Application-Aware Approximate Homomorphic Encryption: Configuring FHE for Practical Use

Andreea Alexandru[1], Ahmad Al Badawi[1], Daniele Micciancio[1,2], and Yuriy Polyakov[1]

[1]Duality Technologies
[2]University of California, San Diego

Abstract

Fully Homomorphic Encryption (FHE) is a powerful tool for performing privacy-preserving analytics over encrypted data. A promising method for FHE over real and complex numbers is approximate homomorphic encryption, instantiated with the Cheon-Kim-Kim-Song (CKKS) scheme. The CKKS scheme enables efficient evaluation for many privacy-preserving machine learning applications. While the efficiency advantages of CKKS are clear, there is currently a lot of confusion on how to securely instantiate the scheme for any given application, especially after secret-key recovery attacks were discovered by Li and Micciancio (EUROCRYPT'21) for the IND-CPA$^{\mathsf{D}}$ setting, i.e., where decryption results are shared with other parties. On the one hand, the generic definition of IND-CPA$^{\mathsf{D}}$ is application-agnostic and often requires impractically large parameters. On the other hand, practical CKKS implementations target specific applications and use tighter parameters. A good illustration are the recent secret-key recovery attacks against a CKKS implementation in the OpenFHE library by Guo et al. (USENIX Security'24). These attacks misuse the library by employing different circuits during parameter estimation and run-time computation, yet they do not violate the generic (application-agnostic) IND-CPA$^{\mathsf{D}}$ definition.

To address this confusion, we introduce the notion of *application-aware homomorphic encryption* and devise related security definitions, which correspond more closely to how homomorphic encryption schemes are implemented and used in practice. We then formulate the guidelines for implementing the application-aware homomorphic encryption model to achieve IND-CPA$^{\mathsf{D}}$ security for practical applications of CKKS. We also show that our application-aware model can be used for secure, efficient instantiation of exact homomorphic encryption schemes.

1 Introduction

Homomorphic encryption is a cryptographic primitive that enables the evaluation of certain computations over encrypted inputs without intermediate decryptions. In its most powerful form, Fully Homomorphic Encryption (FHE) allows the evaluation of arbitrary arithmetic or boolean circuits, and has seen considerable improvements and extensions since Gentry's breakthrough [21] in 2009.

Today, there are several families of efficient FHE schemes, which can be divided along several axes. One of such axes is whether the result of the encrypted computations is exact or approximate. In the exact FHE family, we have schemes which achieve a negligible correctness error when homomorphically evaluating arithmetic circuits over finite fields: Brakerski-Gentry-Vaikuntanathan (BGV) [7] and Brakerski/Fan-Vercauteren (BFV) [6, 18], or when performing bit-wise/small plaintext-space operations homomorphically: Ducas-Micciancio (DM/FHEW) [16], Chillotti-Gama-Georgieva-Izabachene (CGGI/TFHE) [11], and Lee-Micciancio-Kim-Choi-Deryabin-Eom-Yoo (LMKCDEY) [36]. The approximate FHE family allows for small errors to corrupt the least significant bits of the message. Cheon-Kim-Kim-Song (CKKS) [10] is the main representative of this family and can be seen as an FHE scheme over fixed-point numbers, which enables significantly more efficient computations than exact FHE schemes over real-valued data in privacy-preserving large-scale applications such as secure genome-wide association studies [3], logistic regression [26], and convolutional neural network inference [35].

However, the efficiency of the CKKS scheme comes at a cost. First, *correct decryption* now takes a more involved meaning compared to the other exact (Ring) Learning With Errors-based FHE schemes, as it requires the scheme parameters to be set so that the decrypted output is not "too far" from the expected cleartext output. Second, its approximate nature makes CKKS depart from the exact FHE schemes not only conceptually, but also in terms of security, since the error corrupting the decrypted output can be used in certain passive attacks to recover the secret key.

IND-CPA$^{\mathsf{D}}$ Security. The security model for FHE schemes is

passive, i.e., FHE schemes are proven secure against Chosen Plaintext Attacks (CPA), where all ciphertexts are properly computed (using the scheme's algorithms) and the adversary cannot submit *arbitrary (maliciously chosen)* ciphertexts to decryption oracles. It is folklore that no FHE scheme can achieve IND-CCA2 security (arbitrary decryption oracle access), and only FHE schemes that do not rely on circular security assumptions can achieve IND-CCA1 security (decryption oracle access only available before the challenge). Thus, FHE schemes are not resilient to active attacks without additional security measures, e.g., zero-knowledge proofs, and should not be used in this scope.

