





Reactive Correctness, sIND-CPA^D -Security and Deterministic Evaluation for TFHE

Nigel P. Smart^{1,2}   and Michael Walter²  

¹ KU Leuven, Computer Security and Industrial Cryptography, Leuven, Belgium

² Zama, Paris, France

Abstract. We examine the relationship between correctness definitions for Fully Homomorphic Encryption (FHE) and the associated security definitions. We show that reactive notions of correctness imply IND-CPA^D and sIND-CPA^D security. But that to obtain both IND-CPA^D and sIND-CPA^D security we need to use a randomized version of the evaluation procedure. Such randomized evaluation procedures cause problems in real life deployments of FHE solutions, so we then go on to show how one can de-randomize the evaluation procedure and still obtain sIND-CPA^D security in the random oracle model for the specific FHE scheme of TFHE.

Keywords: Fully Homomorphic Encryption

Contents

1	Introduction	2
2	Preliminaries	3
2.1	Notation:	3
2.2	Fully Homomorphic Encryption:	3
3	SHE/FHE Security Notions	4
4	SHE/FHE Correctness Notions	6
5	sIND-CPA^D Security for General FHE	8
6	TFHE	11
6.1	The Basic TFHE Scheme	11
6.2	Assumptions on Basic TFHE	14
6.3	Extensions to TFHE	16
7	sIND-CPA^D Security for TFHE	17
7.1	Assumptions	17
7.2	A Specific re-rand Procedure for TFHE	21
7.3	Proving TFHE.DeRandEval Provides sIND-CPA^D Security in the Random Oracle Model	22
	References	24

1 Introduction

For a long time after its introduction by Gentry in 2009 [Gen09] the “correct” security notion for Fully Homomorphic Encryption (FHE) was believed to be IND-CPA. However, in recent years it has become apparent that this is too weak a notion. The reason is that, in the real world, an attacker not only has access to ciphertexts which have been generated by the encryption algorithm (as in the case of traditional IND-CPA), it has access to ciphertexts which have been the result of applying the homomorphic evaluation of *adversarially chosen functions* on ciphertexts which have been selected adversarially. This led to the development of the notion of IND-CPA^D, [LM21]. This is not just a theoretical concern, as schemes which are not IND-CPA^D secure are vulnerable to real attacks [LM21, CSBB24, CCP⁺24].

But IND-CPA^D security may also not be enough in the case of public key FHE schemes. In [BJSW25] the notion of sIND-CPA^D was introduced. In this model the attacker not only has access to validly generated ciphertexts, i.e. ciphertexts which either result from calling an encryption oracle on a message of their choosing or result from application of the evaluation function on functions of their choosing, but also the ability to produce (valid) ciphertexts using randomness of their choosing. For public key FHE schemes it is clear that an attacker can always generate ciphertexts with randomness (for the public key encryption algorithm) of their choosing. Thus sIND-CPA^D security is clearly closer to the “correct” security definition than simple IND-CPA, or IND-CPA^D security, at least in the public key setting.

Both IND-CPA^D and sIND-CPA^D security are related to the underlying correctness notion for FHE schemes. Again, the situation for many years was that a simple correctness notion was sufficient, namely that an FHE scheme was correct if the decryption of the evaluation of a function on ciphertexts produced the correct result; where the underlying messages and a single evaluated function were chosen by the adversary. In this *standard correctness* definition, the input ciphertexts to the evaluation function are validly produced, using the public key encryption algorithm, using randomness unknown to the adversary.

However, this simple correctness notion is not sufficient, especially when taking into account the sIND-CPA^D security definition. As pointed out above, in the sIND-CPA^D model the adversary has control over encryption randomness of ciphertexts that it generates itself, which is inherent in the public key setting. It follows that the underlying correctness definition needs to reflect this. To this end, [BJSW25] introduced a new notion of ACER Correctness, where ACER stands for Adversarially Chosen Encryption Randomness. Here the above standard definition was extended to allow the ciphertext inputs to the evaluation function to be generated using randomness chosen by the adversary.

Another wrinkle is that in the IND-CPA^D and sIND-CPA^D games the adversary is able to *interactively* decide a sequence of functions to evaluate. The chosen evaluations can depend on the specific ciphertexts created via encryption calls, and those returned from prior evaluation queries. We call such a security definition *reactive*, as it is equivalent to the notion of reactive security used in the MPC literature. Clearly, the standard correctness definition and its straight-forward adaptation to the ACER setting from [BJSW25] do not reflect this and are thus not strong enough – a fact that has also previously been noted in [ABMP24]. It follows that schemes that were designed to achieve (some version of) the standard correctness definition, which is typically the case in practice, may not achieve IND-CPA^D security, let alone sIND-CPA^D security.

Our Contribution We first provide a reactive version of ACER correctness (cf. Definition 4) and clarify the relationship to sIND-CPA^D security. This can be viewed as an adaptation of a similar result for a reactive version of standard correctness and IND-CPA^D security from [ABMP24]. Specifically, we prove that our reactive version of ACER correctness and IND-CPA security imply sIND-CPA^D security.

In our main result, we then show how to achieve ACER correctness in the fully reactive sense for TFHE. The naive approach would be to simply select parameters large enough to achieve perfect correctness. However, this is highly undesirable, since this will incur a significant performance penalty. Instead, we take an approach already suggested in [BJSW25], where it was shown that randomizing the input ciphertexts before applying evaluation can provide ACER correctness for a scheme satisfying standard correctness. This is a natural idea as the randomization prevents the adversary from biasing the encryption randomness towards higher failure probabilities. We adapt this approach to TFHE. Somewhat surprisingly, we are able to show in Section 7.2 that this does not only achieve non-reactive ACER correctness, but the introduced randomness also yields fully reactive ACER correctness. Combining this with our first result above shows that the randomized scheme is sIND-CPA^D secure.

This makes, by definition, a non-deterministic evaluation operation. However, non-deterministic evaluation functions cause problems in real FHE systems as they immediately rule out trivial forms of verifiability via replication needed to avoid active attacks against the evaluation operation. See for example [Sma23] where deterministic evaluation is used to ensure security of a full system.

To address this issue, we show in Section 7.3 that de-randomizing our randomization of TFHE evaluation using a random oracle also provides a scheme which is ACER correct. Hence, our de-randomized version of the randomized evaluation procedure produces a scheme which is sIND-CPA^D secure (in the random oracle model) and has deterministic evaluation, without having to rely on large and inefficient parameters.

We remark that we need to make a set of assumptions to obtain our results. All of our assumptions are made explicit and discussed in Section 7.1. We believe all them to be reasonable and to hold in practice.

Finally, while we focus on TFHE in this work, we note that many of our techniques are straight-forward to adapt to other (exact) FHE schemes due to the structural similarities between LWE-based encryption schemes. Thus, even our results on TFHE from Section 7 have applications beyond TFHE.

2 Preliminaries

2.1 Notation:

We let $x \leftarrow X$ denote the uniformly random assignment to the variable x from the set X , assuming a uniform distribution over X . We also write $x \leftarrow y$ as shorthand for $x \leftarrow \{y\}$. If \mathcal{D} is a probability distribution over a set X , then we let $x \leftarrow \mathcal{D}$ denote sampling from X with respect to the distribution \mathcal{D} . If A is a (probabilistic) algorithm then we denote by $a \leftarrow A$ the assignment of the output of A , where the probability distribution is over the random tape of A . Elements in \mathbb{Z} , \mathbb{Z}_q or \mathbb{R} will be denoted by a and vectors by \mathbf{a} . The inner product between two vectors \mathbf{a} and \mathbf{b} will be denoted by $\mathbf{a} \cdot \mathbf{b}$.

2.2 Fully Homomorphic Encryption:

A general Fully Homomorphic Encryption (FHE) scheme will be defined by a tuple of algorithms

- $\text{FHE.Gen}(1^{\text{sec}})$: On input of a security parameter sec this outputs a secure set of parameters params for the underlying FHE scheme.
- $\text{FHE.KeyGen}(\text{params})$: A probabilistic algorithm which on input of a set of secure parameters will output a public key pk and a secret key sk .
- $\text{FHE.Enc}(m, \text{pk}; r)$: On input of a message $m \in \mathcal{M}$ and randomness $r \in \mathcal{R}$ will produce a ciphertext ct . If the randomness r is not important, or can be chosen

internally by the encryption algorithm, then we write $\text{FHE.Enc}(m, \text{pk})$.

- $\text{FHE.Dec}(\text{ct}, \text{sk})$: This deterministic algorithm, on input of a ciphertext ct and a secret key sk will return the underlying message m .
- $\text{FHE.Eval}((\text{ct}_1, \dots, \text{ct}_m), F, \text{pk})$: On input of a sequence of m ciphertexts $(\text{ct}_1, \dots, \text{ct}_m)$ and a function $F : \mathcal{M}^m \rightarrow \mathcal{M}^n$, and the public key pk , will produce a sequence of n ciphertexts $(\text{ct}'_1, \dots, \text{ct}'_n)$.

We do not discuss here either security or correctness of an FHE scheme, since these form the basis of the following two sections.

3 SHE/FHE Security Notions

The standard security notion for SHE/FHE is that of IND-CPA. Namely, for all (two-stage) adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we should have that the probability

$$\begin{aligned} \text{Adv}_{\text{IND-CPA}}^{\mathcal{A}} = \Pr \Big[& b = b' \quad : \quad b \leftarrow \{0, 1\}, (\text{pk}, \text{sk}) \leftarrow \text{FHE.KeyGen}(\text{params}), \\ & (m_0, m_1, \text{state}) \leftarrow \mathcal{A}_1(\text{pk}), \quad \text{ct} \leftarrow \text{FHE.Enc}(m_b, \text{pk}), \\ & b' \leftarrow \mathcal{A}_2(\text{ct}, \text{state}) \Big] \end{aligned}$$

should be negligible.

However, it has been established that IND-CPA security is not necessarily sufficient to secure certain FHE applications. Thus the more modern notion, see e.g., [LM21, LMSS22], is to consider a related notion called IND-CPA^D security. Formally, IND-CPA^D considers the indistinguishability experiment $\text{IndExp}^{\mathcal{A}}$ given in Figure 1, for security parameter sec . The underlying security experiment is indexed by a random bit $b \in \{0, 1\}$. A common state state is maintained, which is made up of triplets of the form (m_0, m_1, c) . The components of the j^{th} entry of state are respectively accessed as $\text{state}[j].m_0$, $\text{state}[j].m_1$, and $\text{state}[j].c$. Note, we present a generalized variant of IND-CPA^D security in which the evaluation function can take an arbitrary m -input to n -output function; this is done in order to ensure compatibility with the evaluation function of FHE schemes which we consider in this paper.

The IND-CPA^D security model captures the scenario where a user can only submit honestly generated (or evaluated) ciphertexts for decryption. In particular, the model implicitly assumes that, on input a m -variate function g and m indexes j_1, \dots, j_m , the evaluation oracle returns a *valid* encryption of $g(\text{state}[j_1].m_b, \dots, \text{state}[j_m].m_b)$ —or at least, *with high probability*. Such an encryption scheme is termed *computationally correct*, see Definition 3 below.

Informally, an FHE scheme is *perfectly* secure for the IND-CPA^D security notion if the best an attacker can do is to guess the value of b at random in the above experiment (i.e., the attacker outputs a random bit b'). With such a strategy, the attacker will recover the value of b with probability $\frac{1}{2}$. The advantage is therefore defined as the distance between the probability that the guess $b' = b$ and $\frac{1}{2}$. This is formalized as

Definition 1. A public-key FHE scheme is IND-CPA^D-secure if for any PPT adversary \mathcal{A} , it holds that

$$\text{Adv}_{\text{IND-CPA}^D}^{\mathcal{A}} = \left| \Pr \left[\text{IndExp}_b^{\mathcal{A}}(\text{sec}) = 1 \right] - \frac{1}{2} \right|$$

is negligible in security parameter sec .

