemes in the presence of a malicious adversary that has access to a conditional decryption oracle that allows it to decrypt the result of some homomorphic computation (including modifying the same) but not any other ciphertexts. We denote this security notion with $\mathtt{IND\text{-}CPA}^{\mathtt{C}}$ where $\mathtt{C}$ denotes the ability of the adversary to perform conditional decryptions. We further highlight how the existing notions fail to capture this scenario, which justifies the introduction of a new security notion.
2. **Error and Key Recovery Attack:** We propose a generic full key recovery attack on (R)LWE-based exact FHE schemes. Our attack first recovers the underlying error values in the FHE ciphertexts and removes them to build a system of exact equations, which is then solved to recover the underlying secret key. Our attack works on both the secret and public key variants of the (R)LWE-based exact FHE schemes.
3. **Constructing IND-CPA$^{\mathbf{C}}$-secure FHE:** We propose a method that can be used to convert any IND-CPA secure FHE scheme into an IND-CPA$^{\mathtt{C}}$ secure FHE scheme. Our methods require the server to generate proofs of honest evaluation of the pre-defined function. The proof is then verified, and the resultant ciphertext is decrypted only when the verification holds. Both verification and decryption are done inside a garbled circuit, which prevents

the server from tampering with these values without breaking the security guarantees of the garbling scheme itself.

1.2 Organization

The rest of the paper is organized as follows: Sect. 2 introduces the notations and the necessary background . Sect. 3 explains our threat model and formally defines the security notion of $\texttt{IND-CPA}^{\texttt{C}}$. Sect. 4 explains our error and key recovery attack on (R)LWE-based exact FHE schemes. Sect. 5 explains how the IND-CPA secure FHE schemes can be converted into $\texttt{IND-CPA}^{\texttt{C}}$ secure FHE schemes. Sect. 6 outlines future work, and Sect. 7 concludes our paper.

2 Notations and Background

2.1 Notations

Boldface letters like $\mathbf{a}$ are used to denote vectors while a_i denote their i^{th} coefficient. $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ denotes a set of integers, rationals, and reals, resp. S^+ denotes a subset of some set S containing only the positive elements, and S_q denotes a finite set containing elements from $[-\frac{q}{2}, \frac{q}{2} - 1]$. We denote with S^k a set of vectors of dimension k where the coefficients of each vector are uniformly sampled from set S. $x \in X$ denotes that an element x belongs to a set X. For some finite set S, $s \overset{\$}{\leftarrow} S$ denotes that s has been sampled uniformly from S, while $\mathbf{s} \overset{\$}{\leftarrow} S$ denotes that each element of a vector $\mathbf{s}$ has been sampled uniformly from S. We write $s \leftarrow \mathcal{N}_{\mu,\sigma}$ to denote that s has been sampled from a Gaussian distribution $\mathcal{N}_{\mu,\sigma}$ parameterized by mean μ and s.d. σ. We use c and $\hat{m}$ interchangeabily to denote an FHE ciphertext that encrypts some message m. $\lfloor\cdot\rfloor$, $\lceil\cdot\rceil$, $\lfloor\cdot\rceil$ and $[\cdot]_q$ denotes, resp., the floor, ceil, rounding, and residue modulo q operations, while $|\cdot|$ denotes the bit length of an element. All logs are in base-2 unless stated otherwise. We denote with PPT a probabilistic polynomial time machine.

2.2 LWE Problem

Learning With Errors [36] problem is based on the addition of random errors (sampled from a Gaussian distribution) to each equation in a system of equations, thus turning it into a system of approximate equations, as follows:

$$\begin{aligned}
a_{11}s_1 + a_{12}s_2 + \ldots + a_{1k}s_k &\approx b_1 \bmod q \\
a_{21}s_1 + a_{22}s_2 + \ldots + a_{2k}s_k &\approx b_2 \bmod q \\
&\vdots \\
a_{m1}s_1 + a_{m2}s_2 + \ldots + a_{mk}s_k &\approx b_m \bmod q
\end{aligned}$$

For brevity, let $k > 1$ be an integer and $\mathbf{s} \overset{\$}{\leftarrow} S$, where $S \in \mathbb{Z}^k$. An LWE sample is denoted by a tuple $(\mathbf{a}, b) \in \mathbb{Z}_q^k \times \mathbb{Z}_q$, where $\mathbf{a} \overset{\$}{\leftarrow} \mathbb{Z}_q^k$ and $b = \mathbf{a} \cdot \mathbf{s} + e \in \mathbb{Z}_q$.

Here $e \leftarrow \mathcal{N}_{0,\sigma}$ (with $\sigma \in \mathbb{R}^+$) is an error value. In the ring version denoted RLWE [29], the ciphertexts are of the form $(\mathbf{a}, \mathbf{b}) \in R_q^2$, where each component is defined over ring $R_q = \mathbb{Z}_q/(x^k + 1)$. Here $x^k + 1$ is an irreducible polynomial over $\mathbb{Q}$ provided the ring dimension k is a power of 2. (R)LWE problem has the following two variants -

– *Search problem*: having access to polynomially many (R)LWE samples, retrieve $\mathbf{s}$.
– *Decision problem*: distinguish between LWE (resp. RLWE) samples and uniformly random samples drawn from $\mathbb{Z}_q^k \times \mathbb{Z}_q$ (resp. $\mathcal{R}_q \times \mathcal{R}_q$).

Both versions are considered hard to solve, even for a quantum computer. However, once these error values are recovered, they can be removed from the corresponding ciphertexts to obtain a system of exact equations which can then be trivially solved.

2.3 Fully Homomorphic Encryption

A fully homomorphic encryption scheme FHE defined over a message space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$ is a tuple of PPT algorithms (`FHE.KeyGen`, `FHE.Enc`, `FHE.Eval`, `FHE.Dec`) where the algorithms are defined as follows[2]:

1. `FHE.KeyGen(`1^λ`)`: Upon input the security parameter λ, `FHE.KeyGen` outputs a pair of keys (`pk`, `sk`) where `pk` and `sk` are the public key and secret key, resp.
2. `FHE.Enc(`pk`,`x`)`: Upon input the public key `pk` and a message x, `FHE.Enc` outputs a ciphertext `c`$= \hat{x}$.[3]
3. `FHE.Eval(`pk`,`f`,`c_1`,`$\cdots$`,`c_n`)`: Upon input the public key pk, a circuit $f{:}\mathcal{M}^n \rightarrow \mathcal{M}$ and a set of ciphertexts $(c_1, \cdots, c_n)$ that encrypt the messages $x_s = (x_1, \cdots, x_n)$, `FHE.Eval` outputs a ciphertext $c = \widehat{f(x_s)}$.
4. `FHE.Dec(`sk`,`c`)`: Upon input the secret key sk and a ciphertext $c = \hat{x}$, `FHE.Dec` outputs a message x.

Correctness: An (exact) FHE scheme is said to be correct if, for all $m \in \mathcal{M}$, $c \in \mathcal{C}$, and f that can be efficiently evaluated, it holds that:

$$\Pr\left[\texttt{FHE.Dec}(sk, c_r) = f(x_1, \cdots, x_n)\right] \geq 1 - negl(\lambda).$$

where $c_r =$`FHE.Eval(`pk`,`f`,`c_1`,`$\cdots$`,`c_n`)`.

Security: An FHE scheme is said to be secure if, for all $m \in \mathcal{M}$, $c \in \mathcal{C}$, and a PPT adversary $\mathcal{A}$, c does not leak any information about its underlying message m to $\mathcal{A}$ beyond what is publicly known.

[2] We explain the working principles of practical (R)LWE-based FHE schemes in Appendix A.

[3] We use (a, b) to generically denote the ciphertexts for (R)LWE-based FHE schemes.

2.4 Garbled Circuits

A garbling scheme `Gb` [4,25] defined over a family of circuits $\mathcal{F} = \{f_n\}$, where f_n is a set of boolean circuits that takes n-bit inputs, is a tuple of PPT algorithms (`Gb.Garble`, `Gb.Enc`, `Gb.Eval`) where the algorithms are defined as follows:

1. $\texttt{Gb.Garble(}1^\lambda, f\texttt{)}$: Upon input the security parameter λ and a circuit $f \in \mathcal{F}$, `Gb.Garble` outputs a garbled circuit $\mathit{\Gamma}$ and a secret key sk, where sk contains a pair of garbling labels $(L_i^0,\ L_i^1)$ for each bit i of the n-bit input.
2. $\texttt{Gb.Enc(}sk\texttt{,}x\texttt{)}$: Upon input the secret key sk and a message x, where $x \in \{0,1\}^n$, `Gb.Enc` outputs an encoding c, where $|c|$ is independent of the size of f. The encoding c is of the form $(L_i^{b_i})$ for each bit b_i of x.
3. $\texttt{Gb.Eval(}\mathit{\Gamma}\texttt{,}c\texttt{)}$: Upon input the garbled circuit $\mathit{\Gamma}$ and an encoding c, `Gb.Eval` outputs a value $y = f(x)$.

Correctness: A Garbling scheme is said to be correct if for all $\mathit{\Gamma}$ that is a garbling of a circuit f, and for all messages x, it holds that:

$$\Pr\begin{bmatrix} (\mathit{\Gamma}, sk) \leftarrow \texttt{Gb.Garble(}1^\lambda, f\texttt{);} \\ c \leftarrow \texttt{Gb.Enc(}sk\texttt{,}x\texttt{);} \\ \texttt{Gb.Eval(}\mathit{\Gamma}\texttt{,}c\texttt{)} = f(x) \end{bmatrix} \geq 1 - negl(\lambda).$$

Security: A garbling scheme is said to be secure against a PPT adversary $\mathcal{A}$ if it neither leaks x nor f to $\mathcal{A}$ and thus ensures both input and circuit privacy.

2.5 Two-outcome Attribute-Based Encryption

A two-outcome attribute-based encryption scheme `ABE` [19] defined over a class of predicates $\mathcal{P} = \{p_n\}$, which are represented as circuits that take n-bit inputs, and a message space $\mathcal{M}$, is a tuple of PPT algorithms (`ABE.Setup`, `ABE.KeyGen`, `ABE.Enc`, `ABE.Dec`) where the algorithms are defined as follows:

1. $\texttt{ABE.Setup(}1^\lambda\texttt{)}$: Upon input security parameter λ, `ABE.Setup` outputs a pair of keys (`fmpk`, `fmsk`) where `fmpk` and `fmsk` are the master public key and master secret key, resp.
2. $\texttt{ABE.KeyGen(}fmsk\texttt{,}p\texttt{)}$: Upon input the master secret key $fmsk$ and a predicate $p \in \mathcal{P}$, `ABE.KeyGen` outputs a key fsk_p corresponding to p.
3. $\texttt{ABE.Enc(}fmpk\texttt{,}x\texttt{,}M_0\texttt{,}M_1\texttt{)}$: Upon input the master public key $fmpk$, an attribute $x \in \{0,1\}^n$ and two messages $M_0,\ M_1 \in \mathcal{M}$, `ABE.Enc` outputs a ciphertext c.
4. $\texttt{ABE.Dec(}fsk_p\texttt{,}c\texttt{)}$: Upon input the secret key fsk_p corresponding to predicate p, and a ciphertext c, `ABE.Dec` outputs M_0 if $p(x) = 0$ and it outputs M_1 if $p(x) = 1$.