Notwithstanding, the passive scenario needs to be extended in the case of FHE schemes, as there are vulnerabilities arising from incomplete security definitions. In particular, IND-CPA-security (access only to an encryption oracle) is too weak for approximate FHE schemes. Li and Micciancio [37] devised a key recovery attack on the CKKS scheme when the plaintext output of the computation is revealed to the adversary, i.e., when giving the adversary a very weak decryption oracle. The Li-Micciancio attack runs in expected polynomial time and exploits the fact that only from the input plaintext, output ciphertext, and output plaintext, an adversary can retrieve the error from the ciphertext and use it to compute the secret key via linear algebra techniques. To better capture the security of approximate FHE schemes, the authors introduced a new definition for passive security, IND-CPA$^{\mathsf{D}}$, which additionally gives the adversary access to an evaluation oracle and limited access to a decryption oracle for outputs of the evaluation oracle [37].

In terms of countermeasures to this kind of attack, Li et al. [38] showed how to postprocess the raw decryption output of the CKKS scheme to achieve IND-CPA$^{\mathsf{D}}$ security. The mitigation adds Gaussian noise, a.k.a. flooding noise, the magnitude of which depends on the worst-case[1] error growth of the homomorphic computation.

As a response to the Li-Micciancio attack, most of the FHE libraries that implement CKKS added practical mitigations and/or security disclaimers. For instance, Microsoft SEAL [44] included a security disclaimer advising against sharing the decrypted CKKS ciphertexts. OpenFHE [41], HElib [28], and HEAAN [27] implemented the Gaussian flooding technique, whereas Lattigo [34] implemented a rounding procedure (equivalent to noise flooding). Two primary strategies are employed to estimate the Gaussian noise used for flooding: static noise estimation and dynamic noise estimation [38]. *Static noise estimation* can be performed offline and computes the flooding noise distribution parameter based on publicly known bounds on the inputs and the function to be evaluated. Doing this using theoretical worst-case bounds can overestimate the actual noise by a large margin, and negatively impacts performance. So, in practice, is it common to use a more pragmatic approach, such as selecting a representative input from the set of allowed inputs, executing the computation, and observing the resulting noise bound, or by using heuristic noise estimation expressions. This approach is supported by the OpenFHE [41], HElib [28], Lattigo [34] and HEAAN libraries [27] (under restricted conditions). *Dynamic noise estimation* computes the approximation error during decryption at run-time using input ciphertexts and secret key, which may provide very tight approximation error estimates but may still leak some (practically small) amount of information about the secret key. The OpenFHE library supports this approach as well. All libraries allow sophisticated users to further enhance these protective measures by estimating desired output precision and establishing tighter bounds for the flooding noise.

Although the indiscriminate use of worst-case estimation to determine the decryption noise achieves security, it often leads to impractically large parameters. All the above libraries implement noise estimation procedures based on heuristics [13, 14, 40] to obtain more practical parameters. Generally, the focus in practice is to design parameters for particular applications specified by the users at run-time, rather than design parameters suitable for all applications.

Attacks by Guo et al. Recently, Guo et al. [23] claimed that any non-worst-case countermeasure added as part of the CKKS decryption is still vulnerable to the Li-Micciancio key recovery attack. In [23], "worst-case" refers not only to the input choice, but also to the encryption randomness. The authors focus on the OpenFHE library and illustrate two attacks, that we briefly explain below.

Their first attack employs two different circuits in order to illustrate a worst-case versus average-case difference. The attacker uses the circuit corresponding to the addition of n separate inputs in order to estimate the noise for decryption. However, in the computation phase, the attacker provides the addition circuit on *n copies of the same* ciphertext. Note that indeed, the error obtained by adding n independent encryptions differs from the noise incurred by adding the same encryption n times, and the latter is significantly larger, leading to a key recovery attack. Notwithstanding, notice that from a circuit perspective, although the circuits $C(x_1,\ldots,x_n) = x_1 + \ldots + x_n$ and $C'(x_1,\ldots,x_n) = x_1 + \ldots + x_1$ have the same worst-case error estimate and the same output when the inputs to the first circuit are all equal to x_1, their representations are not the same and they are two different circuits. Hence, for each of the two circuits, a different noise estimate should be computed and employed, which is what FHE libraries seem to assume in practice.

In the second attack, the authors of [23] specify a circuit with n inputs in the noise estimation phase, and a circuit with $n' \gg n$ inputs in the evaluation phase. As mentioned before, different circuits are expected to produce different errors, and existing FHE libraries seem to assume that the noise added in

[1] Here, "worst-case" is over the choice of the input and computation to be performed. Error growth can still be analyzed on the average with respect to the encryption randomness.

the decryption phase is only designed to be valid for a specific circuit.