In [BJSW25] it is argued that even IND-CPA^D is not a strong enough a security definition; especially in the public key setting (which is what we consider in this document). The reason is that in the corresponding security game, an adversary can only submit fresh ciphertexts to the state state that have been generated using the encryption oracle,

$\text{IndExp}_b^{\mathcal{A}}(\text{sec})$

1. For security parameter sec the parameter generation algorithm is run to obtain params , i.e. $\text{params} \leftarrow \text{FHE.Gen}(1^{\text{sec}})$. The key generation algorithm is then run to obtain keys pk , and sk ; $(\text{pk}, \text{sk}) \leftarrow \text{FHE.KeyGen}(\text{params})$. Let \mathcal{M} denote the message space.
2. The adversary \mathcal{A} receives pk and is given access to three oracles sharing a common state state initialized to \emptyset :
 - An encryption oracle Enc that on input a pair of messages $(m_0, m_1) \in \mathcal{M} \times \mathcal{M}$ returns the ciphertext $c \leftarrow \text{FHE.Enc}(m_b, \text{pk})$. The state is updated as $\text{state} \leftarrow \text{state} \cup (m_0, m_1, c)$.
 - An evaluation oracle Eval that on input of a sequence of m indexes $(j_1, \dots, j_m) \in \{1, \dots, |\text{state}|\}^m$ and a function $g: \mathcal{M}^m \rightarrow \mathcal{M}^n$ returns the n ciphertexts

$$(c_1, \dots, c_n) \leftarrow \text{FHE.Eval}((\text{state}[j_1].c, \dots, \text{state}[j_m].c), g, \text{pk}).$$

The state is updated as follows

- Compute $(m_{0,1}, \dots, m_{0,n}) \leftarrow g(\text{state}[j_1].m_0, \dots, \text{state}[j_m].m_0)$.
 - Compute $(m_{1,1}, \dots, m_{1,n}) \leftarrow g(\text{state}[j_1].m_1, \dots, \text{state}[j_m].m_1)$.
 - $\text{state} \leftarrow \text{state} \cup \{(m_{0,i}, m_{1,i}, c_i)\}_{i=1}^n$.
 - A decryption oracle Dec that on input an index $j \in \{1, \dots, |\text{state}|\}$ checks whether $\text{state}[j].m_0 = \text{state}[j].m_1$ and, if so, returns $\text{FHE.Dec}(\text{state}[j].c, \text{sk})$.
3. The adversary \mathcal{A} interacts with the oracles and eventually outputs a bit b' .
 4. The output of the experiment is defined to be one if $b' = b$, and zero otherwise.

Figure 1: The IND-CPA^D security experiment $\text{IndExp}_b^{\mathcal{A}}(\text{sec})$.

which generates *the randomness itself*, following the specifications of the scheme. This is very hard, if not impossible, to enforce in applications where the adversary may generate ciphertexts (which is almost always the case in public key setting): even if the adversary is forced to prove the well-formedness of the ciphertexts, as, for example, in the protocol in [Sma23] or, implicitly, in the construction in [BCF⁺25]. Such proofs do not prove that the encryption randomness was indeed chosen according to the prescribed distribution. In order to fix this, we need the sIND-CPA^D notion.

As formalized in Figure 2, the sIND-CPA^D model is obtained by modifying the security experiment from Figure 1 by giving the adversary access to another encryption oracle Enc' . On input of a message $m \in \mathcal{M}$ and encryption randomness seed seed , this oracle returns the ciphertext $c \leftarrow \text{FHE.Enc}(m, \text{pk}, \text{seed})$, and updates the state by setting $\text{state} \leftarrow \text{state} \cup (m, m, c)$. Note that we cannot simply allow the adversary to submit the randomness seed for the Left-or-Right-type oracle Enc , since that would render the definition unachievable. Again we present a generalization of the definition in [BJSW25] so as to accommodate evaluations on functions which produce multiple outputs.

Definition 2. A public-key FHE scheme is sIND-CPA^D if for any PPT adversary \mathcal{A} , it holds that

$$\text{Adv}_{\text{sIND-CPA}^D}^{\mathcal{A}} = \left| \Pr \left[\text{sIndExp}_b^{\mathcal{A}}(\text{sec}) = 1 \right] - \frac{1}{2} \right|$$

is negligible in security parameter sec .

$\text{sIndExp}_b^A(\text{sec})$

1. For security parameter sec the parameter generation algorithm is run to obtain params , i.e. $\text{params} \leftarrow \text{FHE.Gen}(1^{\text{sec}})$. The key generation algorithm is then run to obtain keys pk , and sk ; $(\text{pk}, \text{sk}) \leftarrow \text{FHE.KeyGen}(\text{params})$. Let \mathcal{M} denote the message space and \mathcal{R} the randomness space of seeds for the encryption algorithm.
2. The adversary \mathcal{A} receives pk and is given access to three oracles sharing a common state state initialized to \emptyset :
 - An encryption oracle Enc that on input a pair of messages $(m_0, m_1) \in \mathcal{M} \times \mathcal{M}$ returns the ciphertext $c \leftarrow \text{FHE.Enc}(m_b, \text{pk})$. The state is updated as $\text{state} \leftarrow \text{state} \cup (m_0, m_1, c)$.
 - Another encryption oracle Enc' that on input a message and randomness seed $(m, \text{seed}) \in \mathcal{M} \times \mathcal{R}$ returns the ciphertext $c \leftarrow \text{FHE.Enc}(m, \text{pk}, \text{seed})$. The state is updated as $\text{state} \leftarrow \text{state} \cup (m, m, c)$.
 - An evaluation oracle Eval that on input of a sequence of m indexes $(j_1, \dots, j_m) \in \{1, \dots, |\text{state}|\}^m$ and a function $g: \mathcal{M}^m \rightarrow \mathcal{M}^n$ returns the n ciphertexts

$$(c_1, \dots, c_n) \leftarrow \text{FHE.Eval}((\text{state}[j_1].c, \dots, \text{state}[j_m].c), g, \text{pk}).$$

The state is updated as follows

- Compute $(m_{0,1}, \dots, m_{0,n}) \leftarrow g(\text{state}[j_1].m_0, \dots, \text{state}[j_m].m_0)$.
 - Compute $(m_{1,1}, \dots, m_{1,n}) \leftarrow g(\text{state}[j_1].m_1, \dots, \text{state}[j_m].m_1)$.
 - $\text{state} \leftarrow \text{state} \cup \{(m_{0,i}, m_{1,i}, c_i)\}_{i=1}^n$.
 - A decryption oracle Dec that on input an index $j \in \{1, \dots, |\text{state}|\}$ checks whether $\text{state}[j].m_0 = \text{state}[j].m_1$ and, if so, returns $\text{FHE.Dec}(\text{state}[j].c, \text{sk})$.
3. The adversary \mathcal{A} interacts with the oracles and eventually outputs a bit b' .
 4. The output of the experiment is defined to be one if $b' = b$, and zero otherwise.

Figure 2: The sIND-CPA^D security experiment $\text{sIndExp}_b^A(\text{sec})$.

4 SHE/FHE Correctness Notions

Correctness of homomorphic computation is, of course, crucial to any deployment of a FHE scheme; what is less obvious is that various (non-obvious) definitions of correctness are also important in order to achieve IND-CPA^D and sIND-CPA^D security, as we shall see later.

The simplest notion of correctness is that of *perfect correctness* which requires

$$\Pr \left[\text{FHE.Dec}(\text{FHE.Eval}(\text{FHE.Enc}(m_1, \text{pk}), \dots, \dots, \text{FHE.Enc}(m_k, \text{pk}), g, \text{pk}), \text{sk}) = g(m_1, \dots, m_k) \right] = 1 \quad (1)$$

for all (m_1, \dots, m_k) in the message space, all functions g which are desired to be homomorphically computable¹, all $(\text{sk}, \text{pk}) \leftarrow \text{FHE.KeyGen}(\text{params})$, and all $\text{params} \leftarrow \text{FHE.Gen}(1^{\text{sec}})$. However, for many schemes, performance can be improved if one relaxes the definition such that Eq. (1) is only required to hold with probability $1 - \epsilon$ for small ϵ

¹i.e. all functions in the case of a FHE scheme.

over the randomness of FHE.Gen , FHE.KeyGen , FHE.Enc and FHE.Eval . In the literature, this notion is typically referred to as *statistical correctness*.

The previous simple notion of correctness suffices for many applications, but when considering IND-CPA^D and sIND-CPA^D security one needs more elaborate definitions. As the security games we consider (IND-CPA^D and sIND-CPA^D , see Section 3) are *reactive* in the sense that the adversary may choose the function g adaptively, we need such a reactive version of correctness for our security proofs of IND-CPA^D and sIND-CPA^D security. Similar correctness notions have been considered previously in [ABMP24]. Thus we now give more general notions of correctness.

$\text{Cor}^{\mathcal{A}}(\text{sec})$

1. For security parameter sec the parameter generation algorithm is run to obtain params , i.e. $\text{params} \leftarrow \text{FHE.Gen}(1^{\text{sec}})$. The key generation algorithm is then run to obtain keys pk and sk ; $(\text{pk}, \text{sk}) \leftarrow \text{FHE.KeyGen}(\text{params})$. The message space is denoted by \mathcal{M} and the space of encryption randomness by \mathcal{R} .
 2. The adversary \mathcal{A} receives pk and is given access to two oracles sharing a common state state initialized to \emptyset :
 - Encryption oracle Enc that on input a message $m \in \mathcal{M}$ returns the ciphertext $c \leftarrow \text{FHE.Enc}(m, \text{pk})$. The state is updated as $\text{state} \leftarrow \text{state} \cup (m, c)$. If we have $\text{FHE.Dec}(c, \text{sk}) \neq m$ then game terminates and outputs one.
 - An evaluation oracle Eval that on input of a sequence of m indexes $(j_1, \dots, j_m) \in \{1, \dots, |\text{state}|\}^m$ and a function $g: \mathcal{M}^m \rightarrow \mathcal{M}^n$ executes the following steps
 - Computes the ciphertexts $(c_1, \dots, c_n) \leftarrow \text{FHE.Eval}((\text{state}[j_1].c, \dots, \text{state}[j_m].c), g, \text{pk})$.
 - Compute the messages $(m_1, \dots, m_n) \leftarrow g(\text{state}[j_1].m, \dots, \text{state}[j_m].m)$.
 - If for any index $i \in [1, \dots, n]$ we have $\text{FHE.Dec}(c_i, \text{sk}) \neq m_i$ then game terminates and outputs one.
 - Add $\{(m_i, c_i)\}_{i=1}^n$ to the game state and return (c_1, \dots, c_n) to \mathcal{A} .
- If \mathcal{A} aborts, output zero.

Figure 3: The security game for FHE correctness.

The first of these we call *computational correctness*, see Definition 3 and the related Figure 3. As the name suggests, it relaxes statistical correctness and only requires correctness to hold against PPT adversaries. On the other hand, Definition 3 strengthens statistical correctness by allowing the adversary to adaptively choose the function g depending on the ciphertexts, and also use outputs of FHE.Eval as inputs to FHE.Eval . This definition of correctness more closely matches how FHE schemes are used in practice; namely data is encrypted, functions are evaluated, other data is encrypted, and then other functions are chosen to be evaluated.

Definition 3 (Computational Correctness). A public-key FHE scheme $(\text{FHE.Gen}, \text{FHE.KeyGen}, \text{FHE.Enc}, \text{FHE.Dec}, \text{FHE.Eval})$ is said to be *computationally correct* if for any PPT adversary \mathcal{A} it holds that

$$\Pr \left[\text{Cor}^{\mathcal{A}}(\text{sec}) = 1 \right]$$

is negligible in sec , where $\text{Cor}^{\mathcal{A}}(\text{sec})$ is the game in Figure 3.

We do not explicitly need this definition of Computational Correctness in this work, as it is mainly tied to the definition of IND-CPA^D security, but we include it here as it is a useful stepping stone towards the definitions that we actually require. The following lemma from [ABMP24] shows the connection between Computational Correctness and IND-CPA^D security.

Lemma 1. *Any public-key FHE scheme that is IND-CPA secure and computationally correct (cf. Definition 3) is secure in the IND-CPA^D sense.*

In our work, we require (computational) ACER correctness, for sIND-CPA^D security. Unlike Computational Correctness, the definition of (computational) ACER correctness, enables us to argue about correctness in the presence of Adversarially Chosen Encryption Randomness; see Definition 4 and the associated Figure 4. This notion reflects the setting of sIND-CPA^D security, where the adversary has access to an oracle that takes encryption randomness as input. This definition is a generalization of the definition of ACER correctness from the statistical, non-reactive version given in [BJSW25] to a computational, reactive notion, which accomodates evaluations on functions which produce multiple outputs.

Definition 4 (ACER Correctness). A public-key FHE scheme (FHE.Gen, FHE.KeyGen, FHE.Enc, FHE.Dec, FHE.Eval) is said to be *computationally ACER correct* if for any PPT adversary \mathcal{A} it holds that

$$\Pr \left[\text{ACERCor}^{\mathcal{A}}(\text{sec}) = 1 \right]$$

is negligible in sec , where $\text{ACERCor}^{\mathcal{A}}(\text{sec})$ is the game in Figure 4.

5 sIND-CPA^D Security for General FHE

Our first result on sIND-CPA^D security is inspired by a result in [BJSW25], which we generalize to the case of computational ACER correctness. The results in this section apply to any general FHE scheme as defined in this document.