Correctness: A two-outcome ABE scheme is said to be correct if for all predicates $\mathcal{P} = \{p_n\}$, all attributes $x \in \{0,1\}^n$, and messages $M_0, M_1 \in \mathcal{M}$, it holds that:

$$\Pr\begin{bmatrix} (\texttt{fmpk}, \texttt{fmsk}) \leftarrow \texttt{ABE.Setup(}1^\lambda\texttt{);} \\ fsk_p \leftarrow \texttt{ABE.KeyGen(}fmsk\texttt{,}p\texttt{);} \\ c \leftarrow \texttt{ABE.Enc(}fmpk\texttt{,}x\texttt{,}M_0\texttt{,}M_1\texttt{);} \\ \texttt{ABE.Dec(}fsk_p\texttt{,}c\texttt{)} = M_{p(x)} \end{bmatrix} \geq 1 - negl(\lambda).$$

Security: A two-outcome ABE scheme is said to be secure against a PPT adversary $\mathcal{A}$ if it leaks nothing about M_0 to $\mathcal{A}$ if $p(x) = 1$ and about M_1 if $p(x) = 0$. However, the attribute x is not hidden by ABE and may leak irrespective of the value of $p(x)$.

Based on the aforementioned security guarantee, one may observe that which M will be revealed as output depends on the control variable $p(x) = b$ which is computed on the server side and thus is available to the server. A malicious server can alter its value, say by setting $p(x) = \bar{b}$, to force it to output $M_{\bar{b}}$.

2.6 Functional Encryption

A functional encryption scheme FE defined over a class of functions $\mathcal{F} = \{f_n\}$, which are represented as boolean circuits that take n-bit inputs, is a tuple of PPT algorithms (`FE.Setup`, `FE.KeyGen`, `FE.Enc`, `FE.Dec`) where the algorithms are defined as follows:

1. `FE.Setup`(1^λ): Upon input security parameter λ, `FE.Setup` outputs a pair of keys (`fmpk`, `fmsk`) where `fmpk` and `fmsk` are the master public key and master secret key, resp.
2. `FE.KeyGen`$(fmsk, f)$: Upon input the master secret key $fmsk$ and a function $f \in \mathcal{F}$, `FE.KeyGen` outputs a key fsk_f corresponding to f.
3. `FE.Enc`$(fmpk, x)$: Upon input the master public key $fmpk$ and an input $x \in \{0,1\}^n$, `FE.Enc` outputs a ciphertext c.
4. `FE.Dec`(fsk_f, c): Upon input the secret key fsk_f corresponding to predicate f, and a ciphertext c, `FE.Dec` outputs a value $y = f(x)$.

Correctness: An FE scheme is said to be correct if for all functions $f \in \mathcal{F}$, and for all inputs $x \in \{0,1\}^n$, it holds that:

$$\Pr\left[\begin{array}{c}(\mathtt{fmpk},\mathtt{fmsk}) \leftarrow \mathtt{FE.Setup}(1^\lambda);\\ fsk_f \leftarrow \mathtt{FE.KeyGen}(fmsk, f);\\ c \leftarrow \mathtt{FE.Enc}(fmpk, x);\\ \mathtt{FE.Dec}(fsk_f, c) = f(x)\end{array}\right] \geq 1 - negl(\lambda).$$

Security: An FE scheme is said to be secure against a PPT adversary $\mathcal{A}$ if $\mathcal{A}$ learn nothing about the input x apart from the computational result $f(x)$, for a circuit corresponding to f for which a key was issued (the adversary can learn this circuit itself).

The security guarantee of FE is different from ABE since, in the case of ABE, the adversary can know the value of x since it is public. On the other hand, in the case of FE, the value of x remains hidden from the adversary. In [19], this is ensured by encrypting the value of x using some FHE scheme.

2.7 Succinct Non-Interactive Argument of Knowledge

A Succinct Non-Interactive Argument of Knowledge `SNARK` defined over a binary relation $\mathcal{R}$ is a tuple of PPT algorithms (`SNARK.Setup`, `SNARK.Prover`, `SNARK.Verifier`) where the algorithms are defined as followed:

1. $\texttt{SNARK.Setup(}1^\lambda\texttt{, }CS\texttt{)}$: Upon input security parameter λ and a constraint system CS, `SNARK.Setup` outputs a common reference string `crs` and a verification state `st`.
2. `SNARK.Prover(crs,x,w)`: Upon input a common reference string `crs`, a statement $\mathbf{x}$ and a witness $\mathbf{w}$, `SNARK.Prover` outputs a proof π.
3. $\texttt{SNARK.Verifier(st,x,}\pi\texttt{)}$: Upon input a verification state `st`, a statement $\mathbf{x}$ and a proof π, `SNARK.Verifier` outputs a verification bit $b \in \{0,1\}$.

Completeness: A SNARK scheme is said to be complete if and only if for any security parameter λ, statement $\mathbf{x}$ and corresponding witness $\mathbf{w}$, and constraint instance CS, it holds that:

$$\Pr\left[\texttt{SNARK.Verifier}(st,x,\pi)=1 \;\middle|\; \begin{array}{c} CS(x,w)=1 \\ (crs,st)\leftarrow \texttt{SNARK.Setup}(1^\lambda,CS) \\ \pi \leftarrow \texttt{SNARK.Prover}(crs,x,w) \end{array}\right]=1$$

Knowledge Soundness: A SNARK scheme satisfies knowledge soundness if and only if for any PPT algorithm $\texttt{SNARK.Prover}^*$, there exists a PPT extractor `Extr` such that for any security parameter λ, a constraint instance CS and state z, it holds that:

$$\Pr\left[\begin{array}{c} \texttt{SNARK.Verifier}(st,x,\bar{\pi})=1 \\ \wedge \\ CS(x,w^*)\neq 1 \end{array} \;\middle|\; \begin{array}{c} (crs,st)\leftarrow \texttt{SNARK.Setup}(1^\lambda,CS) \\ (\bar{\pi},x)\leftarrow \texttt{SNARK.Prover}^*(crs;z) \\ w^*\leftarrow \texttt{Extr}(crs;z) \end{array}\right]\leq \mathrm{negl}(\lambda)$$

3 Security Notions and Threat Model

In this section, we introduce our threat model. We then highlight how the existing security notions fail to capture the scenario of conditional decryption in practice. Finally, we define our IND-CPA$^{\mathsf{C}}$ security notion.

3.1 Threat Model

The security of public-key encryption schemes, including FHE, in the presence of an adversary is often evaluated under the IND-CPA, IND-CCA1, and IND-CCA2 security notions. These notions are defined in the form of a game between a challenger $\mathcal{C}h$ and a PPT adversary $\mathcal{A}$, where $\mathcal{A}$ is given access to various oracles that it can query upon inputs of its choice. Under the IND-CPA notion, $\mathcal{A}$ is provided access to only an encryption oracle, while under the IND-CCA1 and IND-CCA2 notions, $\mathcal{A}$ is additionally provided access to an unrestricted decryption oracle.[4] Given that public-key encryption schemes are required to be at least IND-CPA secure, existing FHE schemes [6, 7, 13, 15, 16] fall under this security model [14]. On the other hand, the malleability requirement of FHE

[4] $\mathcal{A}$ is only allowed to make polynomially many queries to each oracles.

ciphertexts prevents them from being IND-CCA2 secure [14]. Moreover, none of the existing FHE schemes [6, 7, 12, 13, 15, 16] are even IND-CCA1 secure since they require publishing encryption of the secret key to aid in certain ciphertext-maintenance operations such as bootstrapping and relinearization. Since there is no restriction on the type of ciphertext that can be used to query the decryption oracle, an adversary can trivially obtain the secret key by directly querying the decryption oracle on the publicly available encryption of the secret key [17].

It is well known in the literature that providing access to an unrestricted decryption oracle that can decrypt any ciphertext of the adversary's choice will break the security of all practically existing FHE schemes. However, there exist practical scenarios where the adversary often obtains access to a restricted decryption oracle (for example, the spam filtering application from Sect. 1). Such a scenario occurs in protocols where the server is provided the ability to decrypt the result $\widehat{f(x)}$ of some fixed computation f, and nothing else. However, given the decryption operation runs on the server, which itself is malicious, it is free to tamper with $\widehat{f(x)}$. Moreover, the server can locally decrypt the perturbed $\widehat{f(x)}$ to observe the effect of the perturbation on the final result; whether the result changes or remains the same. In Sect. 4, we show how an adversary (the server in our case) can exploit this observation to perform full key recovery attacks on practical FHE schemes. However, existing security notions fail to capture this scenario in practice owing to the fact that these notions assume that only the client has the ability to perform a decryption operation.

Thus, in this paper, we introduce a new security notion termed IND-CPA$^{\mathsf{C}}$ that encompasses the aforementioned practical scenario. Our notion allows a PPT adversary $\mathcal{A}$ access to a conditional decryption oracle that it can query on ciphertexts c encrypting $f(x)$ for some fixed function f.[5] It is additionally allowed to query the oracle on a perturbed version of c. Given existing FHE schemes differ from each other based on the choice of the underlying hard problem and instantiation parameters, the number of perturbations required for a successful key recovery also differs across multiple FHE schemes. Therefore, this notion can aid in assessing the resistance of practical FHE schemes against full key recovery attacks. Finally, given the presence of the conditional decryption setting on the server, it can covertly carry out the attack without getting detected by the client. The reason is that the client does not get access to the decryption result, which remains in the possession of the server. Thus, this setting allows the server to behave maliciously without the risk of getting exposed.[6] We now justify the need for a new security notion by explaining how the existing notions relevant to the FHE schemes fail to encompass our threat model.

[5] We assume this function f to be one-way; otherwise, $\mathcal{A}$ can easily compute $f^{-1} \cdot f(x)$ to obtain x, thereby leaking the input.

[6] This is unlike the existing notions where the client performs the decryption and can detect if the decryption outputs an incorrect result, signifying that the ciphertext returned by the server is malformed, and so can abort the protocol.

3.2 Need For New Security Notion

The necessity to define a new security notion arises from the fact that there exist cases where the adversary is given access to a conditional decryption oracle, which we refer to as $\mathtt{Dec}^{\mathtt{C}}\mathtt{()}$. In other words, the adversary can obtain the decryptions of ciphertexts c only if they encrypt the result of some pre-defined function f for which the conditional decryption functionality is provided. These cases cannot be modeled by the IND-CCA1 notion, which allows an adversary to query the decryption oracle on any ciphertext of its choice. Moreover, IND-CCA1 attacks such as [10,34] utilize malformed ciphertexts, i.e., ciphertexts not generated through an honest execution of `FHE.Enc()` or `FHE.Eval()` algorithms. Such ciphertexts will be rejected by $\mathtt{Dec}^{\mathtt{C}}\mathtt{()}$ since they are not encryptions of $f(x)$.

There have been further works that, similar to ours, try to capture more realistic deployment settings of FHE-based protocols by proposing different security notions. In this section, we highlight how these notions, namely IND-CPA$^{\mathtt{D}}$ [9,11], IND-CPA$^{\mathtt{rD}}$ [30], and Func-CPA [1,2], fail to capture the scenario of the conditional decryption.