It is clear that there is a gap in the existing FHE libraries' description of the supported application specification for which security is guaranteed during evaluation. For instance, the OpenFHE library in [23] devises the noise flooding bounds for classes of circuits using a tuned heuristic estimation with confidence intervals, and assumes that during the evaluation phase the user queries only allowed functions. However, there is also a misunderstanding around the idea of worst-case noise estimation and IND-CPA$^{\mathsf{D}}$-security. Generic IND-CPA$^{\mathsf{D}}$-security requires that one should perform noise estimation over all circuits which satisfy the desired level of correctness, and use the obtained maximum bound in the decryption mechanism. Therefore, note that even using the worst-case estimates for the circuit C as suggested in [23] would not necessarily ensure generic IND-CPA$^{\mathsf{D}}$-security, as there might be other circuits satisfying correctness for which this noise is not sufficient. Although [38]'s FHE with a differentially private mechanism formulation hinted at a formalization for classes of allowed circuits, IND-CPA$^{\mathsf{D}}$ was still used in its application-agnostic form; [23] also did not formalize this aspect, despite making certain choices in how the noise estimates were computed. This signals a second gap in the literature on the IND-CPA$^{\mathsf{D}}$-security, which resulted in the attacks from [23]. In this work, we help to bridge both gaps.

A broader perspective. These attacks can also be interpreted as specifying a certain set of encryption parameters, computed to achieve correctness for a given circuit, but then using those parameters to evaluate a distinct circuit.

Note that such attacks are not specific to approximate FHE. We discuss a folklore attack [2] on the family of exact FHE schemes, which succeeds against any kind of noise estimation technique. In the case of Learning with Errors (LWE)-based exact FHE, evaluating a different circuit than the one for which the encryption parameters were computed can lead to an overflow in the ciphertext error, corrupting the underlying plaintext. Such decryption failures can be used to mount a key recovery attack; see also [15, 39]. Moreover, concurrent works [8, 9] propose key recovery attacks similar to [2, 23] that take advantage of incorrect decryption results in exact FHE schemes. Hence, a more refined definition for exact FHE schemes (exact in the given applications class and inexact outside), which accounts for an allowed class of circuits, is also of practical interest.

A different, stronger notion for FHE security is *function privacy*, which also hides the computation performed over the encrypted inputs; in other words, all honestly produced ciphertexts should have the same distribution. However, achieving function-privacy for the popular FHE schemes requires expensive procedures such as superpolynomial noise flooding or bootstrapping [5, 17, 20, 32, 33]. The IND-CPA$^{\mathsf{D}}$ definition does not include function-privacy but can be extended to do so. Satisfying function-privacy would address the key recovery attack outlined above as the noise added during decryption would be large enough for any circuit in a specified class, but the cost would be substantial.

Finally, we note that the Li-Micciancio attack and the noise flooding mitigation are also known to be applicable in the threshold encryption setting for all FHE schemes, where distributed decryption is achieved by parties publishing a partial decryption using their secret key shares [1, 33].

1.1 Our Contribution

There is a major gap between the generic IND-CPA$^{\mathsf{D}}$ definition and the use of approximate homomorphic encryption in practice for scenarios where decryption results may be publicly shared. To achieve compliance with the generic definition, impractically large parameters would need to be used. The practical implementations of approximate homomorphic encryption in common FHE libraries typically work with more efficient parameter sets and implicitly assume that these parameters can be used only for specific applications. This leads to misinterpretation of the proper usage of FHE libraries, resulting in attacks like [23].

The main goal of our work is to close this gap by introducing the notion of *application-aware homomorphic encryption scheme*, related definitions, and guidelines for practical use of IND-CPA$^{\mathsf{D}}$-secure approximate homomorphic encryption. Concretely, our contributions can be summarized as follows:

- We present the notion of application-aware homomorphic encryption scheme and devise related security definitions, which correspond more closely to how homomorphic encryption schemes are implemented and used in practice. Application-aware homomorphic encryption adds a description of an application specification to be supported to the correctness and security of the scheme. Our definition is motivated by leveled FHE but we also show how it extends to scenarios with bootstrapping. Furthermore, the application-aware model can be used with both worst-case and average-case noise estimation.
- We formulate guidelines for implementing the application-aware homomorphic encryption model in practice. We also discuss how these guidelines can be supported by FHE libraries, for instance, by having validators for checking the compliance of a given computation with the application specification. We highlight that libraries by themselves cannot prevent all possible misuses, but can provide helper capabilities to minimize the risks of unsafe use.
- We show that the attacks by Guo et al. [23] automatically become invalid when properly formulated using our application-aware homomorphic encryption model. Moreover, our application-aware definitions provide a useful tool to understand the correct way to use FHE libraries, and detect possible misuses.