Theorem 1. *Any public-key FHE scheme that is IND-CPA secure and computationally ACER correct is secure in the sIND-CPA^D sense.*

Proof. We define a modified sIND-CPA^D game, the *sim-IND-CPA^D* game in Figure 5, that corresponds to the game that the reduction simulates in the proof of Theorem 1. We then show that the sIND-CPA^D and the *sim-IND-CPA^D* game are computationally indistinguishable assuming that the scheme is computationally ACER correct, see Lemma 2 below. Note that the only difference between the sIND-CPA^D game and the *sim-IND-CPA^D* game is how the decryption oracle computes the message to return to the adversary: In the sIND-CPA^D game, the oracle decrypts the corresponding ciphertext in the state. In the *sim-IND-CPA^D* game, the oracle will actually simply look up the message in the state and return it. Then we show, see Lemma 3, that if the FHE scheme is IND-CPA secure then the advantage of the adversary breaking the *sim-IND-CPA^D* is negligible. Thus from the two Lemmas the theorem follows. \square

Lemma 2. *Let (FHE.Gen, FHE.KeyGen, FHE.Enc, FHE.Dec, FHE.Eval) be a public-key homomorphic encryption scheme that is computationally ACER correct. Then for any $b \in \{0, 1\}$ the $\text{sIndExp}_b^{\mathcal{A}}$ experiment and the $\text{sim-IndExp}_b^{\mathcal{A}}$ experiment are computationally indistinguishable.*

ACERCor^A(sec)

1. For security parameter sec the parameter generation algorithm is run to obtain params , i.e. $\text{params} \leftarrow \text{FHE.Gen}(1^{\text{sec}})$. The key generation algorithm is then run to obtain keys pk and sk ; $(\text{pk}, \text{sk}) \leftarrow \text{FHE.KeyGen}(\text{params})$. The message space is denoted by \mathcal{M} and the space of encryption randomness by \mathcal{R} .
 2. The adversary \mathcal{A} receives pk and is given access to two oracles sharing a common state state initialized to \emptyset :
 - Encryption oracle Enc that on input a message and randomness $(m, r) \in \mathcal{M} \times \mathcal{R}$ returns the ciphertext $c \leftarrow \text{FHE.Enc}(m, \text{pk}; r)$. The state is updated as $\text{state} \leftarrow \text{state} \cup (m, c)$. If we have $\text{FHE.Dec}(c, \text{sk}) \neq m$ then game terminates and outputs one.
 - An evaluation oracle Eval that on input of a sequence of m indexes $(j_1, \dots, j_m) \in \{1, \dots, |\text{state}|\}^m$ and a function $g: \mathcal{M}^m \rightarrow \mathcal{M}^n$ executes the following steps
 - Computes the ciphertexts $(c_1, \dots, c_n) \leftarrow \text{FHE.Eval}((\text{state}[j_1].c, \dots, \text{state}[j_m].c), g, \text{pk})$.
 - Compute the messages $(m_1, \dots, m_n) \leftarrow g(\text{state}[j_1].m, \dots, \text{state}[j_m].m)$.
 - If for any index $i \in [1, \dots, n]$ we have $\text{FHE.Dec}(c_i, \text{sk}) \neq m_i$ then game terminates and outputs one.
 - Add $\{(m_i, c_i)\}_{i=1}^n$ to the game state and return (c_1, \dots, c_n) to \mathcal{A} .
- If \mathcal{A} aborts, output zero.

Figure 4: The security game for computational ACER correctness.

Proof. Let \mathcal{A} be an adversary that is able to distinguish the $\text{sIndExp}_b^{\mathcal{A}}$ experiment and the $\text{sim-IndExp}_b^{\mathcal{A}}$ experiment for any $b \in \{0, 1\}$. Note that, this means that at some point \mathcal{A} makes a query to the decryption oracle where the response differs between the $\text{sIndExp}_b^{\mathcal{A}}$ experiment and the $\text{sim-IndExp}_b^{\mathcal{A}}$ experiment, since otherwise the experiments are identical. In that case we have that $\text{state}[j].m_b \neq \text{FHE.Dec}(\text{state}[j].c, \text{sk})$ and \mathcal{A} can distinguish with probability one. Let p_e be the probability of that event. Then the probability p_w of \mathcal{A} in winning the corresponding distinguishing game is $p_w = (1 - p_e)/2 + p_e = (1 + p_e)/2$ and its advantage is $2p_w - 1 = p_e$. So if \mathcal{A} 's advantage is non-negligible, the probability p_e is also non-negligible.

We build an adversary \mathcal{B} against the ACER correctness of the FHE scheme. For simplicity, we assume that \mathcal{A} never makes a query to its sIND-CPA^D decryption oracle, where the oracle returns \perp . This is without loss of generalization, since this only happens when $S[j].m_0 \neq S[j].m_1$ and \mathcal{A} could check this condition itself. Recall that \mathcal{B} has an encryption and evaluation oracle available, while \mathcal{A} has an additional (left-right) encryption oracle and a decryption oracle. The algorithm \mathcal{B} runs \mathcal{A} and handles the queries of \mathcal{A} in the following way, while keeping track of the game state state .

- $\text{Enc}(m_0, m_1)$: Choose $\text{seed} \leftarrow \mathcal{R}$ and forward $\text{Enc}(m_b; \text{seed})$ to the encryption oracle of \mathcal{B} to obtain c and return it to \mathcal{A} .
- $\text{Enc}'(m; \text{seed})$: Forward $\text{Enc}(m; \text{seed})$ to the encryption oracle of \mathcal{B} to obtain c and return it to \mathcal{A} .
- $\text{Eval}(j_1, \dots, j_m, g)$: Forward $\text{Eval}(j_1, \dots, j_m, g)$ to evaluation oracle of \mathcal{B} in order to obtain (c_1, \dots, c_n) and return them to \mathcal{A} .
- $\text{Dec}(j)$: return $\text{state}[j].m_b$.

Since \mathcal{B} simulates the $\text{sIndExp}_b^{\mathcal{A}}$ experiment to \mathcal{A} , with probability p_e at some point during

sim-IndExp_b^A(sec)

1. For security parameter `sec` the parameter generation algorithm is run to obtain `params`, i.e. `params` \leftarrow `FHE.Gen`(1^{sec}). The key generation algorithm is then run to obtain keys `pk` and `sk`; `(pk, sk)` \leftarrow `FHE.KeyGen`(`params`). The message space is denoted by \mathcal{M} and the space of encryption randomness by \mathcal{R} .
2. The adversary \mathcal{A} receives `pk` and is given access to three oracles sharing a common state `state` initialized to \emptyset :
 - An encryption oracle `Enc` that on input a pair of messages $(m_0, m_1) \in \mathcal{M} \times \mathcal{M}$ returns the ciphertext $c \leftarrow \text{FHE.Enc}(m_b, \text{pk})$. The state is updated as `state` \leftarrow `state` \cup (m_0, m_1, c) .
 - Another encryption oracle `Enc'` that on input a message and randomness seed $(m, \text{seed}) \in \mathcal{M} \times \mathcal{R}$ returns the ciphertext $c \leftarrow \text{FHE.Enc}(m, \text{pk}, \text{seed})$. The state is updated as `state` \leftarrow `state` \cup (m, m, c) .
 - An evaluation oracle `Eval` that on input of a sequence of m indexes $(j_1, \dots, j_m) \in \{1, \dots, |\text{state}|\}^m$ and a function $g: \mathcal{M}^m \rightarrow \mathcal{M}^n$ returns the n ciphertexts

$$(c_1, \dots, c_n) \leftarrow \text{FHE.Eval}((\text{state}[j_1].c, \dots, \text{state}[j_m].c), g, \text{pk}).$$

The state is updated as follows

- Compute $(m_{0,1}, \dots, m_{0,n}) \leftarrow g(\text{state}[j_1].m_0, \dots, \text{state}[j_m].m_0)$.
 - Compute $(m_{1,1}, \dots, m_{1,n}) \leftarrow g(\text{state}[j_1].m_1, \dots, \text{state}[j_m].m_1)$.
 - `state` \leftarrow `state` $\cup \{(m_{0,i}, m_{1,i}, c_i)\}_{i=1}^n$.
 - A decryption oracle `Dec` that on input an index $j \in \{1, \dots, |\text{state}|\}$ checks whether `state`[j]. m_0 = `state`[j]. m_1 and, if so, returns `state`[j]. m_b .
3. The adversary \mathcal{A} interacts with the oracles and eventually outputs a bit b' .
 4. The output of the experiment is defined to be one if $b' = b$, and zero otherwise.

Figure 5: The simulatable sIND-CPA^D security experiment sim-IndExp_b^A(sec).

the game we have `state`[j]. $m_b \neq \text{FHE.Dec}(\text{state}[j].c, \text{sk})$. The only way that `state`[j]. c could have ended up in the game state is through a query to \mathcal{B} 's encryption or evaluation oracle. Both oracles check if `state`[j]. $m \neq \text{FHE.Dec}(\text{state}[j].c, \text{sk})$ and thus the ACERCor game would have terminated already and \mathcal{B} would have succeeded. We conclude that \mathcal{B} wins the ACERCor game with probability p_e , which was assumed to be non-negligible. \square

Lemma 3. *Let $(\text{FHE.Gen}, \text{FHE.KeyGen}, \text{FHE.Enc}, \text{FHE.Dec}, \text{FHE.Eval})$ be a public-key homomorphic encryption scheme that is IND-CPA secure. Then, for any PPT adversary \mathcal{A} ,*

$$\left| \Pr \left[\text{sim-IndExp}_b^{\mathcal{A}}(\text{sec}) = 1 \right] - \frac{1}{2} \right|$$

is negligible in sec.

Proof. Let \mathcal{A} be an adversary that is able to win the sim-IND-CPA^D game with non-negligible advantage and let q be the number of queries \mathcal{A} makes to its encryption oracle. We construct an IND-CPA adversary \mathcal{B} that has non-negligible advantage. We first define a set of $q + 1$ hybrids $\{\mathcal{H}_i\}_{i=0}^q$, where in \mathcal{H}_i the encryption oracle encrypts m_0 in the first $q - i$ queries, while it encrypts m_1 in all other queries. Note that $\mathcal{H}_0 = \text{sim-IndExp}_0^{\mathcal{A}}$ and $\mathcal{H}_q = \text{sim-IndExp}_1^{\mathcal{A}}$. By the hybrid argument, there must be an i such that the advantage of \mathcal{A} in distinguishing \mathcal{H}_i and \mathcal{H}_{i+1} is non-negligible. Note that the hybrids are all simulatable

without the secret key sk due to the absence of any decryption operation. So \mathcal{B} will simulate \mathcal{H}_i , but for the $(q - i)$ -th query, it will send (m_0, m_1) to its own encryption oracle. When receiving $c_b = \text{FHE.Enc}(m_b, \text{pk})$, \mathcal{B} records c_b in the game state and from that point on will simulate \mathcal{H}_{i+1} . It is easy to see that \mathcal{B} simulate \mathcal{H}_i if $b = 0$ and \mathcal{H}_{i+1} otherwise. It follows that \mathcal{B} 's advantage in winning the IND-CPA game is the same as \mathcal{A} 's advantage in distinguishing \mathcal{H}_i and \mathcal{H}_{i+1} . \square

It is folklore that parameter selection for FHE schemes is a delicate balancing act between correctness, security and efficiency. Theorem 1 shows that for applications that require sIND-CPA^D security, parameters must be chosen to ensure IND-CPA security and computational ACER correctness. A straight-forward way of achieving ACER correctness is by selecting parameters such that the scheme is perfectly correct, at least with overwhelming probability over key generation. However, this is typically undesirable from an efficiency standpoint, as it requires worst-case bounds on many variables leading to large parameters.

Another approach was proposed in [BJSW25]: in certain settings, ACER correctness may be achieved by randomizing the evaluation procedure. Specifically, if a scheme is correct under honestly generated ciphertexts and there is a way to randomize ciphertexts, then ACER-type correctness is obtained by randomizing the ciphertext prior to the evaluation call. We show how to do this for TFHE in Section 7.2 (after first covering TFHE in Section 6).

As already pointed out in the introduction, in many applications it can be very advantageous for the evaluation procedure to be deterministic. So, in Section 7.3 we show that our randomization can be de-randomized using random oracles. In the end, this will result in a sIND-CPA^D secure version of TFHE with deterministic evaluation without having to rely on large parameters that ruin efficiency. In particular, compared to typical (non- sIND-CPA^D -secure) instantiations, the parameters need hardly any change at all and the (de-randomized) randomization procedure is extremely efficient in comparison to the evaluation.