IND-CPA$^{\mathbf{D}}$. Introduced in [27], IND-CPA$^{\mathtt{D}}$ assumes a passive adversary that follows the protocol specification honestly. Such an adversary is provided access to a restricted decryption oracle that can only decrypt ciphertexts generated through an honest evaluation of `FHE.Enc()` or `FHE.Eval()` algorithms. Furthermore, this notion allows the adversary to obtain decryption of $\widehat{f(x)}$ for any $f \in \mathcal{F}$, as long as it is obtained by homomorphically computing f on $\hat{x}$. Recently, the authors of [9] and [11] showed IND-CPA$^{\mathtt{D}}$ attacks on exact FHE schemes such as [6, 7, 13, 15, 16]. However, these attacks are either infeasible to carry out in practice, or they do not work for FHE schemes which perform bootstrapping after each homomorphic operation. For example, the attack in [11] takes approximately 300 years to carry out on bootstrapped FHE schemes such as [13] and [15]. On the other hand, the attack in [9] does not work if a bootstrapping operation is performed before the decryption operation, as it resets the noise level and prevents the occurrence of decryption failure, which is exploited in this attack. Finally, this notion assumes that the decryption operation will run on the client side for any function, while our threat model assumes the server itself has the decryption capability (although conditional) for a specific function.

IND-CPA$^{\mathbf{rD}}$. Introduced in [30], IND-CPA$^{\mathtt{rD}}$ is an extension of IND-CPA$^{\mathtt{D}}$ notion where the adversary is allowed to tamper with the ciphertext obtained from querying the `FHE.Enc()` oracle. Before proceeding with the evaluation of the requested function f (chosen by the adversary), the `FHE.Eval()` oracle first ensures the validity of the input ciphertexts with the help of the `FHE.Dec()` oracle. Evaluation and decryption proceeds, and the adversary receives the decrypted value only if all the input ciphertexts are found to be valid; otherwise, it receives a $\perp$. One can observe that under this notion, the adversary is allowed to choose the function f to be evaluated and whose result it wants to obtain. On the other hand, in the conditional decryption setting, f is fixed at the beginning for which the decryption capability is provided. Moreover, their setting requires the decryption result to be shared with the evaluator (during the validity-checking step). However, this setting can lead to IND-CCA1 style attacks if the evaluator turns out to be malicious. Finally, their attacks do not apply to symmetric key FHE setting. The reason is that their setting assumes the encrypting party (which uses the public key to perform encryptions) to be malicious, which is trying to recover the secret decryption key. On the other hand, in the symmetric key setting, the encrypting and decrypting parties are the same entity, which already possesses the secret decryption key.

Func-CPA. Introduced in [2], Func-CPA notion is defined for *client-aided protocols* where a portion of computation is offloaded to the client-side. In this notion, the adversary is provided access to an oracle that, upon input a ciphertext $\hat{x}$ and a function f, first decrypts $\hat{x}$ to obtain the underlying message x, evaluates $f(x)$, and then encrypts $f(x)$ before returning it back to the adversary. Thus, in this notion, an adversary is allowed to see only ciphertexts and no clear data. On the other hand, this output is trivially leaked to the adversary by the conditional decryption oracle itself, albeit for a fixed function not chosen by the adversary.

The comparisons made above highlight that none of the existing security notions capture the setting where the server is provided an additional capability of performing local (conditional) decryptions. This necessitates the introduction of our IND-CPA$^{\mathsf{C}}$ notion to better understand the security implications of such scenarios in practice. We now explain our notion in details.

3.3 IND-CPA$^{\mathsf{C}}$ Notion

In this section, we provide an indistinguishability-based definition of our notion. We begin with the generic IND-CPA notion; the adversary chooses a pair of messages (m_0, m_1) of equal length and sends them to the challenger. The challenger samples a uniform bit $b \in \{0,1\}$ and returns back the encryption of m_b to the adversary. The goal of the adversary is to determine whether the received challenge ciphertext c is an encryption of m_0 or m_1 with a probability significantly better than $\frac{1}{2}$. To derive IND-CPA$^{\mathsf{C}}$, the adversary is additionally provided access to an evaluation oracle that can be queried on any function belonging to $\mathcal{F}$ (where $\mathcal{F}$ is a class of functions that the underlying FHE scheme supports). Finally, the adversary is also provided access to a conditional decryption oracle that can only be queried on a ciphertext encrypting $f(x)$ for some fixed function f where x is some plaintext message the adversary can choose. In practice, the function f is chosen by the client for which conditional decryption is granted to the server. Moreover, the conditional decryption oracle does not allow the server to obtain decryption of $g(x)$, where $g \in \mathcal{F}$ and $g \neq f$. Finally, the adversary only has access to the conditional decryption oracle before it receives the challenge ciphertext c. We now formally define our notion.

Definition 1. *Let `FHE = (FHE.KeyGen, FHE.Enc, FHE.Eval, FHE.Dec)` be an exact public-key fully homomorphic encryption scheme as defined in Sect. 2.3 over a message space $\mathcal{M}$, a ciphertext space $\mathcal{C}$, and a class of functions $\mathcal{F}$. We define an experiment $\Psi_{\texttt{IND-CPA}^{C},\mathcal{A}}(1^{\lambda})$ over the security parameter λ between a challenger $\mathcal{C}h$ and a PPT adversary $\mathcal{A}$, where $\mathcal{A}$ is given access to the following oracles:*

- *An encryption oracle `Enc()` that takes as input a message $x \in \mathcal{M}$, executes `FHE.Enc(pk, x)` to obtain a ciphertext $c \in \mathcal{C}$, and returns it back.*
- *An evaluation oracle `Eval()` that takes as input ciphertexts $(c_i, \ldots, c_j) \in \mathcal{C}$ and a function $f \in \mathcal{F}$, executes `FHE.Eval(pk, f, c_i, ..., c_j)` to obtain a ciphertext $c_r \in \mathcal{C}$, and returns it back.*
- *A conditional decryption oracle $\texttt{Dec}^{C}\texttt{()}$ defined for a fixed function $f \in \mathcal{F}$ that, given an FHE ciphertext c, checks whether `FHE.Dec(sk, c)` $= f(x)$, i.e., c is an encryption of $f(x)$, for any $x \in \mathcal{M}$ and returns the result of `FHE.Dec(sk, c)` to $\mathcal{A}$ if true, otherwise outputs a $\perp$.*

IND-CPA$^{\mathbf{C}}$ experiment $\Psi_{\texttt{IND-CPA}^{C},\mathcal{A}}(1^{\lambda})$*:*

Query Phase

1. *The adversary $\mathcal{A}$ queries the `Enc()` oracle on its messages $(m_i, \ldots, m_j)$ where $i \leq j$ and receives their corresponding FHE ciphertexts $(c_i, \ldots, c_j)$.*
2. *The adversary $\mathcal{A}$ queries the `Eval()` oracle on any function $g \in \mathcal{F}$ of its choice and the ciphertexts $(c_i, \ldots, c_j)$ and receives as output a ciphertext c_r.*
3. *The adversary $\mathcal{A}$ queries the $\texttt{Dec}^{C}\texttt{()}$ oracle on the ciphertext c_r. This oracle outputs a $\perp$ if $g \neq f$, where $f \in \mathcal{F}$ is the function for which conditional decryption is allowed. When $g = f$, $\mathcal{A}$ receives as output the value $f(m_i, \ldots, m_j)$. $\mathcal{A}$ also queries the $\texttt{Dec}^{C}\texttt{()}$ oracle on a series of ciphertexts $c'_r = c_r + \epsilon$, where ϵ is a crafted perturbation added to the ciphertext c_r which varies across each query. In this case, the decrypted value may or may not be equal to $f(m_i, \ldots, m_j)$.*

Challenge Phase

1. *The adversary $\mathcal{A}$ chooses two messages $(m_0, m_1) \in \mathcal{M}$ of equal length and sends them to the challenger $\mathcal{C}h$.*
2. *The challenger $\mathcal{C}h$ samples a bit $bt \xleftarrow{\$} \{0,1\}$ and provides $\mathcal{A}$ with a ciphertext $c_{bt} =$ `FHE.Enc(pk, m_bt)`.*

Output Phase

1. *The adversary $\mathcal{A}$ is not allowed to query the $\texttt{Dec}^{C}\texttt{()}$ oracle. It outputs a bit bt'.*
2. *The game outputs 1 if $bt' = bt$, and 0 otherwise.*

The construction IND-CPA^{C} is secure if for all PPT adversaries $\mathcal{A}$, there exists a negligible function `negl` such that, for all λ,

$$\texttt{Pr}[\Psi_{\texttt{IND-CPA}^C,\mathcal{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \texttt{negl}(\lambda) \tag{1}$$

where the probability is taken over the randomness used by $\mathcal{A}$ and the randomness used by the game (in `FHE.Gen` *and* `FHE.Enc`*).*

In Section 4.3, we show that none of the practical (R)LWE-based FHE schemes [6,7,13,15,16] are secure under the IND-CPA$^{\mathsf{C}}$ notion by demonstrating a full key recovery attack.

Conditional Decryption with Malicious Server

The idea of conditional decryption has shown to be realizable in practice in [19], which allows a server to decrypt the homomorphic ciphertext $\widehat{f(x)}$, where f is a pre-defined function. The protocol[7] provides the server with the FHE decryption circuit and the corresponding secret key by embedding it into a garbled circuit. However, to evaluate this garbled circuit and in turn decrypt the value of $\widehat{f(x)}$, the server is required to know the correct set of labels that corresponds to the individual bits of $\widehat{f(x)}$. The client cannot provide the labels corresponding to these bits, as they are determined by the server after it obtains $\widehat{f(x)}$ on its end. To overcome this issue, the client utilizes an ABE scheme by encrypting the set of labels corresponding to each bit of $\widehat{f(x)}$ to obtain ABE ciphertexts $\{c_1, \ldots, c_\ell\}$. It then generates a set of ABE decryption keys fsk_f^i where each key corresponds to the individual gates that make up the circuit evaluating f. The client sends both the FHE ciphertexts $\hat{x}$ and ABE ciphertexts $\{c_1, \ldots, c_\ell\}$ along with the ABE decryption keys fsk_f^i to the server. The server first performs the homomorphic evaluation of f on $\hat{x}$ to obtain $\widehat{f(x)}$. It then uses fsk_f^i to decrypt the ABE ciphertexts $\{c_1, \ldots, c_\ell\}$ to obtain the GC labels. In an honest execution of this protocol, the ABE decryption will always output the GC labels that correspond to the bits of $\widehat{f(x)}$. The server then uses these labels to evaluate the garbled circuit, which outputs the value of $f(x)$ in the clear.

In a malicious setting, the server may try to utilize the protocol to decrypt some ciphertext $\widehat{g(x)}$ that is obtained by homomorphically evaluating another function g on $\hat{x}$. However, the protocol will reject such ciphertexts and abort. The reason is that the client has only issued the ABE decryption key fsk_f^i that corresponds to a function f for which decryption is allowed in the context of the application. Now, when the ABE ciphertexts $\{c_1, \ldots, c_\ell\}$ are decrypted using fsk_f^i, the labels obtained will not correspond to the bits of $\widehat{g(x)}$. Executing the garbled circuit on these labels will output an incorrect value, i.e., a value different from $g(x)$. This ensures that the server can only obtain the decrypted result when evaluation is done for the pre-defined function f for which the ABE keys were obtained and not for any other function $g \neq f$.