- We demonstrate that our definitions are applicable to both approximate and exact homomorphic encryption schemes. In the exact case, the goal is to forbid the output of incorrect decryption results, as the latter can be used to mount secret key recovery attacks [8, 9]. We show how our application-aware security model also addresses these recent attacks. For instance, the BFV attacks in [8, 9] bypass the normal OpenFHE mechanism to generate the parameters for a given application class and then employ this user-chosen parameter set to yield incorrect decryption results, which are then used for a successful secret key recovery attack. This violates the application-aware model.

1.2 Organization

We describe the foundational concepts and background in Section 2. Section 3 introduces our new application-aware security model and defines its properties. Section 4 proves the equivalence of IND-CPA and IND-CPA$^{\mathsf{D}}$ security notions for application-aware FHE schemes. Section 5 proposes practical guidelines for implementing the application-aware model. Section 6 examines the recent secret-key recovery attacks for both approximate and exact FHE schemes from our model's perspective and discusses the implications of worst-/average-case estimations. We summarize our contributions in Section 7 and outline future research directions.

2 Preliminaries

We denote scalars as lowercase letters and vectors as lowercase boldface letters. A function $f : \mathbb{N} \to [0,1]$ is negligible if for every polynomial p, there exists a positive scalar m such that $f(n) < 1/p(n)$ for all $n \geq m$. Let κ denote the computational security parameter. We denote general sampling as $x \leftarrow X$ where X is a distribution potentially parameterized by κ.

Since we will describe several security notions, it is useful to introduce some general notation and definitions for security and correctness properties of encryption schemes. There are two types of properties, described by either a *decision game* (e.g., indistinguishability of ciphertexts) or a *search game* (e.g., security against key recovery attacks), that we define in Appendix A.1. We will use search games also to model correctness properties, in which case we say that a scheme is $\mathcal{G}$-correct for a game $\mathcal{G}$.

We first describe describe (homomorphic) encryption syntax, using the notation from [37, 38]. For ease of exposition, we do not distinguish between the notation of public encryption keys and public evaluation keys, and denote all by pk.

Definition 1 (PK-FHE scheme)**.** *A public-key homomorphic encryption scheme with plaintext space $\mathcal{M}$, ciphertext space $\mathcal{C}$, public key space $\mathcal{PK}$, secret-key space $\mathcal{SK}$, and space of evaluatable circuits $\mathcal{L}$, is a tuple of four probabilistic polynomial-time algorithms*

$$\mathsf{KeyGen} : 1^{\mathbb{N}} \to \mathcal{PK} \times \mathcal{SK}, \qquad \mathsf{Enc} : \mathcal{PK} \times \mathcal{M} \to \mathcal{C},$$
$$\mathsf{Dec} : \mathcal{SK} \times \mathcal{C} \to \mathcal{M}, \qquad \mathsf{Eval} : \mathcal{PK} \times \mathcal{L} \times \mathcal{C} \to \mathcal{C}.$$

To illustrate some issues related to the probabilistic definition of correctness for (homomorphic) encryption, we first describe a very strong notion of perfect correctness. For any positive integer k, we write $\mathcal{L}_k$ for the set of all circuits $C(x_1,\ldots,x_k) \in \mathcal{L}$ that take precisely k inputs.

Definition 2 (Perfect Correctness)**.** *An FHE scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with message space $\mathcal{M}$ is* perfectly correct *for some class of circuits $\mathcal{L}$ if for all $x_1,\ldots,x_k \in \mathcal{M}$, $C \in \mathcal{L}_k$ and security parameter κ,*

$$\mathsf{Dec}_{sk}(\mathsf{Eval}_{pk}(C, \mathsf{Enc}_{pk}(x_1),\ldots,\mathsf{Enc}_{pk}(x_k))) = C(x_1,\ldots,x_k)$$

with probability 1 over the choice of $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and the randomness of Enc and Eval.

Requiring correctness to hold with probability 1 may seem unrealistically strong, as a negligible failure probability is usually acceptable. However, simply relaxing the above correctness property to hold except with negligible probability is usually too weak to capture a meaningful notion of correctness.[2] In order to capture the adaptive selection of the input messages x_i and circuit C, correctness properties need to be formulated as security games.

Definition 3 (Exact FHE Correctness)**.** *The correctness of an FHE scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with message space $\mathcal{M}$ and class of circuits $\mathcal{L}$ is defined by the following search game:*

$\mathsf{Expr}^{\mathsf{exact},\mathcal{E}}[\mathcal{A}](\kappa)$:
$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\kappa)$
$(x_1,\ldots,x_n) \leftarrow \mathcal{A}(1^\kappa, pk)$
$ct_i \leftarrow \mathsf{Enc}_{pk}(x_i)$ *for* $i = 1,\ldots,n$
$C \leftarrow \mathcal{A}(ct_1,\ldots,ct_n)$
$ct \leftarrow \mathsf{Eval}_{pk}(C, ct_1,\ldots,ct_n)$
if $\mathsf{Dec}_{sk}(ct) \neq C(x_1,\ldots,x_n)$
then return 1 else return 0.