6 TFHE

After having dealt with general FHE schemes, we now turn our attention to TFHE in particular. The key issue we want to address is to enable sIND-CPA^D security, whilst not requiring the need to perform (true) re-randomization during the evaluation procedure. In the next section we show how using a random oracle to de-randomization the implicit re-randomization needed can be achieved for TFHE. In this section we cover enough details of the TFHE encryption scheme in order to understand the remainder of this contribution.

For those who require a more in-depth discussion of the algorithms should consult [CGGI16, CGGI20, CLOT21, Joy24, BdBB⁺25]. We only present those algorithms which are pertinent to our discussion which follows, we do not discuss the precise details of the decryption or evaluation operations. We first present the basic scheme, then we present the underlying assumptions we make about this scheme (i.e. that it is IND-CPA secure), then finally we present our modifications which will enable us to show that TFHE can be made sIND-CPA^D secure.

6.1 The Basic TFHE Scheme

A ciphertext modulus in TFHE is a power of two $Q = 2^K$, for some value K (usually equal to 64). The plaintext modulus $P = 2^p$ is also a power of two, but much smaller than Q . Typically P is either 8 or 32, and we let $\Delta = Q/P$ denote the ratio. There are three main LWE dimensions associated with the TFHE scheme, $\hat{\ell}$, ℓ and $w \cdot N$. The value $\hat{\ell}$ is a power of two, the value ℓ can be any integer, w is a small value (typically one, two or three)

and N is also a power of two. Due to our public key encryption methodology, a fresh encryption will have LWE dimension $\hat{\ell}$. However, this will immediately be converted to one whose LWE dimension is either ℓ (called a ciphertext of $\text{type} = \text{LWE}$) or $w \cdot N$ (called a ciphertext of $\text{type} = \text{F-GLWE}$). The exact choice of type is an implementation decision.

We let \mathcal{D}^n denote a suitable noise distribution, which returns “small” vectors of dimension n . The exact definition of the noise distribution is unimportant for this paper, but it should clearly be chosen to ensure security of the resulting LWE distributions.

Each LWE dimension has an associated secret key, but for the purposes of this paper we are only interested in the secret key $\hat{\mathbf{s}} \in \{0, 1\}^{\hat{\ell}}$ associated with public key encryption. The main public key $(\mathbf{pk}_a, \mathbf{pk}_b)$ is chosen so that \mathbf{pk}_a is uniformly randomly selected from $(\mathbb{Z}/(Q))^{\hat{\ell}}$ and

$$\mathbf{pk}_b \leftarrow \mathbf{pk}_a \odot \overset{\leftrightarrow}{\hat{\mathbf{s}}} + \mathbf{e},$$

where \odot denotes the ring multiplication in the ring $\mathbb{Z}/(Q)[X]/(X^{\hat{\ell}} + 1)$, $\overset{\leftrightarrow}{\mathbf{a}}$ denotes the vector with the coefficients in the reverse order, so that if $\mathbf{a} = (a_0, \dots, a_{N-1})$ then $\overset{\leftrightarrow}{\mathbf{a}} = (a_{N-1}, \dots, a_0)$, and $\mathbf{e} \leftarrow \mathcal{D}^{\hat{\ell}}$ denotes a “small” error vector. See [Joy24] for details. We assume an algorithm $\text{TFHE.KeyGen}(\text{params})$ which generates the above secret key and public key material, as well as additional secret key and public key material needed for the other algorithms within TFHE (see [BdBB⁺25] for further details).

The Public Key TFHE Scheme: Encryption

TFHE.Enc-Sub(m, \mathbf{pk} ; seed):

Here \mathbf{pk} is the public key $(\mathbf{pk}_a, \mathbf{pk}_b)$. The value **seed** specifies the random seed used to generate the internal random values in the following; this will be dropped from the notation when it is not important. The output is an LWE ciphertext of dimension $\hat{\ell}$.

1. $\mathbf{r} \leftarrow \{0, 1\}^{\hat{\ell}}, \mathbf{e}_1 \leftarrow \mathcal{D}^{\hat{\ell}}, e_2 \leftarrow \mathcal{D}^1$.
2. $\mathbf{a} \leftarrow \mathbf{pk}_a \odot \overset{\leftrightarrow}{\mathbf{r}} + \mathbf{e}_1$.
3. $b \leftarrow \mathbf{pk}_b \cdot \mathbf{r} + e_2 + \Delta \cdot m$.
4. Return (\mathbf{a}, b) . In some applications of this function we will require the return of $((\mathbf{a}, b), (\mathbf{r}, \mathbf{e}_1, e_2))$ in order to prove knowledge and correctness of the encryption via the ZKPoKs given in [Lib24].

TFHE.Enc(m, \mathbf{pk} ; seed):

Here \mathbf{pk} is the public key $(\mathbf{pk}_a, \mathbf{pk}_b, \text{PKSK})$. The output is an LWE ciphertext of type type . Again we drop **seed** from the notation when this is not important.

1. $\mathbf{ct}' \leftarrow \text{TFHE.Enc-Sub}(m, (\mathbf{pk}_a, \mathbf{pk}_b); \text{seed})$.
2. $\mathbf{ct} \leftarrow \text{TFHE.DimensionSwitch}(\mathbf{ct}', \text{PKSK})$.
3. Return \mathbf{ct} .

Figure 6: The Public Key TFHE Scheme: Encryption

Along with other key material there is a key PKSK (Public-key Key-Switching Key) which enables a function $\text{TFHE.DimensionSwitch}(\mathbf{ct}, \text{PKSK})$ to be defined which takes an LWE ciphertext of dimension $\hat{\ell}$ and returns an LWE ciphertext of type type , i.e. one of dimension ℓ or $w \cdot N$, encrypting the same message. To encrypt a message $m \in \mathbb{Z}/(P)$ (see Figure 6) we use the method of Joye [Joy24] in the sub-procedure $\mathbf{ct}' \leftarrow \text{TFHE.Enc-Sub}(m, \mathbf{pk})$, and then switch the output \mathbf{ct} to be of dimension ℓ or $w \cdot N$ using $\mathbf{ct} \leftarrow \text{TFHE.DimensionSwitch}(\mathbf{ct}', \text{PKSK})$. The output ciphertext \mathbf{ct} is of the form

(\mathbf{a}, b) , where \mathbf{a} is a vector over $\mathbb{Z}/(Q)$, of dimension ℓ if $\text{type} = \text{LWE}$ or of dimension $w \cdot N$ if $\text{type} = \text{F-GLWE}$, and the value b is an integer in $\mathbb{Z}/(Q)$ of the form $b = \mathbf{a} \cdot \mathbf{s} + e + \Delta \cdot m$ for some “small” noise value e .

When using the ZKPoKs from [Lib24] to prove correctness and knowledge of the message underlying an encryption we actually append a ZKPoK of correct encryption to the execution of `TFHE.Enc-Sub`, i.e. before the execution of the `TFHE.DimensionSwitch` operation. This proof of knowledge shows that the randomizers $(\mathbf{r}, \mathbf{e}_1, e_2)$ used in `TFHE.Enc-Sub` are valid, and that the encryptor knows the underlying plaintext m . The values $(\mathbf{r}, \mathbf{e}_1, e_2)$ are obtained, in practice, via a XOF from the input `seed` to `TFHE.Enc`. Thus, if we model the XOF as a random oracle, the use of the ZKPoKs will not only allow any simulator to extract the values $(m, \mathbf{r}, \mathbf{e}_1, e_2)$, but it will also allow a simulator to extract the `seed` values from $(\mathbf{r}, \mathbf{e}_1, e_2)$. This ability to extract `seed` from the ZKPoKs, via the random oracle, will be important in our later proofs.

There is an algorithm `TFHE.Dec(ct, s)` which takes a ciphertext $\mathbf{ct} = (\mathbf{a}, b)$ of type `type` and decrypts it under the secret key \mathbf{s} . This algorithm computes $p \leftarrow b - \mathbf{a} \cdot \mathbf{s} \pmod{Q}$ and then writes p as $p = e + \Delta \cdot m$, and so extracts $m \pmod{P}$ via rounding.

Basic Homomorphic Operations: Addition and Scalar Multiplication

`TFHE.Add(ct1, ct2):`

We have input LWE/FGLWE ciphertexts \mathbf{ct}_1 and \mathbf{ct}_2 encrypting messages $m_1 \in \mathbb{Z}/(P)$ and $m_2 \in \mathbb{Z}/(P)$, respectively. The output LWE/FGLWE ciphertext is an encryption of $m_1 + m_2$.

1. Write $\mathbf{ct}_1 = (\mathbf{a}_1, b_1)$.
2. Write $\mathbf{ct}_2 = (\mathbf{a}_2, b_2)$.
3. $\mathbf{a}' \leftarrow \mathbf{a}_1 + \mathbf{a}_2$.
4. $b' \leftarrow b_1 + b_2$.
5. Return (\mathbf{a}', b') .

`TFHE.ScalarMult(α , ct):`

We have \mathbf{ct} an LWE/FGLWE ciphertext encrypting $m \in \mathbb{Z}/(P)$ and $\alpha \in \mathbb{Z}/(P)$; where recall $\mathbb{Z}/(P) = \{-P/2 + 1, \dots, P/2\}$. The output LWE/FGLWE ciphertext is an encryption of $\alpha \cdot m$.

1. Write $\mathbf{ct} = (\mathbf{a}, b)$.
2. Consider α as an integer (i.e. in \mathbb{Z}).
3. $\mathbf{a}' \leftarrow \alpha \cdot \mathbf{a}$ (i.e. scalar multiplication of the vector \mathbf{a} by the scalar α).
4. $b' \leftarrow \alpha \cdot b$.
5. Return (\mathbf{a}', b') .

Figure 7: Basic Homomorphic Operations: Addition and Scalar Multiplication

With just the previous set up, one has (assuming parameters are chosen correctly) a simple linearly homomorphic encryption scheme which can evaluate small linear circuits. This is done using the algorithms in Figure 7. Such a linear function will be called admissible if it is guaranteed to result in a plaintext value whose top bit is set to zero. The reason for requiring the top bit set to zero, is that if this is guaranteed we can then apply a Programmable Bootstrapping (PBS) operation to apply an arbitrary function f homomorphically to the ciphertext. This is done using an operation denoted $\mathbf{ct}' \leftarrow \text{TFHE.PBS}(\mathbf{ct}, f, \text{PK})$, where PK now contains some extra public key material (which will not concern us) which enables the computation of this operation. If the input ciphertext \mathbf{ct} of type `type` encrypts a value m (with most-significant bit set to zero) then the output

ciphertext \mathbf{ct}' is also of type \mathbf{type} and encrypts the value $f(m)$. Note that $\mathbf{TFHE.PBS}$ will internally use ciphertexts of type \mathbf{LWE} if the input ciphertexts are of type $\mathbf{F-GLWE}$, and vice-versa.

Using linear functions and PBS operations we can then evaluate arbitrary functions using the $\mathbf{TFHE.Eval}$ operation given in Figure 8. The function F in this figure corresponds to a function on m arguments, producing n outputs. In applying $\mathbf{TFHE.Eval}$ one needs to be careful in ensuring that the function decomposition and encoding of inputs respects the fact that the linear functions need to be admissible. As long as this is done, the algorithm $\mathbf{TFHE.Eval}$ enables evaluations of arbitrary functions.

TFHE Evaluation

$\mathbf{TFHE.Eval}((\mathbf{ct}_0, \dots, \mathbf{ct}_{m-1}), F, \mathbf{pk})$:

1. Decompose F into a sequence of admissible linear functions L_i and look-up-table evaluations T_j .
2. For each L_i gate, evaluate the gate using the operations $\mathbf{TFHE.Add}$ and $\mathbf{TFHE.ScalarMult}$.
3. For each T_j gate, evaluate the gate using the operation $\mathbf{TFHE.PBS}$.
4. Output the ciphertexts corresponding to the output gates of the decomposition of F .

Figure 8: TFHE Evaluation for an Arbitrary Function

6.2 Assumptions on Basic TFHE

The TFHE system relies on multiple LWE assumptions, as we have multiple LWE dimensions involved, i.e. $\hat{\ell}$, ℓ and $w \cdot N$. Assuming appropriate circular security assumptions on these LWE instances one can prove the following theorem.