[7] A detailed description of this protocol is provided in Appendix B.

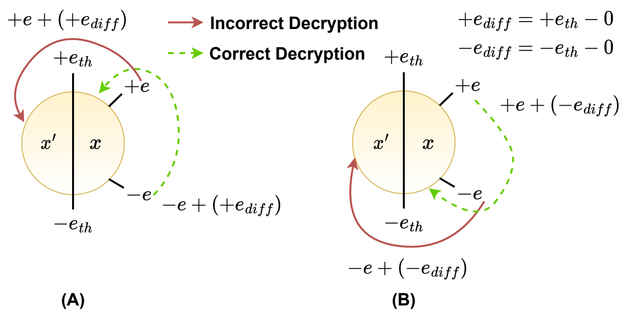

Fig. 1: Different error bounds plotted on a number line showing the effects of perturbation when (A) the positive range and (B) the negative range is chosen.

4 IND-CPA$^{\mathbf{C}}$ Attack on FHE Schemes

We now show that the practically existing (R)LWE-based exact FHE schemes [6, 7, 13, 15, 16] are vulnerable to a full key recovery attack under the IND-CPA$^{\mathsf{C}}$ notion. In other words, an adversary, which is the server itself in our case, can recover the entire secret key of the client even if it has access to a conditional decryption oracle. To highlight this vulnerability, we present a generic attack that works across all (R)LWE-based exact FHE schemes existing in practice. Our attack works by first recovering the underlying error in the FHE ciphertexts, which are then removed to obtain the noiseless versions of these ciphertexts. Given the security of (R)LWE comes from the presence of these errors, recovering them is akin to recovering the underlying secret key. We now present our attack, where we first show our attack intuition and then provide our attack description.

4.1 Attack Intuition

The ciphertexts in (R)LWE-based FHE schemes are of the form $c = (\mathbf{a}, b)$, where $\mathbf{a} \in \mathbb{Z}_q^k$ is a vector of dimension k whose coefficients are sampled uniformly from $\mathbb{Z}_q$ and $b = \mathbf{a} \cdot \mathbf{s} + x + e$. Here $\mathbf{s}$ is the secret key drawn uniformly from either $\{0, \pm 1\}^k$ or $\{0, 1\}^k$, x is the plaintext message, and e is an error value sampled from some Gaussian error distribution. With each homomorphic computation over these ciphertexts, the underlying error e grows. If this e remains below a pre-defined error threshold $\pm e_{th}$, decryption results in the correct message x. On the other hand, once this error grows beyond $\pm e_{th}$, decryption results in an incorrect message $x' \neq x$, causing a decryption failure. In practice, this growth of error is controlled by either bringing it down through bootstrapping [18] or by choosing appropriate parameters during the FHE setup phase to ensure that the error does not cross the threshold before the entire circuit is evaluated. However, the server being malicious can introduce crafted error values into the ciphertexts to induce a decryption failure. Moreover, our threat model provides the server with a local decryption oracle that it can use to decrypt a ciphertext $c = \widehat{f(x)}$

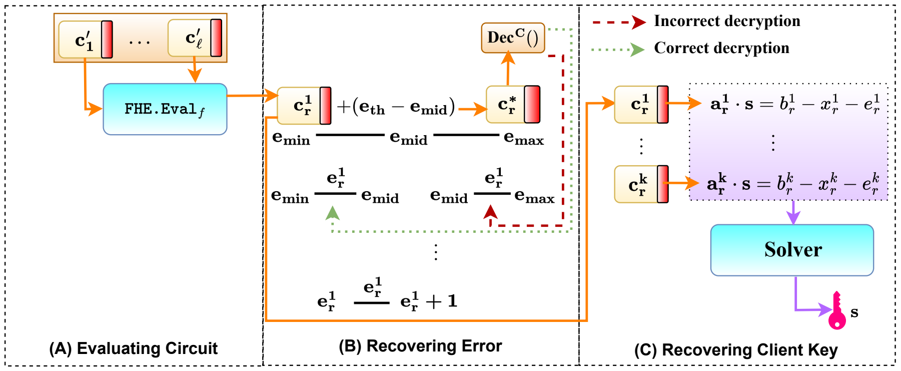
$\mathbf{c}'_1$ … $\mathbf{c}'_\ell$
FHE.Eval$_f$
(A) Evaluating Circuit
$\mathbf{c}^1_\mathbf{r}$ $+(\mathbf{e}_{\mathbf{th}} - \mathbf{e}_{\mathbf{mid}})$ $\mathbf{c}^*_\mathbf{r}$
$\mathbf{Dec}^\mathbf{C}()$
$\mathbf{e}_{\mathbf{min}}$ $\mathbf{e}_{\mathbf{mid}}$ $\mathbf{e}_{\mathbf{max}}$
$\mathbf{e}_{\mathbf{min}}$ $\mathbf{e}^1_\mathbf{r}$ $\mathbf{e}_{\mathbf{mid}}$ $\mathbf{e}_{\mathbf{mid}}$ $\mathbf{e}^1_\mathbf{r}$ $\mathbf{e}_{\mathbf{max}}$
$\mathbf{e}^1_\mathbf{r}$ $\mathbf{e}^1_\mathbf{r}$ $\mathbf{e}^1_\mathbf{r} + \mathbf{1}$
(B) Recovering Error
Incorrect decryption
Correct decryption
$\mathbf{c}^1_\mathbf{r}$ $\mathbf{a}^1_\mathbf{r} \cdot \mathbf{s} = b^1_r - x^1_r - e^1_r$
$\mathbf{c}^k_\mathbf{r}$ $\mathbf{a}^k_\mathbf{r} \cdot \mathbf{s} = b^k_r - x^k_r - e^k_r$
Solver
s
(C) Recovering Client Key

Fig. 2: End-to-End attack showing how (A) the function f is evaluated, (B) the underlying error is recovered using a binary search-based approach, and (C) recovered errors along with original ciphertexts are used to form a system of exact equations which is then solved to recover the client key.

that is the result of the homomorphic evaluation of f on input $\hat{x}$. The server can now introduce these errors in c to transform it into $c' = \widehat{f(x)}'$ and then decrypt it locally to check whether it still decrypts to $f(x)$ or not. One must note that the server can still only decrypt a ciphertext that is either obtained by homomorphically evaluating f on some input or by perturbing this ciphertext and nothing else. We now present an attack strategy that can be used to infer the bound of errors between which e lies and then use it to recover e itself. Once the errors are recovered, the attacker can recover the underlying secret key $\mathbf{s}$.

4.2 Modifying The Final Result

Fig. 1 shows the plot of the error e and its bound on a number line. From the figure, we can observe that the value of e is bounded by the range $[0, +e_{th}]$ or $[-e_{th}, 0]$ depending on whether e is positive or negative, resp. Here $+e_{th}$ and $-e_{th}$ are the positive and negative thresholds, resp. Now, for the decryption to be correct, i.e., decrypting c generates the message $f(x)$ and not any other message $x' \neq f(x)$, e needs to be either less than $+e_{th}$ or greater than $-e_{th}$. However, since the server neither knows the value nor the sign of e, it does not know whether it is less than $+e_{th}$ or greater than $-e_{th}$, and in turn does not know whether it is bounded by the range $[0, +e_{th}]$ or $[-e_{th}, 0]$. Thus, to infer the sign of e, the server has to guess about the same. Suppose it guesses e to be positive. In this case, it computes a quantity $e_{diff} = +e_{th} - 0 = +e_{th}$ and adds it to c. It then decrypts c and checks whether the decrypted value is equal to $f(x)$ or not. As depicted in Fig. 1(A), if the server guessed incorrectly and the sign of e is negative, the final error after perturbation lies within the permissible range (less than the threshold $+e_{th}$), albeit in the opposite sign domain. In contrast, if the server guessed correctly and the sign of e is positive, the final error after

Algorithm 1 Error Recovery using Binary Search

```
1: $e_{th}$ := error threshold
2: $e_{min} := 0$
3: $e_{max} := e_{th}$
4: $c$ := resultant ciphertext of some homomorphic computation $f$
5: $f(x)$ := original message in the ciphertext $c$
6: $e$ := original error in the ciphertext $c$
7: $start \leftarrow e_{min}$
8: $end \leftarrow e_{max}$
9: $e_{temp} \leftarrow 0$
10: function GETERROR($c$, $start$, $end$)
11:     if $start == end - 1$ then return $e_{temp}$
12:     else
13:         $mid \leftarrow \lfloor \frac{start+end}{2} \rfloor$
14:         $e_{diff} \leftarrow e_{th} - mid$
15:         $c \leftarrow c + e_{diff} = \mathbf{a} \cdot \mathbf{s} + f(x) + e + e_{diff}$
16:         $x' \leftarrow \mathtt{Dec}^{\mathtt{C}}(c)$
17:         $c \leftarrow c - e_{diff} = \mathbf{a} \cdot \mathbf{s} + f(x) + e + e_{diff} - e_{diff}$
            $= \mathbf{a} \cdot \mathbf{s} + f(x) + e$
18:         if $x' == f(x)$ then
19:             $e_{temp} \leftarrow mid$
20:             GETERROR($c$, $start$, $mid$)
21:         else
22:             GETERROR($c$, $mid$, $end$)
23:         end if
24:     end if
25: end function
```

the addition of perturbation lies beyond the permissible range (more than the threshold $+e_{th}$) in the positive domain.

The server may also start by guessing that the error e is $-ve$. In that case, it computes the quantity $e_{diff} = -e_{th} - 0 = -e_{th}$ and adds it to c. As depicted in Fig. 1(B), if the server guessed incorrectly and the sign of e is positive, the final error after perturbation lies within the permissible range (less than the threshold $-e_{th}$), albeit in the opposite sign domain. In contrast, if the server guessed correctly and the sign of e is negative, the final error after the addition of perturbation lies beyond the permissible range (more than the threshold $-e_{th}$) in the negative domain. Therefore, it is easy to note that a decryption failure would only occur when the server correctly guesses the sign of the error.

4.3 Error and Key Recovery Attack

Fig. 2 shows the overall process of our attack, which requires recovering errors e from at least k ciphertexts, where k is the length of the secret key $\mathbf{s}$. As already stated in Sect. 2.2, once these errors are recovered, the system of approximate equations (consisting of at least k equations) can be turned into a system of exact equations which can then be solved to recover $\mathbf{s}$. However, our attack additionally requires recovering the underlying plaintext message $f(x)$ from each ciphertext c since it is of the form $b = \mathbf{a} \cdot \mathbf{s} + f(x) + e$, and removing both $f(x)$ and e turns it into $b = \mathbf{a} \cdot \mathbf{s}$. The server can recover the message $f(x)$ by first homomorphically evaluating f on ψ (the set of encrypted client inputs) using the public key pk to obtain $c = \widehat{f(x)}$ and then decrypt it locally. To recover the error, we propose a binary search-based strategy, as shown in Alg. 1. The server first sets the values of two variables e_{min} and e_{max} to either 0 and $+e_{th}$ if e was determined positive, or 0 and $-e_{th}$ if e was determined negative, to cover the entire range of possible errors. Once done, the server proceeds to use active perturbations in the ciphertext c and decrypt it locally to observe the effect of the perturbation on the final result $f(x)$.