The above definition illustrates some adaptive choices, but for simplicity we have considered an adversary that chooses the messages $x_1,\ldots,x_n$ non-adaptively from each other. (They may still depend on the public key.) More generally, one may let $\mathcal{A}$ choose the messages x_i sequentially, one at a time, after seeing the encryption of the previous messages, perform multiple evaluation queries, etc. We provide the adaptive definitions in full generality in Appendix A.2.

[2] Consider a pathological encryption scheme $\mathsf{Enc}_{\mathsf{pk}}(x)$ that, if the input message equals the public key, $x = \mathsf{pk}$, outputs garbage. This satisfies the definition, because for any message m, the probability that any specific public key $\mathsf{pk} = m$ is chosen is negligible. Still, the scheme is not correct if messages can be chosen after (and possibly depend on) pk. Similar issues arise if the circuit C may depend on pk or the input ciphertexts.

Remark 1. *Since the definition for search games allows for nonzero (but negligible) advantage, the definition of correctness for exact FHE schemes also allows for some small probability that ciphertexts do not decrypt correctly. However, just like any game based property, this failure probability is required to be negligible. If an FHE scheme has a non-negligible correctness error, then it does not satisfy Definition 3, and it is not considered a correct exact FHE scheme.*

In the case of an *approximate* FHE scheme, such as CKKS, it is never (or very rarely) the case that the correctness property is satisfied. In other words, any adversary will typically achieve advantage very close to 1 in the search game of Definition 3. Capturing approximate FHE schemes requires a different correctness definition, with respect to an error estimation function Estimate. While there are multiple approximate correctness definitions (see [38]), here we focus on the static approximate correctness, where Estimate can be computed as a function of the circuit C alone.

Definition 4 (Static Approximate Correctness)**.** *Let* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be an FHE scheme with normed message space* $\mathcal{M}$*. Let* $\mathcal{L}$ *be a space of circuits, and let* $\mathsf{Estimate} : \mathcal{L} \to \mathbb{R}_{\geq 0}$ *be an efficiently computable function. The tuple* $\widetilde{\mathcal{E}} = (\mathcal{E}, \mathsf{Estimate})$ *satisfies* static approximate correctness *if it is correct for the following search game:*

$$
\begin{aligned}
\mathsf{Expr}^{\mathsf{approx},\widetilde{\mathcal{E}}}[\mathcal{A}](\kappa): \ & (sk, pk) \leftarrow \mathsf{KeyGen}(1^\kappa) \\
& (x_1, \ldots, x_n) \leftarrow \mathcal{A}(1^\kappa, pk) \\
& ct_i \leftarrow \mathsf{Enc}_{pk}(x_i) \text{ for } i = 1, \ldots, n \\
& C \leftarrow \mathcal{A}(ct_1, \ldots, ct_n) \\
& ct \leftarrow \mathsf{Eval}_{pk}(C, ct_1, \ldots, ct_n) \\
& x \leftarrow \mathsf{Dec}_{sk}(ct) \\
& \text{if } \|x - C(x_1, \ldots, x_n)\| > \mathsf{Estimate}(C) \\
& \text{then return 1 else return 0.}
\end{aligned}
$$

In practice, the error estimation function Estimate is defined in a modular way, starting from the error estimate of the input ciphertexts ct_i (which are fresh encryptions of the messages x_i), and proceeding gate by gate, computing an error estimate for each wire of the circuit C. The Estimate function can be used to compute either a provable worst-case bound on the error or a possibly heuristic average-case bound. In this work, we will touch upon both cases. But, in either case, the adversary advantage in the (approximate) correctness game is always assumed to be negligible.

2.1 Generic Security Definitions

The standard definition of secure encryption (not necessarily homomorphic), against passive adversaries, is that of indistinguishability against chosen plaintext attacks.

Definition 5 (IND-CPA Security)**.** *Let* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be a homomorphic encryption scheme.* IND-CPA security *is defined by the following decision game:*

$$
\begin{aligned}
\mathsf{Expr}_b^{\mathrm{cpa}}[\mathcal{A}](1^\kappa): \ & (sk, pk) \leftarrow \mathsf{KeyGen}(1^\kappa) \\
& (x_0, x_1) \leftarrow \mathcal{A}(1^\kappa, pk) \\
& ct \leftarrow \mathsf{Enc}_{pk}(x_b) \\
& b' \leftarrow \mathcal{A}(ct) \\
& \mathsf{return}(b').
\end{aligned}
$$

The above experiment defines a corresponding notion of security, and that for a scheme to be secure, efficient adversaries should only achieve negligible advantage.