Theorem 2. *When instantiated with appropriate parameters the TFHE algorithm given by $(\mathbf{TFHE.Gen}, \mathbf{TFHE.KeyGen}, \mathbf{TFHE.Enc}, \mathbf{TFHE.Dec}, \mathbf{TFHE.Eval})$ is an IND-CPA secure fully homomorphic encryption scheme under various appropriate LWE assumptions.*

When setting parameters for TFHE we also make assumptions, not only related to security but also to correctness. In particular we provide a weak correctness definition tailored to FHE schemes based on (G)LWE. The key differences to Definition 3 are that

1. The evaluation oracle ignores the ciphertexts in the state and generates fresh encryptions of the messages, thus moving closer to a non-reactive version of correctness;
2. To generate these fresh ciphertexts the evaluation oracle uses a secret key version of the encryption algorithm, which implies that the masks of the ciphertexts are uniformly random.

Definition 5 (Average-Case Correctness of LWE-based FHE Schemes). For a public-key LWE-based FHE scheme $(\mathbf{FHE.KeyGen}, \mathbf{FHE.Enc}, \mathbf{FHE.Dec}, \mathbf{FHE.Eval})$ let σ_e , σ_a and σ_h be the standard deviation of the noise distribution of ciphertexts output by

- $\mathbf{FHE.Eval}$ under uniformly random input masks,
- $\mathbf{FHE.Enc}$ under adversarially chosen encryption randomness,
- $\mathbf{FHE.Enc}$ under honestly chosen encryption randomness,

respectively. Then the scheme is said to be *average-case correct* if for all PPT adversary \mathcal{A} it holds that

$$\Pr \left[\text{AvgCor}_{\sigma_{\text{Eval}}, \sigma_{\text{Enc}}}^{\mathcal{A}}(\text{sec}) = 1 \right]$$

is negligible in sec , where $\text{AvgCor}^{\mathcal{A}}(\text{sec})$ is the game in Figure 10 and $\sigma_{\text{Eval}} = \sqrt{\sigma_e^2 + \sigma_h^2}$ and $\sigma_{\text{Enc}} = \sqrt{\sigma_a^2 + \sigma_h^2}$.

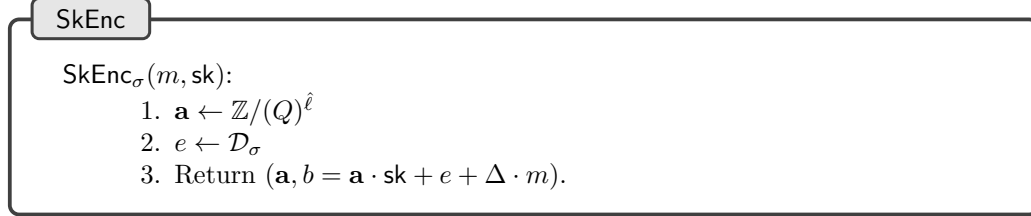


Figure 9: LWE-based secret key encryption algorithm used in the AvgCor game. Here, \mathcal{D}_{σ} refers to the discrete Gaussian with standard deviation σ .

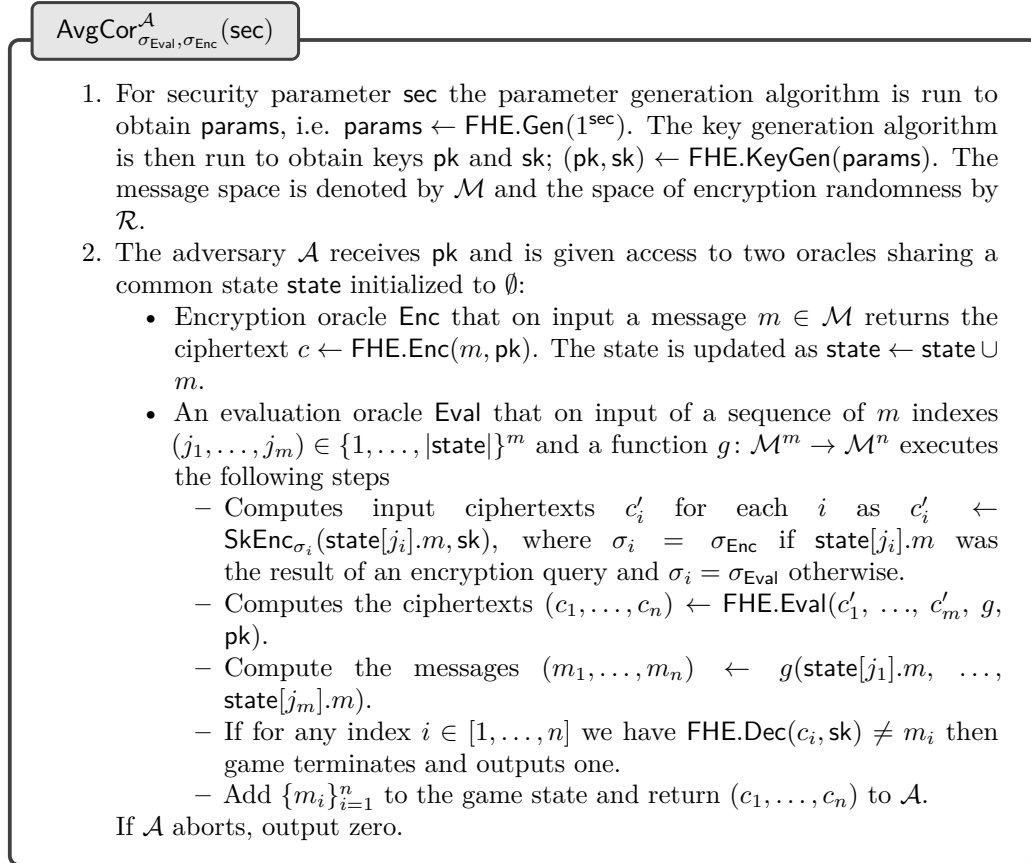


Figure 10: The security game for Average-Case correctness assumption.

Definition 5 reflects the assumptions made in practice when selecting the parameters for TFHE, see for example [Tap23], thus we have the Lemma:

Lemma 4. *In practice TFHE is average-case correct.*

While Definition 5 is tailored specifically towards TFHE, similar assumptions are typically made when selecting parameters for other exact FHE schemes. Thus, we believe that our approach can be adapted to other schemes as well.

6.3 Extensions to TFHE

To achieve the stronger notion of sIND-CPA^D security, see Definition 2/Figure 2, we need to define a “randomized” evaluation procedure. We first define a re-randomization procedure as in Figure 11, which takes as inputs a set of ciphertexts of type `type`. This algorithm simply executes the basic encryption procedure `TFHE.Enc-Sub` to obtain encryptions of zero, then it applies key-switching, via `TFHE.DimensionSwitch` to obtain ciphertexts of type `type`, and then it adds the result to the ciphertexts needing re-randomization.

TFHE Re-Randomization

`TFHE.ReRand((ct0, ..., ctm-1), (seed0, ..., seedm-1), pk):`

1. For $i \in [0, \dots, m-1]$ do
 - (a) $\mathfrak{z}t_i \leftarrow \text{TFHE.Enc}(0, \text{pk}; \text{seed}_i)$.
 - (b) $\text{ct}'_i \leftarrow \text{TFHE.Add}(\text{ct}_i, \mathfrak{z}t_i)$.
2. Return $(\text{ct}'_0, \dots, \text{ct}'_{m-1})$.

Figure 11: TFHE Re-Randomization

From this we can define a pre-processing procedure which should be executed before every application of the `TFHE.Eval` operation, see Figure 12. Note, the input to `Hash2-sec` contains **all** the input ciphertexts ct_i , all the input auxillary data aux_i , the argument position i , and the function description F . In practice the function description F can be any unique identifier for the function, for example a hash of the function description, or simply just a name. Finally, note that the m applications of `Hash2-sec` can be implemented in an optimized manner, in which the first part of the input, i.e. $(\text{ct}_0, \dots, \text{ct}_{m-1}, \text{aux}_0, \dots, \text{aux}_{m-1}, F)$ can be entered into the hash state once. Then the state can be replicated m times, and the final input of i can then be applied. Thus the cost is roughly one execution of the hash function on the full data string, and not m executions.

TFHE Derandomized Pre-Processing for Eval

`TFHE.PreProcEval(((ct0, aux0), ..., (ctm-1, auxm-1)), F, pk):`

1. For $i \in [0, \dots, m-1]$ do
 - (a) $\text{seed}_i \leftarrow \text{Hash}^{2\text{-sec}}(\text{ct}_0, \dots, \text{ct}_{m-1}, \text{aux}_0, \dots, \text{aux}_{m-1}, F, i)$.
2. $(\text{rt}_0, \dots, \text{rt}_{m-1}) \leftarrow \text{TFHE.ReRand}((\text{ct}_0, \dots, \text{ct}_{m-1}), (\text{seed}_0, \dots, \text{seed}_{m-1}), \text{pk})$.
3. Return $(\text{rt}_0, \dots, \text{rt}_{m-1})$.

Figure 12: TFHE Derandomized Pre-Processing for Eval

Finally, we can present the randomized evaluation procedure in Figure 13. We assume here that F is a function mapping m ciphertext inputs to n ciphertext outputs.

So the only thing left is to define what aux_i^π is. We only require, see Section 7, that for ciphertexts ct_i which are “fresh”, i.e. inputs to an evaluation which are not outputs of an evaluation, that aux_i^π “contains” a ZKPoK π associated with the ciphertext. Here “contains” could be via a hash of the ZKPoK. As explained earlier during encryption a ZKPoK can be attached to the ciphertext to prove that the encryptor knows the underlying message, and that they have encrypted the ciphertext correctly; such ZKPoKs are given in [Lib24]. The need to include the ZKPoK in the auxillary data for fresh ciphertexts is a “proof artifact” in our proof of sIND-CPA^D security for TFHE when using evaluation

TFHE Derandomized Eval

TFHE.DeRandEval($((\mathbf{ct}_0, \text{aux}_0^\pi), \dots, (\mathbf{ct}_{m-1}, \text{aux}_{m-1}^\pi)), F, \text{pk})$):

1. $(\mathbf{rt}_0, \dots, \mathbf{rt}_{m-1}) \leftarrow \text{TFHE.PreProcEval}(((\mathbf{ct}_1, \text{aux}_1^\pi), \dots, (\mathbf{ct}_{m-1}, \text{aux}_{m-1}^\pi)), F, \text{pk})$.
2. $(\mathbf{ct}'_0, \dots, \mathbf{ct}'_{n-1}) \leftarrow \text{TFHE.Eval}((\mathbf{rt}_0, \dots, \mathbf{rt}_{m-1}), F, \text{pk})$.
3. Return $((\mathbf{ct}'_0, \perp), \dots, (\mathbf{ct}'_{n-1}, \perp))$.

Figure 13: TFHE Derandomized Eval

procedure TFHE.DeRandEval from Figure 13. For non-fresh ciphertexts, i.e. outputs of the TFHE.Eval operation, we set aux^π to be \perp , as in line 3 of Figure 13.

7 sIND-CPA^D Security for TFHE

Our main TFHE cryptosystem achieves sIND-CPA^D security via a de-randomized evaluation procedure, in the random oracle model. Namely, the use of the TFHE.DeRandEval evaluation methodology. The rest of this section is devoted to showing that, with appropriate choice of parameters, our specific TFHE scheme using the evaluation procedure TFHE.DeRandEval is indeed sIND-CPA^D secure. The main technical difficulty is to show that the scheme achieves computational ACER correctness.

7.1 Assumptions

To prove sIND-CPA^D for our variant of TFHE which uses the TFHE.DeRandEval evaluation methodology we need to make a sequence of additional computational and statistical assumptions. We first elaborate on these here, before moving onto proving our scheme achieves sIND-CPA^D security.

Recall, the inputs to the evaluation procedure TFHE.Eval are LWE ciphertexts $\mathbf{ct}_i = (\mathbf{a}_i, b_i)$ where \mathbf{a}_i is a vector over $\mathbb{Z}/(Q)$ of dimension ℓ or $w \cdot N$ (depending on the value of type). In this section (to unify the treatment) we shall refer to the underlying secret key of dimension ℓ or $w \cdot N$ as simply \mathbf{s} .² With this convension we have $b_i = \mathbf{a}_i \cdot \mathbf{s} + e + \Delta \cdot m_i$. We will refer to \mathbf{a}_i as the mask term and e as the noise term.

Assumption 1. *We assume that the noise components of the ciphertexts in the ACERCor game (applied to TFHE) are computationally indistinguishable from the noise components in the AvgCor game.*

We believe that Assumption 1 is reasonable, even though the noise components are clearly statistically distinguishable. However, they are part of the body of the ciphertexts and thus computationally hidden. Since the noise of the outputs of the Enc and Eval operations are complex combinations of the noise components in the public key and evaluation key, resp., which are computationally hidden from the adversary, it has very little control over the noise to bias it. Note that Assumption 1 essentially states that we do not need to address the noise distribution of the ciphertexts, so the main focus will be on the masks, which are public and thus visible to the adversary.