The central idea of the attack is that given two bounds e_{min} and e_{max}, we first determine whether e is closer to e_{min} or e_{max}, which can be found using the same strategy that was used to determine the sign of e. To elaborate further, we first set two variables *start* and *end* to 0 and e_{th} ($+e_{th}$ or $-e_{th}$ based on sign of e). The first condition we check is if *start* becomes equal to $end - 1$, which acts as the base case of our recursive algorithm and implies that we are left with only one error value in the range, which is e itself. Otherwise, as part of recursion, we compute a term *mid* as the midpoint of the range $[start, end]$. Following the notion of binary search, our objective is to divide the range into two halves and determine whether e lies in the first or second half. We do this by computing an error term $e_{diff} = e_{th} - mid$ which is then added to the locally computed ciphertext c. The idea is that if e lies closer to *end* (right of *mid* if taking $e_{th} = +e_{th}$ and left of *mid* if taking $e_{th} = -e_{th}$), then adding e_{diff} makes the overall error $(e + e_{diff})$ in c to cross the threshold e_{th}. Decrypting c at this point results in a decryption failure, and the server understands that the actual error e lies in between *mid* and *end*. On the other hand, if e lies closer to *start* (left of *mid* if taking $e_{th} = +e_{th}$ and right of *mid* if taking $e_{th} = -e_{th}$), then the addition of e_{diff} will not cause the overall error $(e + e_{diff})$ in c to cross the threshold e_{th}. Quite obviously, decrypting c at this point will not result in a decryption failure, and the server will understand that e lies in between *start* and *mid*. Therefore, similar to the working process of binary search, the server can eliminate half of the error space on every iteration and gradually progress toward the actual error. Once finished, the output of the algorithm is the actual error e of the ciphertext c.

The server can also use the aforementioned process to determine the value of e_{th} itself. It can first generate noiseless encryptions (by setting the error values to zero) of two messages of its choice using a secret key generated by the server itself. It can then perform a homomorphic gate operation, whose e_{th} it wants to determine, over these ciphertexts (whose error value remains zero). It can then run Alg. 1 over the resultant ciphertext by setting e_{min} and e_{max} to 0 and a large value. Once the error is recovered for each ciphertext, the server can trivially retrieve the secret key $\mathbf{s}$ using linear algebra methods such as Gaussian elimination [22] or lattice solvers such as LLL [26] or BKZ [37].

Why this is not a CCA1 attack? We would like to clarify that while the server can decrypt the ciphertext c (including its perturbed version) on its end, it needs to be remembered that c is the result of some homomorphic computation f and not arbitrary ciphertext (which includes the bootstrapping and relinearization keys). Given that the server is only allowed to decrypt the result of an evaluation of f, which it does, our attack falls under the IND-CPA$^{\mathtt{C}}$ notion and not the IND-CCA1 notion.

4.4 Comparison across FHE schemes

We note that the number of ciphertexts required to launch the attack is in the order of the size of the key, or more precisely, $\Omega(k)$. Concretely, our attack requires recovering errors from at least k ciphertexts. Therefore, our attack requires to make at least $k + k \cdot log(e_{th})$ queries to the $\mathtt{Dec}^{\mathtt{C}}\mathtt{()}$ oracle, where k queries are required to recover the underlying messages while $k \cdot log(e_{th})$ queries are required to recover underlying error values. At this point, we would like to highlight that the value of both k and e_{th} varies across the FHE schemes due to the difference in choice of parameters. This further implies that the number of queries made to the $\mathtt{Dec}^{\mathtt{C}}\mathtt{()}$ oracle will also vary across these schemes. Table 1 provides concrete values of the number of queries required to recover the errors from sufficient ciphertexts for three practical FHE schemes, namely TFHE, FHEW, and B/FV.[8] From the table, we can observe that even for the same attack vector, the total number of queries made to the conditional decryption oracle differs across the three FHE schemes. The reason is that both the length of the key and the total number of queries required to recover a single error value vary across these schemes. We can further observe that FHEW requires the minimum number of queries to be made for a successful key recovery, and TFHE requires the highest number of queries. However, for B/FV, the result has been shown for the minimum parameter set, and the number of queries will surpass that of TFHE for higher parameter sets. This shows that the query complexity (the total number of decryption queries made) and, in turn, the IND-CPA$^{\mathtt{C}}$ notion can be used in practice to compare the FHE schemes in terms of their resistance against full key recovery attacks.

To show the feasibility of our attack, we provide a proof-of-concept (PoC) code that demonstrates our attack on a well-known LWE-based FHE scheme.[9] The PoC is built on TFHE scheme with a key size of 630 bits. We were successfully able to recover the underlying errors from 1112 ciphertexts, requiring 35792 decryptions on the server side, which were then used to successfully recover the entire secret key. The overall attack took around 15 minutes to carry out on a 2nd-gen Intel® Xeon® workstation with 128 GB RAM.

[8] The results for BGV will be similar to those of B/FV because of the similarity of these two schemes. We do not provide results for approximate FHE schemes such as CKKS [12] since we only focus on exact FHE schemes in this paper.

[9] PoC can be found on `https://github.com/cracking-tfhe/Anonymous_Repository`.

Table 1: Shows the key length, total no. of queries required to recover errors from one ciphertext, and the total no. of queries made to recover errors from sufficient ciphertexts for three practical exact FHE schemes.

	k	$log(e_{th})$	No. of decryption queries
TFHE	630	30	19950
FHEW	500	6	3980
B/FV	1024	16	16850

4.5 Introducing Errors At Lower Levels

We now show why the attack does not work if the perturbations are induced in ciphertexts generated at lower levels of the circuit that is evaluating f. We present, without loss of generality, the case where f represents a single (homomorphic) addition between two FHE ciphertexts $c_1 = (\mathbf{a_1}, b_1 = \mathbf{a_1} \cdot \mathbf{s} + x_1 + e_1)$ and $c_2 = (\mathbf{a_2}, b_2 = \mathbf{a_2} \cdot \mathbf{s} + x_2 + e_2)$. This operation produces a ciphertext $c_3 = (\mathbf{a_3}, b_3 = \mathbf{a_3} \cdot \mathbf{s} + x_3 + e_3)$, where $\mathbf{a_3} = \mathbf{a_1} + \mathbf{a_2}$, $m_3 = m_1 + m_2$ and $e_3 = e_1 + e_2$. We assume that the sign of e_1 is positive, which the server does not know and is trying to infer. Based on our method from Sect. 4.2, the server assumes e_1 to be positive, computes the quantity $e_{diff} = +e_{th} - 0$ and adds it to c_3. Finally, it decrypts c_3 locally to observe the effect of the added error. Now, based on the sign and value of e_2 and the value of e_1, we have the following possibilities of the final error $e_3' = e_1 + e_{th} + e_2$ in the ciphertext c_3:

❶ **$\mathbf{e_2}$ is positive:** In this case, the value of e_3' crosses the error threshold $+e_{th}$ irrespective of whether $e_1 > e_2$ or $e_1 = e_2$ or $e_1 < e_2$. This induces a decryption failure, and the server infers the sign of e_1 to be positive. The probability of this combination occurring in practice is $P_1 = \Pr[e_1 \text{ is positive}] \times \Pr[e_2 \text{ is positive}] = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$.

❷ **$\mathbf{e_2}$ is negative but $\mathbf{e_2 < e_1}$:** In this case, the value of e_3' crosses the error threshold $+e_{th}$ as $e_2 < e_1$ and so $e_1 - e_2 > 0$. This induces a decryption failure, and the server infers the sign of e_1 to be positive. The probability of this combination occurring in practice is $P_2 = \Pr[e_1 \text{ is positive}] \times \Pr[e_2 \text{ is positive}] \times \Pr[e_1 > e_2] = \frac{1}{2} \times \frac{1}{2} \times \frac{1}{3} = \frac{1}{12}$.

❸ **$\mathbf{e_2}$ is negative but $\mathbf{e_2 = e_1}$:** In this case, the value of e_3' becomes equal to $+e_{th}$ as $e_2 = e_1$ and so $e_1 + e_2 = 0$. This does not induce a decryption failure, and the server mistakenly infers the sign of e_1 to be negative. The server may check this by setting $e_{diff} = -e_{th}$. However, in this case, e_3' becomes equal to $-e_{th}$, and even this does not induce a decryption failure. The probability of this combination occurring in practice is $P_3 = \Pr[e_1 \text{ is positive}] \times \Pr[e_2 \text{ is positive}] \times \Pr[e_1 = e_2] = \frac{1}{2} \times \frac{1}{2} \times \frac{1}{3} = \frac{1}{12}$.

❹ **$\mathbf{e_2}$ is negative but $\mathbf{e_2 > e_1}$:** In this case, the value of e_3' becomes less than $+e_{th}$ as $e_2 > e_1$ and so $e_1 - e_2 < 0$. This does not induce a decryption failure, and the server mistakenly infers the sign of e_1 to be negative. The server may check this by setting $e_{diff} = -e_{th}$. However, in this case, e_3' crosses the error threshold $-e_{th}$, which induces a decryption failure. However, this decryption failure was due to e_2 being negative, but the server assumes that this was caused due to e_1 being negative. Thus the server again mistakenly infers the sign of e_1 to be negative. The probability of this combination occurring in practice is P_4 $=$ Pr[e_1 is positive] $\times$ Pr[e_2 is positive] $\times$ Pr[$e_1 < e_2$] $= \frac{1}{2} \times \frac{1}{2} \times \frac{1}{3} = \frac{1}{12}$.

We get similar cases when e_1 is negative. We can observe that while the server infers the correct sign of e_1 when cases ❶ and ❷ occur, it infers the wrong sign of the error e_1 when cases ❸ and ❹ occurs. Thus the probability of the server to infer the correct sign of e_1 is $P_{\pm} = 2\times(P_1+P_2) = 2\times(\frac{1}{4}+\frac{1}{12}) = 2\times(\frac{3+1}{12}) = \frac{2}{3}$.[10] Now, this probability is for a single ciphertext, and the server needs to extract the sign of the errors from at least k ciphertexts, which it can do with probability $P_{\pm}^k = (\frac{2}{3})^k$. Given that the dimension of the secret key k is set to 1024 and above during practical instantiations of the FHE schemes, the value of $P_{\pm}^k$ turns out to be $\approx 8 \times 10^{-179}$. This implies that to recover the sign of the errors in at least $k = 1024$ ciphertexts, the server needs access to $\approx 10^{178}$ ciphertexts, which is infeasible to obtain in practice. It is thus sufficient to only protect the final computation in order to prevent our attack. However, in practice, there might exist other attack vectors that might not be stopped by simply protecting the final level of the computation. Therefore, instead of providing ad-hoc techniques to prevent against such attacks, we propose a generic technique that works across all FHE schemes and is provably secure. Before going into the details of our technique, we explain how our attack is applicable to the construction of [19].