For simplicity, and as common in homomorphic encryption schemes, we assume all messages belong to a fixed message space $\mathcal{M}$—all messages have (or can be padded to) the same length. An enhanced definition, called IND-CPA$^{\mathsf{D}}$ with decryption oracles, was proposed in [37] to properly model the security of *approximate* homomorphic encryption schemes. Here, we describe a simplified version of the definition, corresponding to the common application scenario where a dataset $(x_1, \ldots, x_n)$ is encrypted at the outset, then a homomorphic computation is performed on it, and finally the result of the homomorphic computation is decrypted. The general definition can be found in Appendix A.2.

Definition 6 (IND-CPA$^{\mathsf{D}}$ Security)**.** *Let* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be a public-key homomorphic (possibly approximate) encryption scheme with plaintext space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$*.* IND-CPA$^{\mathsf{D}}$ security *is defined by the following decision game:*

$$
\begin{aligned}
\mathsf{Expr}_b^{\mathrm{cpad}}[\mathcal{A}](1^\kappa): \ & (sk, pk) \leftarrow \mathsf{KeyGen}(1^\kappa) \\
& (\mathbf{x}_0, \mathbf{x}_1, C) \leftarrow \mathcal{A}(1^\kappa, pk) \\
& \text{if } \mathbf{x}_0, \mathbf{x}_1 \notin \mathcal{M}^n \text{ or } C \notin \mathcal{L} \text{ then abort} \\
& \text{if } C(\mathbf{x}_0) \neq C(\mathbf{x}_1) \text{ then abort} \\
& \boldsymbol{ct} \leftarrow \mathsf{Enc}_{pk}(\mathbf{x}_b) \\
& ct' \leftarrow \mathsf{Eval}_{pk}(C, \boldsymbol{ct}) \\
& y \leftarrow \mathsf{Dec}_{sk}(ct') \\
& b' \leftarrow \mathcal{A}(\boldsymbol{ct}, ct', y) \\
& \mathsf{return}(b').
\end{aligned}
$$

3 Application-Aware Security Models

Ideally, the key generation procedure of a (fully) homomorphic encryption scheme would take as input a required security level, and produce a key that allows to perform arbitrary computations on encrypted data. However, the only known method to achieve FHE (i.e., the ability to perform arbitrary computations using a fixed key) requires the use of a costly bootstrapping procedure. So, many schemes settle for the weaker notion of "somewhat homomorphic" encryption,

where the user provides some information about the computation to be performed at key generation time, and obtains a key that supports that type of computations.

In theory, computations (specified by circuits) are often parameterized by a "depth" d, and the corresponding keys can be used to evaluate any depth-d arithmetic circuit C. Since d can be set to any value, this allows to perform arbitrary computations. However, the depth d needs to be specified at key generation time. The circuit depth and the type of computation can have a big impact on the key generation parameters and efficiency of the scheme. For example, it is often useful to distinguish between the addition and multiplication operations, as the multiplicative depth of the computation has a much bigger impact on efficiency than the additive depth, but both need to be accounted for.

In practice, in most applications, the circuit C to be evaluated is known in advance, and only the input data $\mathbf{x}$ is provided at run-time. In fact, this is also the reason why the standard notion of security for homomorphic encryption most commonly used does not provide *function privacy*: only the encrypted data is considered confidential, while the computation performed on them is publicly known. For approximate encryption schemes (and also some exact ones, like the GSW cryptosystem [22] and its Ring LWE adaptation [16]), the size of the input messages can also have an impact on the correctness and security properties of the scheme. To capture this, the application may specify a set $\bar{\mathcal{M}} \subseteq \mathcal{M}^k$ of possible inputs to the computation $C : \mathcal{M}^k \to \mathcal{M}$. This allows even better fine-tuning of the key generation parameters, producing an evaluation key ek (part of the public keys pk) that supports the computation of interest $C(\mathbf{x})$ on the type of inputs $\mathbf{x} \in \bar{\mathcal{M}}$ that can occur in practice. Since homomorphic encryption algorithms are naturally parameterized by the (multiplicative) depth of the computation, and can encrypt arbitrary messages in $\mathcal{M}$, ek is *syntactically* similar to any key that supports the evaluation of arbitrary circuits of the same depth as C on any input $\mathbf{x} \in \mathcal{M}^k$. However, it is important to note that using ek to evaluate such circuits and input data does not provide any correctness or even security guarantees. Unfortunately, theoretical definitions of homomorphic encryption do not explicitly model restrictions on the computation (beyond specifying the circuit depth), and this has led to some confusion and misuse of homomorphic encryption libraries.