Recall the algorithm TFHE.DimensionSwitch(\mathbf{ct}, PKSK) used in the TFHE.Enc($m, \text{pk}, \text{seed}; P, Q, \text{type}, \hat{\ell}$) algorithm. This maps an LWE instance $\mathbf{ct} = (\mathbf{a}, v)$ of dimension $\hat{\ell}$ to an LWE instance $\mathbf{ct}' = (\mathbf{a}', v')$ of dimension $w \cdot N$ (if type = F-GLWE) or

²Earlier \mathbf{s} was just the secret key of dimension ℓ , with $((\mathbf{s}_i[j])_{i=0}^{w-1})_{j=0}^{N-1}$ being the key of dimension $w \cdot N$.

one of dimension ℓ (if $\text{type} = \text{LWE}$). Our next assumption relates the statistical properties of the input vector \mathbf{a} and the output vector \mathbf{a}' .

Assumption 2. *If the input vector \mathbf{a} to $\text{TFHE.DimensionSwitch}$ is uniformly random modulo q , then the output \mathbf{a}' is indistinguishable from a uniformly random vector modulo q .*

When $\text{type} = \text{LWE}$ and hence (in most cases) $\ell < \hat{\ell}$ we have that this assumption plausibly holds (even statistically) although we have no proof of this, since the operation of $\text{TFHE.DimensionSwitch}(\text{ct}, \text{PKSK})$ is compressing. However, when $\text{type} = \text{F-GLWE}$ the operation is expanding, since $\hat{\ell} < w \cdot N$, and so the assumption only can hold computationally.

Our next assumption is related to a variant of the LWE problem, which we shall call the HintLWE problem.

Definition 6. Let N be a power of 2, Q a ciphertext modulus, and \mathcal{X}_1 and \mathcal{X}_2 be two bounded distributions over $\mathbb{Z}/(Q)$. The HintLWE problem, as considered in this work, is to distinguish the two distributions in Figure 14.

- | | |
|--|--|
| <ul style="list-style-type: none"> • \mathcal{D}_1 <ul style="list-style-type: none"> – $\text{pk}_a \leftarrow \mathbb{Z}/(Q)^{\hat{\ell}}$ – $\mathbf{r} \leftarrow \{0, 1\}^{\hat{\ell}}$ and $\hat{\mathbf{s}} \leftarrow \{0, 1\}^{\hat{\ell}}$ – $\mathbf{e}_1 \leftarrow \mathcal{X}_1^{\hat{\ell}}, \mathbf{e} \leftarrow \mathcal{X}_1^{\hat{\ell}}, e_2 \leftarrow \mathcal{X}_2$. – $\mathbf{a} = \text{pk}_a \odot \overset{\leftrightarrow}{\mathbf{r}} + \mathbf{e}_1$ – $d = (-\hat{\mathbf{s}}, \mathbf{e}) \cdot (\mathbf{e}_1, \mathbf{r}) + e_2$ – Output $(\text{pk}_a, \mathbf{a}, \hat{\mathbf{s}}, \mathbf{e}, d)$. | <ul style="list-style-type: none"> • \mathcal{D}_2 <ul style="list-style-type: none"> – $\text{pk}_a \leftarrow \mathbb{Z}/(Q)^{\hat{\ell}}$ – $\mathbf{r} \leftarrow \{0, 1\}^{\hat{\ell}}$ and $\hat{\mathbf{s}} \leftarrow \{0, 1\}^{\hat{\ell}}$ – $\mathbf{e}_1 \leftarrow \mathcal{X}_1^{\hat{\ell}}, \mathbf{e} \leftarrow \mathcal{X}_1^{\hat{\ell}}, e_2 \leftarrow \mathcal{X}_2$. – $\mathbf{a} \leftarrow \mathbb{Z}/(Q)^{\hat{\ell}}$ – $d = (-\hat{\mathbf{s}}, \mathbf{e}) \cdot (\mathbf{e}_1, \mathbf{r}) + e_2$ – Output $(\text{pk}_a, \mathbf{a}, \hat{\mathbf{s}}, \mathbf{e}, d)$. |
|--|--|

Figure 14: The two distributions for our HintLWE problem.

Note that in contrast to regular LWE, in (our version of) HintLWE the distinguisher gets the noisy inner product of the secret vector $(\mathbf{e}_1, \mathbf{r})$ with a public random vector $(-\hat{\mathbf{s}}, \mathbf{e})$. We assume that this problem is essentially as hard as LWE. This problem can be viewed as a variant of Hint-MLWE as defined in [KLSS23] and Theorem 1 in [KLSS23] gives some evidence of its hardness, although the noise distributions differ.

Consider the first distribution \mathcal{D}_1 in Figure 14 and the public key of our TFHE scheme $(\text{pk}_a, \text{pk}_b)$ where $\text{pk}_a \in \mathbb{Z}/(Q)^{\hat{\ell}}$ and

$$\text{pk}_b = \text{pk}_a \odot \overset{\leftrightarrow}{\hat{\mathbf{s}}} + \mathbf{e}$$

where $\hat{\mathbf{s}} \in \{0, 1\}^{\hat{\ell}}$ and $\mathbf{e} \leftarrow \text{TUniform}(\hat{\ell}, b_{\hat{\ell}})$; recall $\overset{\leftrightarrow}{\hat{\mathbf{s}}}$ is the reverse ordering of the vector $\hat{\mathbf{s}}$. If the adversary knows the secret key $\hat{\mathbf{s}}$, then the adversary knows the randomness within the public key \mathbf{e} . Let ciphertext $\text{ct} = (\mathbf{a}, b)$ be a ciphertext encrypting a message m which has been output by a valid call to the encryption algorithm TFHE.Enc-Sub , i.e. $\mathbf{a} = \text{pk}_a \odot \overset{\leftrightarrow}{\mathbf{r}} + \mathbf{e}_1$ and $b = \text{pk}_b \cdot \mathbf{r} + e_2 + \Delta \cdot m$, for $\mathbf{r} \leftarrow \{0, 1\}^{\hat{\ell}}, \mathbf{e}_1 \leftarrow \mathcal{D}^{\hat{\ell}}$, and $e_2 \leftarrow \mathcal{D}^1$. In such a situation the adversary also knows

$$\begin{aligned}
 b - \mathbf{a} \cdot \hat{\mathbf{s}} - \Delta \cdot m &= (\text{pk}_b \cdot \mathbf{r} + e_2 + \Delta \cdot m) - \mathbf{a} \cdot \hat{\mathbf{s}} - \Delta \cdot m, \\
 &= \left((\text{pk}_a \odot \overset{\leftrightarrow}{\hat{\mathbf{s}}} + \mathbf{e}) \cdot \mathbf{r} + e_2 \right) - (\text{pk}_a \odot \overset{\leftrightarrow}{\mathbf{r}} + \mathbf{e}_1) \cdot \hat{\mathbf{s}}, \\
 &= \left(\mathbf{e} \odot \overset{\leftrightarrow}{\mathbf{r}} - \mathbf{e}_1 \odot \overset{\leftrightarrow}{\hat{\mathbf{s}}} \right)_{\hat{\ell}-1} + e_2, \\
 &= (-\hat{\mathbf{s}}, \mathbf{e}) \cdot (\mathbf{e}_1, \mathbf{r}) + e_2,
 \end{aligned} \tag{2}$$

$$= d.$$

Thus the first distribution in Figure 14 (when $\mathcal{X}_1 = \mathcal{D}^{\hat{\ell}}$ and $\mathcal{X}_2 = \mathcal{D}^1$) is what is seen by the adversary, knowing \hat{s} , if the encryption is carried out according to the algorithm TFHE.Enc-Sub . However, the second distribution in Figure 14 is the same except that the component \mathbf{a} in the ciphertext is generated uniformly at random and not via the encryption algorithm.

Assumption 3. *The problem HintLWE in Definition 6 is computationally hard assuming $\hat{\ell}$, \mathcal{X}_1 and \mathcal{X}_2 are chosen as in our scheme.*

The final assumption we shall need is to ensure that our reduction in Theorem 4 below can program the random oracle and simulate the ACERCor game correctly. This assumption is related to the unpredictability of the output ciphertexts from an evaluation operation related to a *specific* re-randomization procedure for TFHE.

We first formally define unpredictability as follows:

Definition 7. Let $(\text{FHE.KeyGen}, \text{FHE.Enc}, \text{FHE.Dec}, \text{FHE.Eval})$ be an FHE scheme with ciphertext space \mathcal{C} . We say that the scheme is *unpredictable* if there exists an inverter \mathcal{I} such that for any 2-stage PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and for any $(c, \text{state}) \leftarrow \mathcal{A}_1(\text{pk})$, where $(\text{pk}, \text{sk}) \leftarrow \text{FHE.KeyGen}(\text{params})$ and $\text{params} \leftarrow \text{FHE.Gen}(1^{\text{sec}})$, we have that either

$$\Pr \left[\exists i \in [1, \dots, n] \text{ s.t. } c = c'_i \mid (c_1, \dots, c_m, g) \leftarrow \mathcal{A}_2(\text{state}) \text{ for } g: \mathcal{M}^m \rightarrow \mathcal{M}^n, \right. \\ \left. (c'_1, \dots, c'_n) \leftarrow \text{FHE.Eval}((c_1, \dots, c_m), g, \text{pk}) \right]$$

or

$$\Pr \left[\nexists i \in [1, \dots, n] \text{ s.t. } c = c'_i \mid (m_1, \dots, m_m, r_1, \dots, r_m, g) \leftarrow \mathcal{I}(\text{pk}, c) \text{ for } g: \mathcal{M}^m \rightarrow \mathcal{M}^n, \right. \\ \left. c_i \leftarrow \text{FHE.Enc}(m_i, \text{pk}; r_i) \text{ for } i \in [1, \dots, m], \right. \\ \left. (c'_1, \dots, c'_n) \leftarrow \text{FHE.Eval}((c_1, \dots, c_m), g, \text{pk}) \right]$$

is negligible in sec .

Intuitively, Definition 7 states that for any ciphertext that an adversary produces, the evaluation is either randomized enough that it is unlikely that the adversary can produce a set of inputs that yields this specific ciphertext, or it is generally easy to find a pre-image of that ciphertext. Note the lack of state in the input to \mathcal{I} , which could contain arbitrary secret information passed from \mathcal{A}_1 to \mathcal{A}_2 .

Our reason for including the second case in Definition 7 to enable evaluation to return trivial ciphertexts “encrypting” constant values if the input function is trivial and independent of the inputs. This can be more efficient than, for example, returning random encryptions of such values.

The specific randomized variant of TFHE we will combine with our unpredictability definition is the following one: In the evaluation procedure the evaluator first generates uniformly random seeds for each ciphertext, and then they apply the algorithm TFHE.ReRand to the input ciphertexts to TFHE.Eval . This provides the specific randomized evaluation procedure given in Figure 15. The only difference between this evaluation procedure and our final de-randomized one, i.e. TFHE.DeRandEval , is that in TFHE.DeRandEval the seed_i values are created via calls to a random oracle.

In Figure 15 we have an extra technical condition in that the evaluation algorithm not only returns the output evaluation, but it also returns the seeds which have been used to produce the re-randomization as auxillary information aux^{seed} attached to each ciphertext. Note that this is different from the auxillary information aux^π that contains the ZKPoK

TFHE Fully Randomized Eval

TFHE.Eval^R((ct₁, aux₁^{seed}), ..., (ct_m, aux_m^{seed}), F, pk):

1. For $i \in [1, \dots, m]$ do seed_i $\leftarrow \{0, 1\}^{2 \cdot \text{sec}}$.
2. (rt₁, ..., rt_m) \leftarrow TFHE.ReRand((ct₁, ..., ct_m), (seed₁, ..., seed_m), pk).
3. (ct'₁, ..., ct'_n) \leftarrow TFHE.Eval((rt₁, ..., rt_m), F, pk).
4. aux^{seed} \leftarrow (seed₁, ..., seed_m).
5. Return ((ct'₁, aux^{seed}), ..., (ct'_n, aux^{seed})).

Figure 15: TFHE Fully Randomized Eval

discussed in Section 6.3, which will become relevant in Section 7.3. Since the auxiliary information is attached to ciphertexts, we also pass this into the TFHE.Eval^R function (but then ignore it). Outputting such information obviously can only make security harder to obtain. Thus if we prove security with this information leakage then we will have security of an implemented scheme where no such information is leaked. This leakage is technically needed later in our final proof technique for the TFHE.DeRandEval evaluation method in the random oracle model.