4.6 Succinct FE in malicious setting

The aforementioned IND-CPA$^{\mathsf{C}}$ attack shows that practically existing (R)LWE-based FHE schemes are insecure in the presence of a malicious adversary. The implication of this observation is that while the succinct FE construction presented in [19], which is a prime use-case implementation of conditional decryption based FHE application, is secure in a semi-honest setting, the security of the overall FE falters under the IND-CPA$^{\mathsf{C}}$ notion. The leakage of the secret FHE key essentially jeopardizes the underlying privacy guarantee and allows the adversary (server in this case) to decrypt all ciphertext. The overall security guarantee of [19] is based on the security guarantees of constituent cryptoprimitives, such as FHE, ABE and GC. The specific design in [19] essentially allows the server to drive inputs to the FHE decryption oracle. Concretely, the bits of $\widehat{f(x)}$ (computed on the server side) unlock the specific input labels of GC (within which the FHE decryption circuit resides) by driving the specific predicates of ABE decryption. We note that *assuming* the predicate evaluation is honest, the ABE unlocks *exactly* one of its two inputs. In [19], this input then drives exactly one wire of the underlying GC. However, when perturbations are introduced in $\widehat{f(x)}$, it leads to incorrect ABE decryption because of erroneous predicate evaluation. This in turn unlocks the *incorrect* GC label, causing the garbled FHE

[10] The multiplier 2 comes from the fact that we get cases ❶ and ❷ once for when e_1 is positive and when e_1 is negative.

decryption circuit to output either $f(x)$, or $f(x)' \neq f(x)$. A malicious server can use this output as a distinguisher to recover errors in the underlying (R)LWE samples central to FHE, thereby leading to the recovery of the full FHE decryption key. Thus, while the security guarantees of FHE, ABE, and GC remain intact in isolation, the combination of these primitives to implement FHE decryption causes the breakdown of security (leaking the FHE decryption key) in the malicious server model.[11]

5 Constructing IND-CPA$^{\mathsf{C}}$-secure FHE

The attack described in the previous section highlights that none of the existing exact FHE schemes are IND-CPA$^{\mathsf{C}}$ secure. In other words, an active adversary can carry out key recovery attacks on FHE schemes if given access to a conditional decryption oracle. Such attack vectors arise from the lack of integrity checks in FHE ciphertexts, which allows a malicious adversary to tamper with the ciphertexts without getting detected.

To alleviate this problem, a recent direction of research is focusing on incorporating Zero Knowledge Proofs (ZKPs) techniques on top of FHE computations. These solutions work by using SNARKs (cf. Sect. 2.7 for more details) to generate proof of honest homomorphic computations by the server. Once generated, it sends all the ciphertexts (which include the ones corresponding to the intermediate results) associated with the computation along with the proof back to the decrypting party, which is the client in the general setting. The client uses these ciphertexts and proof to first verify the integrity of the homomorphic computation and proceeds with decryption only if the verification holds; otherwise, it aborts the protocol.

The aforementioned technique, termed verifiable FHE (vFHE), was first introduced in [40] and served as a good starting point towards ensuring FHE integrity through ZKPs. However, its implementation was severely limited to a small class of HE schemes that do not involve ciphertext-maintenance operations such as bootstrapping and relinearization. The reason is that the mathematics of such maintenance operations was found to be incompatible with the chosen ZKP solutions. However, recent improvements in ZKP techniques have led to the support for generating proofs of honest evaluation of the ciphertext-maintenance operations. In this direction, the authors of [3] introduced a method of proving the correct evaluation of relinearization and mod-switching operations. Similarly, the authors of [39] showed how to prove the correct evaluation of TFHE [13] bootstrapping operation. Finally, [31] showed how IND-CPA secure FHE schemes can be converted to IND-CCA1 secure FHE schemes with the use of ZKP solutions.

The idea of vFHE was originally proposed to mitigate reaction-based attacks on FHE schemes [8, 14, 42] by aiding the client to differentiate between whether an incorrect decryption was accidental or was intentionally induced by a malicious server. The proposed technique required the server to generate individual proofs for each homomorphic computation it performs, including

[11] A detailed discussion on this can be found in Appendix C.

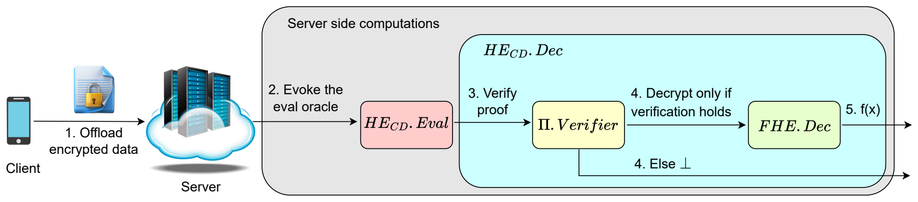

Fig. 3: Schematic of our IND-CPA$^{\mathtt{C}}$ secure FHE scheme $\mathtt{HE}_{\mathtt{CD}}$. The server additionally generates proof of computation, which is then verified. Decryption only proceeds if verification holds; otherwise, the construction outputs a $\perp$.

ciphertext-maintenance operations. Once done, the proofs along with all the FHE ciphertexts were sent back to the client to be verified. The client accepted the (post-decrypted) result of the requested computation only if all the accompanying proofs hold.

Our technique of converting IND-CPA secure FHE schemes to IND-CPA$^{\mathtt{C}}$ secure FHE schemes works along similar lines, with the difference that the generated proofs need not be sent back to the client. In other words, both the proof generation and verification run on the server side, albeit with the verification step running inside a garbled circuit (GC), along with the FHE decryption operation. The idea is to allow the conditional decryption oracle to first verify the proofs on its end to ensure the integrity of the FHE ciphertext and only decrypt it if verification holds. Given the verification and decryption run on the server itself (inside a GC), our solution does not incur significant network overhead as the proofs and ciphertexts are not required to be sent to the client. We now show our construction that can be used to convert an IND-CPA secure exact FHE scheme into an IND-CPA$^{\mathtt{C}}$ secure exact FHE scheme. Fig. 3 provides a summary of our construction.

Definition 2. *Let* `FHE = (FHE.KeyGen, FHE.Enc, FHE.Eval, FHE.Dec)` *be an exact public-key fully homomorphic encryption scheme as defined in Sect. 2.3 over a message space $\mathcal{M}$, a ciphertext space $\mathcal{C}$, and a class of functions $\mathcal{F}$. Moreover, let Π* `= (SNARK.Setup, SNARK.Prover, SNARK.Verifier)` *be a SNARK defined over the language*

$$\mathcal{L} = \{(\{c_i\}, c_r) | \exists f \in \mathcal{F}, c_r = \mathtt{FHE.Eval}(pk, f, \{c_i\})\}$$

Our IND-CPA$^{\mathtt{C}}$ secure FHE scheme $\mathtt{HE}_{\mathtt{CD}}$ is a tuple of algorithms ($\mathtt{HE}_{\mathtt{CD}}\mathtt{.KeyGen}$, $\mathtt{HE}_{\mathtt{CD}}\mathtt{.Enc}$, $\mathtt{HE}_{\mathtt{CD}}\mathtt{.Eval}$, $\mathtt{HE}_{\mathtt{CD}}\mathtt{.Dec}$), where the algorithms are defined as follows:

1. *$\mathtt{HE}_{\mathtt{CD}}\mathtt{.KeyGen}(1^\lambda, CS)$: Upon input the security parameter λ and a constraint system CS corresponding to the language $\mathcal{L}$, it executes $\mathtt{FHE.KeyGen}(1^\lambda)$ to obtain the key pair ($\mathtt{pk}$, $\mathtt{sk}$). It further executes $\Pi\mathtt{.Setup}(1^\lambda, CS)$ to obtain a pair ($\mathtt{crs}$, $\mathtt{st}$). It outputs the pair ($\mathtt{pk}$, $\mathtt{crs}$) while keeping the pair ($\mathtt{sk}$, $\mathtt{st}$) as secret.*
2. *$\mathtt{HE}_{\mathtt{CD}}\mathtt{.Enc}(\mathtt{pk}, x)$: Upon input the public key $\mathtt{pk}$ and a message $x \in \mathcal{M}$, it executes $\mathtt{FHE.Enc}(\mathtt{pk}, x)$ to obtain a ciphertext c. It outputs c.*

3. *$\mathtt{HE_{CD}.Eval}$(pk, f, $\{c_i\}$, crs): Upon input the public key* pk*, a function $f \in \mathcal{F}$, a set of FHE ciphertexts $\{c_i\}$, and a common reference string* crs*, it executes* FHE.Eval(pk, f, $\{c_i\}$) *to obtain the resultant ciphertext c_r. It further executes* Π.Prover(crs, ($\{c_i\}$, c_r), w) *to obtain the proof of execution π. Here the pair ($\{c_i\}$, c_r) serves as the statement, and the intermediate results obtained during the FHE evaluation serve as the corresponding witness w to the underlying SNARK protocol. It outputs the pair (c_r, π).*
4. *$\mathtt{HE_{CD}.Dec}$(sk, st, π, $\{c_i\}$, c_r): Upon input the secret key* sk*, the verification state* st*, the proof π, and the pair of input-output ciphertexts ($\{c_i\}$, c_r), it first executes* SNARK.Verifier(st, ($\{c_i\}$, c_r), π) *and obtains a verification bit bt. If bt $= 1$, it executes* FHE.Dec(sk, c_r) *to obtain the decrypted result $f(\{x_i\})$ which it then outputs. On the other hand, if bt $= 0$, it outputs a $\perp$.*

In an instantiation of this protocol, the client can embed both the decryption and verification steps, along with their corresponding keys, in a garbled circuit. This prevents the server from tampering with any of these two steps. We now prove the security of our proposed construction.

Theorem 1. *If* FHE *is a IND-CPA secure fully homomorphic encryption scheme and Π is a SNARK with knowledge soundness property, then our construction* $\mathtt{HE_{CD}}$ *is an IND-CPA$^{\mathsf{C}}$ secure FHE scheme.*

Proof. The security of our construction reduces to the properties of the underlying FHE and SNARK schemes. Given an FHE ciphertext $c_r = f(\{c_i\})$ and the associated proof π corresponding to the statement ($\{c_i\}$, c_r), the adversary can do one of the following things: ❶ it does not tamper with either c_r or π, ❷ it tampers with c_r to turn it into c_r^* but leaves the π intact, ❸ it leaves c_r intact but tampers with π to turn it into π^*, and ❹ it tampers with both c_r and π to turn them into c_r^* and π^*.
Correctness: For case ❶, Π.Verifier(st, ($\{c_i\}$, c_r), π) $= 1$ from the *completeness* property of the underlying SNARK scheme, and FHE.Dec(sk, c_r) $= f(\{x_i\})$ from the *correctness* property of the underlying FHE scheme. Our construction therefore correctly outputs the value of $f(\{x_i\})$.
Security: For case ❷, Π.Verifier(st, ($\{c_i\}$, c_r^*), π) $= 0$ due to the *knowledge soundness* property of the underlying SNARK scheme. Given decryption does not proceed when the proof does not hold, our construction outputs a $\perp$. Cases ❸ and ❹ follow similar to ❷.

Our construction, following from the *knowledge soundness* property of the underlying SNARK, ensures that the proof verification fails upon any perturbation introduced at any point of the homomorphic computation.[12] This prevents the server from obtaining decryptions of invalid ciphertexts (which were not obtained from an honest evaluation of f over $\{c_i\}$). Finally, the verification and

[12] While our attack works only when the perturbation is introduced in the final resultant ciphertext, future attacks under the IND-CPA$^{\mathsf{C}}$ notion may succeed by tampering with the input or intermediate ciphertexts.

decryption operations are made to execute within a garbled circuit. This ensures that the server is not able to tamper with the output bit of the verification step since the garbling step hides its location. In other words, the server cannot run the verification step honestly (on correct ciphertexts and proof) and then flip the result of the verification bit before proceeding with the decryption on incorrect ciphertexts. Doing so would require the server to know the location of the verification bit inside the garbled circuit, which will be akin to breaking the *security* guarantee of the garbled circuit.