In order to clarify the situation, we introduce the notion of *application-aware* homomorphic encryption scheme and associated security notions which more closely correspond to how homomorphic encryption schemes are implemented and (should be) used in practice. Our definitions apply both to exact and approximate homomorphic schemes. In this section, we focus on the simplest yet general type of computations/applications, where the whole input data is provided at the beginning of the computation, the circuit to be evaluated on it is chosen non-adaptively, and a single value is provided as the final output of the computation. We refer to Appendix A.2 for the fully adaptive definitions.

Definition 7. *Let $\mathcal{M}$ and $\mathcal{L}$ be the message space and function space of a homomorphic encryption scheme. A* computation $\bar{C}$ *is described by a circuit $C\colon \mathcal{M}^k \to \mathcal{M}$, and a subset of its inputs $\mathsf{dom}(\bar{C}) \subseteq \mathcal{M}^k$. The computation $\bar{C}$ represents the restriction of a circuit $C \in \mathcal{L}$ to the domain $\mathsf{dom}(\bar{C})$. We write $\bar{\mathcal{L}}$ for the set of computations, i.e., circuits $\bar{C}$ with restricted domain $\mathsf{dom}(\bar{C})$. An* application $\mathsf{App} \subseteq \bar{\mathcal{L}}$ *is simply a set of computations that admits a compact description.*[3]

We define an application App to be a subset of $\bar{\mathcal{L}}$ to capture scenarios where the user wants to generate a single set of parameters that supports one of several possible computations $\bar{C} \in \mathsf{App}$, e.g., when the specific $\bar{C}$ that needs to be evaluated is not known at key generation time, or when the same keys are used to perform multiple, different computations $\bar{C}_1, \ldots, \bar{C}_i$. (This is also useful towards function-privacy.) However, a common setting in practice is when there is a single computation $\bar{C}$ to be performed (possibly multiple times, but on different inputs $\mathbf{x} \in \mathsf{dom}(\bar{C})$). In this case, $\mathsf{App} = \{\bar{C}\}$ is a singleton set, and one can think of the application being described by a single circuit C and associated domain $\mathsf{dom}(\bar{C})$. We can now define the notion of application-aware homomorphic encryption scheme.

Definition 8. *An application-aware public-key homomorphic encryption scheme for application $\mathsf{App} \subseteq \bar{\mathcal{L}}$ is a tuple of four probabilistic polynomial-time algorithms $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ as in Definition 1 with the only difference that the key generation algorithm takes an application specification $\mathsf{App} \subseteq \bar{\mathcal{L}}$ as an additional parameter:*

$$\mathsf{KeyGen}\colon 1^{\mathbb{N}} \times 2^{\bar{\mathcal{L}}} \to \mathcal{PK} \times \mathcal{SK}.$$

The intuition is that $\mathsf{KeyGen}(\kappa, \mathsf{App})$ will produce keys that can be used to encrypt data in $\mathsf{dom}(\bar{C})$, and then evaluate $\bar{C}$ homomorphically, only for $\bar{C} \in \mathsf{App}$. In the common scenario where $\mathsf{App} = \{\bar{C}\}$ consists of a single computation which is known at key generation time, one can think of $\mathsf{KeyGen}(\kappa, \bar{C})$ as taking as input just $\bar{C} \in \bar{\mathcal{L}}$ rather than a subset of $\bar{\mathcal{L}}$.

Naturally, the correctness and security definitions should be modified accordingly. In the case of approximate homomorphic encryption, the estimation function $\mathsf{Estimate}(\bar{C})$ takes as input not only a circuit C, but also a specification of the application input domain $\mathsf{dom}(\bar{C})$. We provide a unified definition that applies both to exact and approximate homomorphic encryption schemes. Exact schemes correspond to setting $\mathsf{Estimate}(\bar{C}) = 0$, i.e., no approximation is allowed in the final result of the computation.

Definition 9 (Static Approximate Correctness)**.** *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be an (approximate) FHE scheme*

[3] For example, App may be specified by a pair of numbers (d, μ) to represent the set of all computations $\bar{C}$ where $C\colon \mathcal{M}^k \to \mathcal{M}$ is an arithmetic circuit of depth at most d, and $\mathsf{dom}(\bar{C})$ is the set of all inputs $\mathbf{x} \in \mathcal{M}^k$ such that $\|x_i\| \leq \mu$ for all i.

with (normed) message space $\mathcal{M}$ and application space from $\bar{\mathcal{L}}$, and let $\mathsf{Estimate} : 2^{\bar{\mathcal{L}}} \to \mathbb{R}_{\geq 0}$ *be an efficiently computable function. We say that the tuple* $\widetilde{\mathcal{E}} = (\mathcal{E}, \mathsf{Estimate})$ *satisfies* application-aware static approximate correctness *if it is correct for the following search game:*