Combining our unpredictability definition and the above randomized evaluation algorithm leads to our final assumption.

Assumption 4. *The TFHE algorithm with evaluation algorithm TFHE.Eval^R from Figure 15 is unpredictable.*

We believe this is a reasonable assumption. In fact, for a limited set of circuits we can even prove that Assumption 4 is implied by IND-CPA security and ACER correctness:

For simplicity, assume the unpredictability adversary ($\mathcal{A}_1, \mathcal{A}_2$) outputs a tuple (c, c_1, g) , such that $c = \text{TFHE.Eval}^R(c_1, g, \text{pk})$ with noticeable probability, where g is the identity function. An IND-CPA adversary \mathcal{B} against TFHE, given (c, c_1, g) may choose $(m_0, m_1) = (0, 1)$ and receive $c_b \leftarrow \text{TFHE.Enc}(m_b, \text{pk})$. \mathcal{B} runs $c' \leftarrow \text{TFHE.Eval}(c_1 + c_b, g)$. If $c' = c$, output $b' = 0$, otherwise $b' = 1$. The key observation here is that if $b = 0$, \mathcal{B} runs $\text{TFHE.Eval}^R(c_1, g, \text{pk})$ and so with noticeable probability $c' = c$. Otherwise, if $b = 1$, \mathcal{B} runs $\text{TFHE.Eval}^R(c_1 + 1, g, \text{pk})$ and by correctness it follows that $c' \neq c$ except with negligible probability.

We now explain the key difficulty in generalizing this proof sketch to all circuits. Let $(c, (c_1, \dots, c_m), g) \leftarrow (\mathcal{A}_1, \mathcal{A}_2)$ and denote $m_i \leftarrow \text{TFHE.Dec}(c_i, \text{sk})$. In order to ensure that $c' \neq c$ in case $b = 1$, the reduction needs to find m'_i such that $g(m_1, \dots, m_i, \dots, m_m) \neq g(m_1, \dots, m'_i, \dots, m_m)$ without knowing m_i (since it does not know sk). Depending on the circuit g , this might be inefficient or even impossible (if g is the constant circuit).

Still, we believe Assumption 4 is reasonable since the masks of the outputs of TFHE.Eval depend on the input masks in highly non-trivial ways. For the assumption to not hold, there would need to be a circuit along with a noticeable fraction of input ciphertexts that yield the same output ciphertext. Note though that this fraction cannot be all (but a negligible fraction of) ciphertexts, since otherwise there would be an efficient inverter. The structure of TFHE.Eval does not suggest that there is such a circuit.

For readers who wish to avoid Assumption 4, a natural way to obtain an unpredictable scheme is to randomize the output of the evaluation before returning it. For proper re-randomization, this would mean that the first case of Definition 7 is always satisfied. But for efficiency reasons, one may assume that randomizing the input is sufficient to yield an unpredictable output, as we do here, which seems like a plausible assumption.

7.2 A Specific re-rand Procedure for TFHE

In Section 5 we showed that a computationally ACER correct and IND-CPA secure scheme is sIND-CPA^D secure. In this sub-section, as a path on our way to showing a de-randomized variant of TFHE which is sIND-CPA^D secure in the random oracle model, we show that the specific variant of re-rand, for the TFHE scheme, given by TFHE.Eval^R , creates a scheme which is computationally ACER correct.

Theorem 3. *The TFHE algorithm with evaluation algorithm $\text{TFHE.Eval}^R((\text{ct}_1, \text{aux}_1^{\text{seed}}), \dots, (\text{ct}_m, \text{aux}_m^{\text{seed}})), F, \text{pk})$ from Figure 15 is computationally ACER correct if*

- *Assumption 1, re indistinguishability of noise components holds.*
- *Assumption 2, re $\text{TFHE.DimensionSwitch}$, holds.*
- *Assumption 3, re the HintLWE problem, holds.*

Proof. Let \mathcal{A} be an adversary that breaks the computational ACER correctness of TFHE when used with the evaluation algorithm $\text{TFHE.Eval}^R((\text{ct}_1, \dots, \text{ct}_m), F, \text{pk})$. We focus on the *randomizers* $\text{TFHE.Enc}(0, \text{pk}, \text{seed})$ which are computed in the execution of $\text{TFHE.ReRand}((\text{ct}_1, \dots, \text{ct}_m), (\text{seed}_1, \dots, \text{seed}_m), \text{pk})$. Recall, these are computed as values $(\mathbf{a}, b) \in \mathbb{Z}/(Q)^{\ell+1}$, from the public key $(\text{pk}_a, \text{pk}_b)$ and the seed values seed_i , by expanding seed_i to obtain the randomizers $(\mathbf{r}, \mathbf{e}_1, e_2)$. The (\mathbf{a}, b) value is then converted to dimension $\ell + 1$ (resp. $w \cdot N + 1$) via a call to $\text{TFHE.DimensionSwitch}$, to produce the output (\mathbf{a}', b') of $\text{TFHE.Enc}(0, \text{pk}, \text{seed})$.

We first note that we can re-write the computation of b in the following way, with the notation as in equation (2) above,

$$b \leftarrow \mathbf{a} \cdot \hat{\mathbf{s}} - \hat{\mathbf{s}} \cdot \mathbf{e}_1 + \mathbf{e} \cdot \mathbf{r} + e_2, \quad (3)$$

where $(\mathbf{r}, \mathbf{e}_1, e_2)$ is the output obtained on input of the relevant seed_i value. Assuming we know $\hat{\mathbf{s}}$, this is just a syntactical change. Note that the challenger in the correctness game knows the secret key so in the following we assume that b is computed in that way when computing $\text{TFHE.Enc}(0, \text{pk}, \text{seed})$.

Assume that \mathcal{A} makes exactly t queries to the evaluation oracle. Note that only the t -th query may yield an incorrect decryption, since otherwise the game would have terminated before the t -th query. Let the t -th query to the evaluation oracle be $((\text{ct}_1, \text{aux}_1), \dots, (\text{ct}_m, \text{aux}_m), g)$.

We define hybrids $\{\mathcal{H}_i\}_{i=0}^m$. In hybrid \mathcal{H}_i the evaluation oracle will compute the randomizer (\mathbf{a}', b') of the first i ciphertexts as follows:

- $\mathbf{a} \leftarrow \text{pk}_a \odot \mathbf{r} + \mathbf{e}_1$ (for appropriately chosen \mathbf{r} and \mathbf{e}_1).
- Compute b as in equation (3), for appropriately chosen e_2 .
- Compute (\mathbf{a}', b') by applying $\text{TFHE.DimensionSwitch}$ to (\mathbf{a}, b) .

For the $m - i$ last ciphertexts, the game in \mathcal{H}_i will choose \mathbf{a} uniformly at random, and then compute b and (\mathbf{a}', b') in the same way.

Note that in the t -th query in \mathcal{H}_0 , all masks of randomizers are uniformly random, assuming Assumption 2. On the other hand \mathcal{H}_m corresponds exactly to the ACER correctness game.

The theorem will follow from the fact that in \mathcal{H}_0 , the uniformly random masks in the randomizers statistically re-randomize the masks of the ciphertexts, and thus the inputs to TFHE.Eval are ciphertexts with uniformly random masks. A correctness failure in this query thus contradicts Lemma 4 in that TFHE.Eval is not correct assuming uniformly random masks and Gaussian noise components, or Assumption 1, i.e. the computationally hidden noise components in TFHE are distinguishable from Gaussians. It remains to show that \mathcal{H}_0 and \mathcal{H}_m are indistinguishable. We do so by showing that \mathcal{H}_i and \mathcal{H}_{i+1} are indistinguishable, under the HintLWE assumption, by constructing an adversary \mathcal{B} against

the HintLWE problem from a distinguisher. Let $(\mathbf{pk}_a, \mathbf{a}, \hat{\mathbf{s}}, \mathbf{e}, d)$ be the input to \mathcal{B} , where \mathbf{a} is either an LWE sample or uniformly chosen at random. The algorithm \mathcal{B} will set the public key to $\mathbf{pk} = (\mathbf{pk}_a, \mathbf{pk}_a \odot \hat{\mathbf{s}} + \mathbf{e})$. Note that \mathbf{pk}_a , $\hat{\mathbf{s}}$ and \mathbf{e} are from \mathcal{B} 's input.

Algorithm \mathcal{B} then simulates the challenger of the ACERCor game, up to the t -th evaluation query, by simply following the specification. In the t -th evaluation query, the first i and the last $m - i - 1$ randomizers are computed in the obvious way. For the i -th randomizer, \mathcal{B} will use the value \mathbf{a} from its input and set

$$b = \mathbf{a} \cdot \hat{\mathbf{s}} + d.$$

It will then apply `TFHE.DimensionSwitch` to (\mathbf{a}, b) to obtain (\mathbf{a}', b') as the randomizer.

Note that when \mathbf{a} is an LWE sample (i.e. distribution \mathcal{D}_1 of the HintLWE problem), algorithm \mathcal{B} simulates \mathcal{H}_{i+1} , but if \mathbf{a} is uniform, then \mathcal{B} simulates \mathcal{H}_i . The only caveat is that for the LWE sample (\mathbf{a}, b) the reduction \mathcal{B} does not know the encryption randomness $(\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2)$, nor the seed value `seed` which would produce such values during execution of `TFHE.Enc`. Thus \mathcal{B} cannot build the full output; which, according to the construction in Figure 15, should contain `seedi`. But since this t -th query is the last query, the game (in Figure 4) terminates before returning the ciphertext. Hence, \mathcal{B} does not need to return the ciphertext and it will still faithfully simulate one of the two hybrids. This shows that a distinguisher between \mathcal{H}_i and \mathcal{H}_{i+1} can be used to break HintLWE. It follows that \mathcal{H}_0 and \mathcal{H}_m are indistinguishable, which concludes the proof. \square

7.3 Proving TFHE.DeRandEval Provides sIND-CPA^D Security in the Random Oracle Model

We shall now show that in `TFHE.EvalR` from Figure 15 we can replace the generation of the random seeds with the evaluation of a hash function. Thus we obtain exactly the evaluation operation of `TFHE.DeRandEval`.

For technical reasons, we need to append ZKPoKs to fresh ciphertexts. Usually, such proofs are attached to ciphertexts to protect against an adversary passing in invalid ciphertexts. In particular they are used to protect against an adversary entering a ciphertext which produces a selective failure attack on correctness, see [Sma23]. Our usage is also related to correctness, but within our computational ACER game.

For our encryption scheme `TFHE.Enc` we define a ZKPoK Π as a triple of algorithms $(\text{Gen}, \text{Prove}, \text{Verify})$, such that $(\text{pp}, \tau) \leftarrow \Pi.\text{Gen}(\text{sec})$, $\pi \leftarrow \text{Prove}_{\text{pp}}(\text{pk}, \text{ct}, m, \text{seed})$ and $b \leftarrow \text{Verify}_{\text{pp}}(\text{pk}, \text{ct}, \pi) \in \{0, 1\}$. We require the following three properties of the zero knowledge proofs, namely completeness, knowledge-soundness and straightline extractability:

- A ZKPoK Π is *complete*, if for any adversary \mathcal{A} the probability

$$\Pr[\Pi.\text{Verify}_{\text{pp}}(\text{pk}, \text{ct}, \pi) = 0 \wedge \text{TFHE.Enc}(m, \text{pk}, \text{seed}) = \text{ct}]$$

is negligible in `sec`, where

- $\text{params} \leftarrow \text{FHE.Gen}(1^{\text{sec}})$.
- $(\text{pk}, \text{sk}) \leftarrow \text{TFHE.KeyGen}(\text{params})$.
- $(\text{pp}, \tau) \leftarrow \Pi.\text{Gen}(\text{sec})$.
- $(\text{ct}, m, \text{seed}) \leftarrow \mathcal{A}(\text{pp}, \text{pk})$.
- $\pi \leftarrow \Pi.\text{Prove}(\text{pk}, \text{ct}, m, \text{seed})$.
- A ZKPoK Π is *knowledge-sound*, if for any PPT adversary \mathcal{A} there exists a PPT extractor $\mathcal{E}_{\mathcal{A}}$ that has access to \mathcal{A} 's internal state and random coins ρ such that

$$\Pr[\Pi.\text{Verify}_{\text{pp}}(\text{pk}, \text{ct}, \pi) = 1 \wedge \text{TFHE.Enc}(m, \text{pk}, \text{seed}) \neq \text{ct}]$$

is negligible in `sec`, where

- $\text{params} \leftarrow \text{FHE.Gen}(1^{\text{sec}})$.
- $(\text{pk}, \text{sk}) \leftarrow \text{TFHE.KeyGen}(\text{params})$.
- $(\text{pp}, \tau) \leftarrow \Pi.\text{Gen}(\text{sec})$
- $(\text{ct}, \pi) \leftarrow \mathcal{A}(\text{pp}, \text{pk})$
- $(m, \text{seed}) \leftarrow \mathcal{E}_{\mathcal{A}}(\text{pp}, \text{pk}, \text{ct}, \pi, \rho)$.
- A knowledge-sound ZKPoK Π is *straightline extractable*, if the extractor $\mathcal{E}_{\mathcal{A}}$ does not rewind the adversary \mathcal{A} .