6 Discussion and Future Work

Approximate FHE schemes. In this paper, we primarily focus on exact FHE schemes in which decryption outputs an exact result of the homomorphic computation while the underlying error is removed. However, approximate FHE schemes such as [12] also exist in literature where the decryption outputs an approximation of the result of a homomorphic computation. The approximation comes from the fact that the decryption outputs a message $x + e$ where e is the underlying error value and x is the actual message. Authors in [27] showed that such schemes can be broken under the IND-CPA$^{\mathsf{D}}$ notion since the decryption oracle trivially leaks the error value as part of the message itself. To counter these attacks, authors in [28] proposed a post-processing of the decrypted result $x + e$ to introduce an extra error value that smudges the original error value e. Our attack is not directly applicable to such schemes since it relies on the knowledge of the exact value of e, which is smudged during the post-processing step of [28]. We leave the evaluation of the security of approximate FHE schemes under our IND-CPA$^{\mathsf{C}}$ as future work.

Compact proofs. As highlighted in Sect. 5, there exist practical instantiations of SNARKs that support verifiability of FHE computations, even with the presence of ciphertext-maintenance operations. However, the primary issue with these instantiations is that the size of the proof depends on the size of the circuit being evaluated. In other words, the size of the proof grows with the size of the circuit, which also affects the size of the verifier circuit since it now has to verify a large amount of proofs. On the other hand, our solution requires the proofs to be compact (possibly independent of the size of the circuit to be proven) and, in turn, the verification step to be as small and simple as possible, given the same needs to be implemented inside a GC.[13] We leave the constructions of such SNARKS as future work.

[13] A larger verification circuit will not only increase the size of the GC but will require a significant amount of time to run. The exact runtime will be determined by the exact instantiation of the SNARK and GC scheme.

7 Conclusion

Fully homomorphic encryption allows arbitrary computations directly over encrypted data while ensuring the result of such computations remains encrypted. While traditionally the client is tasked with decrypting the FHE ciphertexts, a wide range of applications benefit from the server having the ability to decrypt the result of the homomorphic computation. As shown in [19], such *conditional decryption* is possible in practice where the server is allowed to decrypt the result of a fixed computation and nothing else. However, this added functionality can be exploited by a malicious server to carry out attacks against the underlying FHE scheme. Existing security notions fail to capture this scenario since they assume that only the client can decrypt FHE ciphertexts. Therefore, in this paper, we introduce a new notion denoted IND-CPA$^{\mathsf{C}}$ that captures this setting in a better way. We then show that none of the existing FHE schemes are secure under our notion by demonstrating a full-key recovery attack that exploits the ability of the server to perform conditional decryptions locally. Finally, we propose a technique to convert any IND-CPA secure exact FHE scheme into an IND-CPA$^{\mathsf{C}}$ secure exact FHE scheme. Our construction makes use of SNARKs to generate proof of honest homomorphic computation. During decryption, the proofs are verified, and decryption proceeds only when all the verification holds. Both the verification and the decryption steps run inside a garbled circuit and thus are tamper-proof. We believe that our analysis will aid the application developers in building robust protocols utilizing the added benefit of local decryption without compromising with the security of the underlying FHE scheme.

Acknowledgements. The authors would like to thank the anonymous reviewers for their valuable suggestions, which helped us to improve the paper. The authors would additionally like to thank the Centre for Hardware-Security Entrepreneurship Research & Development (C-HERD), MeitY, Govt. of India, and Information Security Education and Awareness (ISEA), MeitY, Govt. of India, for partially funding this research.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

A Working of Exact FHE schemes

In this section, we describe the overall working principle of FHE schemes, which can be divided into three stages, namely encryption, evaluation, and decryption. In the traditional setting, the evaluation stage runs on the server side while encryption and decryption are performed on the client side. We now briefly explain each of these stages, which differs between LWE and RLWE FHE schemes.

A.1 Encryption Stage

In the LWE schemes, a message x is encrypted under a secret key $\mathbf{s} \in \mathbb{B}$ as $c = (\mathbf{a}, b)$, where $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^k$ and $b = \mathbf{a} \cdot \mathbf{s} + x + e$. Here, $e \leftarrow \mathcal{N}_{0,\sigma}$ is a *small* error value. In the RLWE schemes, a message polynomial $\mathbf{x}$ is encrypted under a secret key $\mathbf{s} \in \mathbb{B}$ as $c = (\mathbf{b}, \mathbf{a})$, where $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$ and $\mathbf{b} = -(\mathbf{a} \cdot \mathbf{s} + \mathbf{e}) + \mathbf{x}$. Here, $\mathbf{e} \leftarrow \mathcal{N}_{0,\sigma}$ is a *small* error polynomial.

A.2 Homomorphic Computation Stage

In the LWE schemes, homomorphic addition between two ciphertexts $c_1 = (\mathbf{a_1}, b_1)$ and $c_2 = (\mathbf{a_2}, b_2)$ is defined as $c_r = c_1 + c_2 = (\mathbf{a_1} + \mathbf{a_2}, b_1 + b_2) = (\mathbf{a_r}, b_r)$. A multiplication between a scalar z and a ciphertext $c_1 = (\mathbf{a_1}, b_1)$ is defined as $c_r = (z \times \mathbf{a_1}, z \times b_1) = (\mathbf{a_r}, b_r)$, where $z \times \mathbf{a_1}$ implies that z is multiplied with each co-efficient of the vector $\mathbf{a_1}$. In practical LWE-based FHE schemes such as [13, 15], each homomorphic operation is followed by a bootstrapping step which resets the underlying error value in the resultant ciphertext.[14] This step works by homomorphically decrypting the ciphertext and uses a bootstrapping key that is an encryption of the decryption key $\mathbf{s}$ under a different (secret) key. In the RLWE schemes, homomorphic addition and scalar multiplication are defined in a similar way as defined in LWE schemes. Additionally, a homomorphic multiplication between the ciphertexts $c_1 = (\mathbf{b_1}, \mathbf{a_1})$ and $c_2 = (\mathbf{b_2}, \mathbf{a_1})$ is defined as $c'_r = (\mathbf{b_1} \cdot \mathbf{b_2}, \mathbf{b_1} \cdot \mathbf{a_2} + \mathbf{b_2} \cdot \mathbf{a_1}, \mathbf{a_1} \cdot \mathbf{a_2})$. Given this operation increases the elements in the ciphertext from 2 to 3 ring elements, it is followed by a relinearization step in the practical RLWE-based FHE schemes such as [6, 7, 16]. This step takes as input a relinearization key and a ciphertext c'_r and converts it back to a ciphertext $c_r = (\mathbf{b_r}, \mathbf{a_r})$ containing only 2 ring elements.

A.3 Decryption Stage

In the LWE schemes, a ciphertext $c = (\mathbf{a}, b)$ encrypted under a secret key $\mathbf{s}$ is decrypted via a two-step process. In the first step, the phase of the underlying message x is computed as $\phi = b - \mathbf{a} \cdot \mathbf{s}$ which gives at output a noisy version $x + e$ of the message. In the second step, this phase ϕ is rounded to remove the error e and recover x. In the RLWE schemes, decrypting a ciphertext $c = (\mathbf{b}, \mathbf{a})$ works in a similar fashion as in the LWE-based schemes, with the difference that the phase is computed as $\phi = \mathbf{b} + \mathbf{a} \cdot \mathbf{s}$.

One may note that for both LWE- and RLWE-based FHE schemes, the decryption stage outputs the correct message only if the underlying error e in the ciphertext being decrypted lies below a pre-defined threshold.

[14] This step is required since the correctness of the decrypted result does not hold once the overall error crosses a pre-defined threshold.

B Conditional Decryption Scheme

Authors in [19] proposed a succinct function encryption scheme FE[15] that can be used to decrypt the result of a homomorphic evaluation of f on some (encrypted) client input $\hat{x}$. In this section, we provide a brief overview of their construction in the form of a game $\prod_{\mathcal{C},\mathcal{S}}^{\mathtt{CD}}$[16] between a client $\mathcal{C}$ and a server $\mathcal{S}$. We assume that the length of the FHE ciphertexts (irrespective of whether they are generated during encryption or evaluation) is ℓ.

Conditional Decryption of FHE output $\prod_{\mathcal{C},\mathcal{S}}^{\mathtt{CD}}$:

Setup Phase (FE.Setup)

1. The client $\mathcal{C}$ runs the ABE.Setup algorithm ℓ times as $(fmpk_i,\ fmsk_i) \leftarrow$ $\mathtt{FE.Setup}(1^\lambda)$ for each bit of the FHE ciphertext to generate a tuple of master public keys as $\mathtt{MPK} = (fmpk_1, \cdots, fmpk_\ell)$ and master secret keys as $\mathtt{MSK} = (fmsk_1, \cdots, fmsk_\ell)$.

Keygen Phase (FE.KeyGen)

1. The client $\mathcal{C}$ takes the master secret key MSK and a function $f \in \mathcal{F}$ that accepts an n-bit input and then selects $\mathtt{FHE.Eval}_f^i$ from a broader set of functions $\mathcal{F}'$. Here $\mathtt{FHE.Eval}_f^i$ computes the ith-bit of $\widehat{f(x)}$, where x is the input of the client.
2. It then runs the ABE.KeyGen algorithm to generate secret keys for each function $\mathtt{FHE.Eval}_f^i$ as $fsk_i \leftarrow \mathtt{ABE.KeyGen(fmpk}_i\mathtt{,\ FHE.Eval}_f^i\mathtt{)}$. One may observe that each function $\mathtt{FHE.Eval}_f^i$ is equivalent to the ABE predicates p from Sect. 2.5.
3. It finally outputs a tuple $fsk_f = (fsk_1, \cdots, fsk_\ell)$ as a secret key for the function f. Here, each secret key fsk_i acts as a secret key corresponding to the predicate $\mathtt{FHE.Eval}_f^i$.

Encryption Phase (FE.Enc)

1. The client $\mathcal{C}$ chooses its n-bit input $x = x_1 \cdots x_n$, where x_i is ith-bit of x.
2. It generates a fresh key pair (hpk, hsk) by running $\mathtt{FHE.KeyGen(}1^\lambda\mathtt{)}$.
3. It then encrypts each bit of x as $\psi_i \leftarrow \mathtt{FHE.Enc(hpk,}x_i\mathtt{)}$ and obtains as output n FHE ciphertexts $\psi = (\psi_1, \cdots, \psi_n)$.
4. It then runs $\mathtt{Gb.Garble(}1^\lambda\mathtt{,\ FHE.Dec(hsk,}\cdot\mathtt{))}$, where $\mathtt{FHE.Dec(hsk,}\cdot\mathtt{)}$ is the FHE decryption circuit. It receives as output a garbled circuit Γ and a set of labels $\{L_i^0, L_i^1\}$ for each bit of the FHE ciphertext.
5. Finally, the client $\mathcal{C}$ produces the ABE ciphertexts as $c_i \leftarrow \mathtt{ABE.Enc(}fmpk_i\mathtt{,}$ $\mathtt{(hpk,\ }\psi\mathtt{),\ }L_i^0\mathtt{,\ }L_i^1\mathtt{)}$, and sends these ciphertexts $(c_1, \cdots, c_\ell)$ along with the garbled circuit Γ to the server $\mathcal{S}$. It also sends the pair (hpk, ψ) consisting of FHE public key and ciphertexts to the server $\mathcal{S}$. This is in accordance with [21] which is the underlying ABE scheme of this construction.

15 Not to be confused with the description of FE from Sect. 2.6.

16 We reuse the same naming convention from [19]; CD stands for conditional decryption.

Decryption Phase (`FE.Dec`)

1. The server $\mathcal{S}$ runs the ABE decryption algorithm on the ciphertexts $(c_1, \cdots, c_\ell)$ to obtain the labels of the garbled circuit as $L_i^{d_i} \leftarrow$ `ABE.Dec(`fsk_i`,` c_i`)`, where d_i is the ith bit of $\widehat{f(x)}$. One may observe that on an honest execution of this operation, the server only obtains the label $L_i^{d_i}$ and not the other label $L_i^{1-d_i}$.
2. Finally, the server $\mathcal{S}$ runs the garbled circuit evaluation algorithm `Gb.Eval(`Γ`,` $L_1^{d_1}, \cdots, L_\ell^{d_\ell}$`)`, which internally runs the FHE decryption circuit `FHE.Dec(hsk,` $d_1, \cdots, d_\ell$`)`.

C Conditional Decryption Scheme in Malicious Setting

In this section, we show that our attack in Sect. 4 does not break the security guarantees of the underlying building blocks, which include FHE, ABE and garbled circuits of [19].

FHE. The security guarantee of FHE comes from the fact that given a ciphertext $c = \mathbf{a} \cdot \mathbf{s} + m_{bt} + e$ to the adversary that is an encryption of one of the two messages m_0 and m_1 of its choice, it cannot determine the value of bt even if it is given access to an encryption oracle. This is guaranteed by the hardness of the (R)LWE problem upon which FHE constructions are based. Now, given that the adversary neither knows the value of $\mathbf{s}$ nor that of e, it cannot determine anyone of them, or for the matter even m_{bt}, by just observing the ciphertext c. At this point, if the adversary adds a safe error ϵ to c then its total error becomes $e + \epsilon$. However, the adversary can still not determine the value of $\mathbf{s}$, e or m_{bt}. This shows that adding a perturbation to an FHE ciphertext does not break its security guarantee of IND-CPA security.

GC. The security guarantee of a garbling scheme comes from the fact that given the garbling Γ of a circuit C and the labels $L_1^{x_1}, \cdots, L_\ell^{x_\ell}$ that encode an input $x = x_1 \cdots x_\ell$, the scheme neither leaks the circuit C nor the input x. However, it cannot ensure whether the label $L_i^{x_i}$ it received as input was the one it was supposed to receive in a correct execution of the protocol. Given the adversary is malicious, it can replace this label with $L_i^{1-x_i}$. The garbled circuit now evaluates C over x' that differs from x by the i^{th} bit. However, the scheme still does not reveal anything about neither C nor x (not even the bit x_i as it was merely changed, which the adversary already knows). This shows that changing an input label of the garbled circuit does not break its security guarantee of hiding the inputs and the circuit.

ABE. The security guarantee of the two-outcome ABE scheme comes from the fact that given a predicate p, an attribute x, and two messages M_0 and M_1, the scheme only outputs the message $M_{p(x)}$ and does not leak anything about the message $M_{(1-p(x))}$. However, the value of $p(x)$ is determined by the value of $\widehat{f(x)}$. Thus, a malicious server can alter the value of $p(x)$ to $1 - p(x)$ by perturbing the value of $\widehat{f(x)}$. The ABE decryption now outputs the message $M_{(1-p(x))}$, but now it hides the message $M_{p(x)}$. Thus, at all points of time, the adversary only receives one message as output of the ABE decryption (the one it chose to receive), while it does not get to know anything about the other message. This shows that changing the computed value of $p(x)$ in ABE does not break its security guarantee of only revealing one message and hiding the other one.

References

1. Akavia, A., Gentry, C., Halevi, S., Vald, M.: Achievable cca2 relaxation for homomorphic encryption. Journal of Cryptology **38**(1), 1–43 (2025). https://doi.org/10.1007/s00145-024-09526-1
2. Akavia, A., Vald, M.: On the privacy of protocols based on cpa-secure homomorphic encryption. Cryptology ePrint Archive, Paper 2021/803 (2021), https://eprint.iacr.org/2021/803
3. Atapoor, S., Baghery, K., Pereira, H.V., Spiessens, J.: Verifiable fhe via lattice-based snarks. IACR Communications in Cryptology **1**(1) (2024). https://doi.org/10.62056/a6ksdkp10
4. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: the ACM Conference on Computer and Communications Security, CCS'12 (2012). https://doi.org/10.1145/2382196.2382279
5. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Theory of Cryptography: 8th Theory of Cryptography Conference, TCC 2011. pp. 253–273 (2011). https://doi.org/10.1007/978-3-642-19571-6_16
6. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886 (2012). https://doi.org/10.1007/978-3-642-32009-5_50
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014). https://doi.org/10.1145/2633600
8. Chaturvedi, B., Chakraborty, A., Chatterjee, A., Mukhopadhyay, D.: "ask and thou shall receive": Reaction-based full key recovery attacks on fhe. European Symposium on Research in Computer Security pp. 457–477 (2024). https://doi.org/10.1007/978-3-031-70903-6_23
9. Checri, M., Sirdey, R., Boudguiga, A., Bultel, J.P.: On the practical CPAD security of "exact" and threshold FHE schemes and libraries. In: Advances in Cryptology – CRYPTO 2024. pp. 3–33 (2024). https://doi.org/10.1007/978-3-031-68382-4_1
10. Chenal, M., Tang, Q.: On key recovery attacks against existing somewhat homomorphic encryption schemes. In: Progress in Cryptology-LATINCRYPT 2014: Third International Conference on Cryptology and Information Security in Latin America. pp. 239–258 (2015). https://doi.org/10.1007/978-3-319-16295-9_13
11. Cheon, J.H., Choe, H., Passelègue, A., Stehlé, D., Suvanto, E.: Attacks against the ind-cpad security of exact fhe schemes. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 2505–2519 (2024). https://doi.org/10.1145/3658644.3690341

12. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security. pp. 409–437 (2017). https://doi.org/10.1007/978-3-319-70694-8_15
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020). https://doi.org/10.1007/s00145-019-09319-x
14. Chillotti, I., Gama, N., Goubin, L.: Attacking fhe-based applications by software fault injections. Cryptology ePrint Archive, Paper 2016/1164 (2016), https://eprint.iacr.org/2016/1164
15. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 617–640 (2015). https://doi.org/10.1007/978-3-662-46800-5_24
16. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144 (2012), https://eprint.iacr.org/2012/144
17. Fauzi, P., Hovd, M.N., Raddum, H.: On the ind-cca1 security of fhe schemes. Cryptography **6**(1), 13 (2022). https://doi.org/10.3390/cryptography6010013
18. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009). https://doi.org/10.1145/1536414.1536440
19. Goldwasser, S., Kalai, Y., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing. p. 555–564 (2013). https://doi.org/10.1145/2488608.2488678
20. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences **28**(2), 270–299 (1984). https://doi.org/10.1016/0022-0000(84)90070-9
21. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Attribute-based encryption for circuits. Journal of the ACM (JACM) **62**(6), 1–33 (2015). https://doi.org/10.1145/2824233
22. Grcar, J.F.: How ordinary elimination became gaussian elimination. Historia Mathematica **38**(2), 163–218 (2011). https://doi.org/10.1016/j.hm.2010.06.003
23. Holdings, A.: Building a secure system using trust-zone technology. Whitepaper (2005)
24. Intel®: Intel® trust domain extensions. Whitepaper pp. 1–9 (2022)
25. Kamara, S., Wei, L.: Garbled circuits via structured encryption. In: Financial Cryptography and Data Security - FC 2013 Workshops, USEC and WAHC 2013 (2013). https://doi.org/10.1007/978-3-642-41320-9_12
26. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische annalen **261**(ARTICLE), 515–534 (1982). https://doi.org/10.1007/bf01457454
27. Li, B., Micciancio, D.: On the security of homomorphic encryption on approximate numbers. In: Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 648–677 (2021). https://doi.org/10.1007/978-3-030-77870-5_23
28. Li, B., Micciancio, D., Schultz, M., Sorrell, J.: Securing approximate homomorphic encryption using differential privacy. In: Annual International Cryptology Conference. pp. 560–589 (2022). https://doi.org/10.1007/978-3-031-15802-5_20

29. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM (JACM) **60**(6), 1–35 (2013). https://doi.org/10.1007/978-3-642-13190-5_1
30. Ma, G., Li, H.: On the security of homomorphic encryption schemes with restricted decryption oracles. Journal of Systems Science and Complexity **37**(5), 2240–2261 (2024). https://doi.org/10.1007/s11424-024-3221-1
31. Manulis, M., Nguyen, J.: Fully homomorphic encryption beyond ind-cca1 security: Integrity through verifiability. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 63–93 (2024). https://doi.org/10.1007/978-3-031-58723-8_3
32. McKeen, F., Alexandrovich, I., Anati, I., Caspi, D., Johnson, S., Leslie-Hurd, R., Rozas, C.: Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave. In: Proceedings of the Hardware and Architectural Support for Security and Privacy 2016. pp. 1–9 (2016). https://doi.org/10.1145/2948618.2954331
33. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing. pp. 427–437 (1990). https://doi.org/10.1145/100216.100273
34. Peng, Z.: Danger of using fully homomorphic encryption: A look at microsoft seal (2019), https://arxiv.org/abs/1906.07127
35. Rackoff, C., Simon, D.R.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology. pp. 433–444 (1991). https://doi.org/10.1007/3-540-46766-1_35
36. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009). https://doi.org/10.1145/1568318.1568324
37. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical programming **66**, 181–199 (1994). https://doi.org/10.1007/BF01581144
38. Sev-Snp, A.: Strengthening vm isolation with integrity protection and more. White Paper, January **53**, 1450–1465 (2020)
39. Thibault, L.T., Walter, M.: Towards verifiable FHE in practice: Proving correct execution of TFHE's bootstrapping using plonky2. Cryptology ePrint Archive, Paper 2024/451 (2024), https://eprint.iacr.org/2024/451
40. Viand, A., Knabenhans, C., Hithnawi, A.: Verifiable fully homomorphic encryption (2023), https://arxiv.org/abs/2301.07041
41. Yao, A.C.: Protocols for secure computations. In: 23rd annual symposium on foundations of computer science (sfcs 1982). pp. 160–164 (1982). https://doi.org/10.1109/SFCS.1982.38
42. Zhang, Z., Plantard, T., Susilo, W.: Reaction attack on outsourced computing with fully homomorphic encryption schemes. In: Information Security and Cryptology-ICISC 2011. pp. 419–436 (2012). https://doi.org/10.1007/978-3-642-31912-9_28