In many protocols one requires a stronger form of soundness called simulation-extractability. For the purposes of this section this is not required.

We can now show that the de-randomized variant of TFHE is computationally ACER correct.

Theorem 4. *The TFHE fully homomorphic encryption scheme, when used with the evaluation algorithm TFHE.DeRandEval , is computationally ACER correct assuming*

- *Assumptions 1, 2, 3 and 4 hold.*
- *The ZKPoKs used are complete, knowledge-sound, and straightline extractable.*
- *The hash function used inside TFHE.PreProcEval is modeled as a random oracle.*

Proof. Since we assume that the Assumptions 1, 2, 3 hold, Theorem 3 implies that the TFHE algorithm with evaluation algorithm TFHE.Eval^R from Figure 15 is computationally ACER correct. Since we also assume that Assumption 4 holds, we also know that the same algorithm with evaluation algorithm TFHE.Eval^R is also unpredictable.

Let \mathcal{A} be an adversary that is able to break the ACER correctness of the TFHE scheme with evaluation algorithm TFHE.DeRandEval in the random oracle model. We will show that \mathcal{A} can be used to create an adversary \mathcal{B} which breaks the ACER correctness of the TFHE algorithm with evaluation algorithm TFHE.Eval^R .

The algorithm \mathcal{B} runs \mathcal{A} and responds to its queries in the following way:

- Encryption queries (m, seed) are forwarded keeping track of the correct indices. When the ACERCor challenger returns the ciphertext ct , \mathcal{B} computes $\pi \leftarrow \Pi.\text{Prove}_{\text{pp}}(\text{pk}, \text{ct}, m, \text{seed})$, records (ct, π) in its internal game state state' and returns it to \mathcal{A} .
- For each evaluation query, we first check if this query has been made before. This includes queries as part of random oracle queries (see below). If so, we simply map the indices of the output ciphertexts in the simulated state to the ciphertexts in the state that corresponds to the previous invocation. Otherwise, evaluation queries (j_1, \dots, j_m, g) are forwarded, potentially re-mapping the indices. Upon receiving the resulting ciphertexts $(\text{ct}_1, \dots, \text{ct}_m)$ and seed_i such that

$$(\text{ct}_1, \dots, \text{ct}_m) \leftarrow \text{TFHE.Eval}((\text{rt}_1, \dots, \text{rt}_m), g, \text{pk}),$$

where

$$(\text{rt}_1, \dots, \text{rt}_m) \leftarrow \text{TFHE.ReRand}((\text{ct}_1, \dots, \text{ct}_m), (\text{seed}_1, \dots, \text{seed}_m), \text{pk}),$$

we set the random oracle to return seed_i on input of $(\text{state}[j_1], \dots, \text{state}[j_k], g, i)$ for all i . We then record (ct_i, \perp) , for $i = 1, \dots, m$, in the internal game state state' and return it to \mathcal{A} .

- Upon a query (c_1, \dots, c_m, g, k) to the random oracle which has not been made before:
 - If there exists an $i \in [1, \dots, m]$ such that $c_i = (\text{ct}_i, \pi_i) \notin \text{state}'$ and $\Pi.\text{Verify}_{\text{pp}}(\text{pk}, \text{ct}_i, \pi_i) = 0$, which includes the case $\pi_i = \perp$, we run \mathcal{I} on ct_i . If it fails, choose $\text{seed} \leftarrow \mathcal{R}$, set the output to the query to seed and return it to \mathcal{A} . Otherwise, we use \mathcal{I} 's output to add ct_i to the game state in the obvious way.

- Then, for each $c_i = (\mathbf{ct}_i, \pi_i) \notin \text{state}'$ where $\Pi.\text{Verify}_{\text{pp}}(\text{pk}, \mathbf{ct}_i, \pi_i) = 1$, run the extractor, to obtain $(m_i, \widehat{\text{seed}}_i) \leftarrow \mathcal{E}_{\mathcal{A}}(\text{pp}, \text{pk}, \mathbf{ct}_i, \pi_i, \rho)$. Observe, with overwhelming probability, we have that $\mathbf{ct}_i = \text{TFHE}.\text{Enc}(m_i, \text{pk}, \widehat{\text{seed}}_i)$. Send $(m_i, \widehat{\text{seed}}_i)$ to the encryption oracle to add \mathbf{ct}_i to the game state state and also record $c_i = (\mathbf{ct}_i, \pi_i)$ in state' .
- Now, let j_1, \dots, j_m correspond to the indices such that $\mathbf{ct}_i = \text{state}[j_i].c$ for all i . Send evaluation query (j_1, \dots, j_m, g) and receive $(c'_1, \dots, c'_m, \text{seed}_1, \dots, \text{seed}_m)$. Update the internal game state state' and program the random oracle such that on input (c_1, \dots, c_m, g, i) it outputs seed_i for all $i \in [1, \dots, m]$. Return seed_k to \mathcal{A} .

Note that the output distribution of the random oracle is simulated correctly, since the seed_i values are uniformly random values, and we are simulating the ACERCor game correctly to \mathcal{A} .

The only case, where the simulation might fail is when a query $q = (c_1, \dots, c_m, g, k)$ to the random oracle contains a ciphertext $c_\alpha = (\mathbf{ct}_\alpha, \pi_\alpha)$ that is not in the internal game state state' , $\Pi.\text{Verify}_{\text{pp}}(\text{pk}, \mathbf{ct}_\alpha, \pi_\alpha) = 0$, and the algorithm \mathcal{I} fails on \mathbf{ct}_α . However, in this case the simulation is still consistent, unless \mathcal{B} needs to add c_α to the internal game state state' at some point and then \mathcal{A} submits (j_1, \dots, j_m, g) to its evaluation oracle and $\text{state}'[j_i].c = c_i$ for all i .

When \mathcal{B} attempts to send this query to its own evaluation query, the resulting seed_k will be different to what \mathcal{B} set the random oracle to return on (c_1, \dots, c_m, g, k) . But the internal game state state' contains only ciphertext/proof pairs (\mathbf{ct}, π) such that either $\Pi.\text{Verify}_{\text{pp}}(\text{pk}, \mathbf{ct}, \pi) = 1$ (added by an encryption query) or $\pi = \perp$ (added by an evaluation query). So, clearly, for c_α to be added to state' , we must have $\pi_\alpha = \perp$ and \mathbf{ct}_α the result of an evaluation query.

Let $q' = (j'_1, \dots, j'_{m'}, g')$ be the query that results in c_α being added to state' . Note that \mathcal{A} must have made the query q' after query q , since otherwise c_α would be in state' when query q is submitted. Furthermore, the way that \mathcal{B} handles random oracle queries, no query of the form $(c'_1, \dots, c'_{m'}, g', k)$ (where $\text{state}'[j'_i].c = c'_i$) could have been made to the random oracle before query q . Now consider \mathcal{A}_1 as the part of \mathcal{A} up to the query q , where it sends \mathbf{ct}_α (as part of c_α). Define \mathcal{A}_2 to be \mathcal{A} after sending query q up query q' , which is the query that results in the ciphertext c_α being added to the game state. Clearly, $(\mathcal{A}_1, \mathcal{A}_2)$ violates the first condition of Definition 7 and we already established that the second condition also does not hold (since \mathcal{I} fails). Thus, $(\mathcal{A}_1, \mathcal{A}_2)$ contradicts the unpredictability assumption.

We conclude that the success probability of \mathcal{B} is at least as large as the one of \mathcal{A} . Finally, since the ZKPoK is straightline extractable, the algorithm \mathcal{B} is efficient. \square

Theorem 5. *The TFHE fully homomorphic encryption scheme, when used with the evaluation algorithm $\text{TFHE}.\text{DeRandEval}$ is sIND-CPA^D secure assuming the assumptions of Theorems 2 and 4 hold.*

Proof. Theorems 2 and 4 ensure that the scheme is IND-CPA secure and computationally ACER correct, resp., so the result follows from Theorem 1. \square

References

- [ABMP24] Andreea Alexandru, Ahmad Al Badawi, Daniele Micciancio, and Yuriy Polyakov. Application-aware approximate homomorphic encryption: Configuring FHE for practical use. Cryptology ePrint Archive, Report 2024/203, 2024. URL: <https://eprint.iacr.org/2024/203>.

- [BCF⁺25] Chris Brzuska, Sébastien Canard, Caroline Fontaine, Duong Hieu Phan, David Pointcheval, Marc Renard, and Renaud Sirdey. Relations among new CCA security notions for approximate FHE. *IACR Communications in Cryptology (CiC)*, 2(1):20, 2025. doi:10.62056/ae0iv7sf.
- [BdBB⁺25] Mathieu Ballandras, Mayeul de Bellabre, Loris Bergerat, Charlotte Bonte, Carl Bootland, Benjamin R. Curtis, Jad Khatib, Jakub Klemsa, Arthur Meyre, Thomas Montaigu, Jean-Baptiste Orfila, Nicolas Sarlin, Samuel Tap, and David Testé. TFHE-rs: A (Practical) Handbook, 2025. URL: <https://github.com/zama-ai/tfhe-rs-handbook/blob/main/tfhe-rs-handbook.pdf>.
- [BJSW25] Olivier Bernard, Marc Joye, Nigel P. Smart, and Michael Walter. Drifting towards better error probabilities in fully homomorphic encryption schemes. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part VIII*, volume 15608 of *Lecture Notes in Computer Science*, pages 181–211, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland. doi:10.1007/978-3-031-91101-9_7.
- [CCP⁺24] Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. Attacks against the IND-CPA^D security of exact FHE schemes. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024: 31st Conference on Computer and Communications Security*, pages 2505–2519, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press. doi:10.1145/3658644.3690341.
- [CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, Hanoi, Vietnam, December 4–8, 2016. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-53887-6_1.
- [CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020. doi:10.1007/s00145-019-09319-x.
- [CLOT21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 670–699, Singapore, December 6–10, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-92078-4_23.
- [CSBB24] Marina Checri, Renaud Sirdey, Aymen Boudguiga, and Jean-Paul Bultel. On the practical CPA^D security of “exact” and threshold FHE schemes and libraries. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part III*, volume 14922 of *Lecture Notes in Computer Science*, pages 3–33, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-68382-4_1.
- [Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. <https://crypto.stanford.edu/craig/>.
- [Joy24] Marc Joye. TFHE public-key encryption revisited. In Elisabeth Oswald, editor, *Topics in Cryptology – CT-RSA 2024*, volume 14643 of *Lecture Notes in Computer Science*, pages 277–291, San Francisco, CA, USA, May 6–9, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-58868-6_11.

- [KLSS23] Duhyeon Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-MLWE. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 549–580, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:[10.1007/978-3-031-38554-4_18](https://doi.org/10.1007/978-3-031-38554-4_18).
- [Lib24] Benoît Libert. Vector commitments with proofs of smallness: Short range proofs and more. In Qiang Tang and Vanessa Teague, editors, *PKC 2024: 27th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 14602 of *Lecture Notes in Computer Science*, pages 36–67, Sydney, NSW, Australia, April 15–17, 2024. Springer, Cham, Switzerland. doi:[10.1007/978-3-031-57722-2_2](https://doi.org/10.1007/978-3-031-57722-2_2).
- [LM21] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 648–677, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland. doi:[10.1007/978-3-030-77870-5_23](https://doi.org/10.1007/978-3-030-77870-5_23).
- [LMSS22] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 560–589, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. doi:[10.1007/978-3-031-15802-5_20](https://doi.org/10.1007/978-3-031-15802-5_20).
- [Sma23] Nigel P. Smart. Practical and efficient FHE-based MPC. In Elizabeth A. Quaglia, editor, *19th IMA International Conference on Cryptography and Coding*, volume 14421 of *Lecture Notes in Computer Science*, pages 263–283, London, UK, December 12–14, 2023. Springer, Cham, Switzerland. doi:[10.1007/978-3-031-47818-5_14](https://doi.org/10.1007/978-3-031-47818-5_14).
- [Tap23] Samuel Tap. *Constructing new tools for efficient homomorphic encryption*. PhD thesis, Université de Rennes, 2023. <https://theses.hal.science/tel-04587370>.