$\mathsf{Expr}^{\mathsf{approx},\widetilde{\mathcal{E}}}[\mathcal{A}](\kappa)$:
$App \leftarrow \mathcal{A}(\kappa)$
$(sk, pk) \leftarrow \mathsf{KeyGen}(\kappa, App)$
$\mathbf{x} \leftarrow \mathcal{A}(pk)$
$ct_i \leftarrow \mathsf{Enc}_{pk}(x_i)$ *for* $i = 1, \ldots, n$
$\bar{C} \leftarrow \mathcal{A}(ct_1, \ldots, ct_n)$
if $\bar{C} \notin App$ or $\mathbf{x} \notin dom(\bar{C})$ then abort
$ct' \leftarrow \mathsf{Eval}_{pk}(\bar{C}, \boldsymbol{ct})$
$y \leftarrow \mathsf{Dec}_{sk}(ct')$
if $\|y - \bar{C}(\mathbf{x})\| > \mathsf{Estimate}(\bar{C})$
then return 1 else return 0.

Definition 10 (Application-aware IND-CPA$^{\mathsf{D}}$ Security)**.** *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be an (approximate) FHE scheme with (normed) message space $\mathcal{M}$ and application space from $\bar{\mathcal{L}}$.* Application-aware IND-CPA$^{\mathsf{D}}$ security *is defined by the following decision game:*

$\mathsf{Expr}_b^{\mathrm{cpad}}[\mathcal{A}](\kappa)$:
$App \leftarrow \mathcal{A}(\kappa)$
$(sk, pk) \leftarrow \mathsf{KeyGen}(\kappa, App)$
$(\mathbf{x}_0, \mathbf{x}_1, C) \leftarrow \mathcal{A}(pk)$
$\boldsymbol{ct} \leftarrow \mathsf{Enc}_{pk}(\mathbf{x}_b)$
$\bar{C} \leftarrow \mathcal{A}(ct)$
if $\bar{C} \notin App$ or $\mathbf{x}_0, \mathbf{x}_1 \notin dom(\bar{C})$ then abort
if $\bar{C}(\mathbf{x}_0) \neq \bar{C}(\mathbf{x}_1)$ then abort
$ct' \leftarrow \mathsf{Eval}_{pk}(\bar{C}, \boldsymbol{ct})$
$y \leftarrow \mathsf{Dec}_{sk}(ct')$
$b' \leftarrow \mathcal{A}(ct', y)$
$\mathsf{return}(b')$.

The exact FHE correctness definition (Definition 3) and IND-CPA-security definition (Definition 5) can be also trivially extended to the application-aware model.

Function-privacy. While the IND-CPA$^{\mathsf{D}}$ definition (both in its application-agnostic and application-aware forms) assumes public functions, it can be generalized to private functions. In the application-aware model, the function-private IND-CPA$^{\mathsf{D}}$ definition allows the adversary to also specify two distinct computations in the application class in the evaluation query. However, function-privacy often requires security even against adversaries that know the secret key, therefore the corresponding definition needs to restrict the adversary to only see ciphertexts that decrypt to equal messages.

Bootstrapping. So far, the bootstrapping procedure of an FHE scheme, which resets the noise of a ciphertext, was implicitly treated as a computation of a certain depth using an evaluation key (which is the encryption of the secret key—in Ring LWE-based FHE, all evaluation keys require circular security). Here we give more details on how to represent bootstrapping in the application-aware model.

FHE schemes are used in three main ways (we prefer an itemized description for clarity): (i) pure leveled computations, (ii) leveled computations and bootstrapping, and (iii) bootstrapping after each gate. In the case of (i), typically used for BGV, BFV and CKKS (and the main focus of our paper), the leveled computation(s) desired to be evaluated should represent the application in the KeyGen algorithm, as described so far. In the case of (iii), chiefly used for DM, CGGI and LMKCDEY, the application should be specified as a single gate with the bootstrapping procedure, as well as the total number of gates. Informally, because bootstrapping is performed after each gate—leading to full composability—the circuit becomes a function of the secret key rather than a function of the user inputs, thus only the number of gates (rather than their type) needs to be specified. The case of (ii) is a combination between (i) and (iii), where the application should be specified as the computation(s) to be performed before bootstrapping, the bootstrapping procedure, and the number of bootstrapping operations to be performed.

Ideally, the specification of the bootstrapping procedure (along with an associated probability of failure) should be done by the library. We will revisit this in Section 5.

Finally, it is crucial to note that performing computations that are not in App may result not only in incorrect results, but also in security loss, including a total key recovery attack. An adversary (to the correctness or security property) that specifies a certain App during key generation, and then carries out a computation $\bar{C}(\mathbf{x})$ for $\bar{C} \notin \mathsf{App}$ or $\mathbf{x} \notin \mathsf{dom}(\bar{C})$ during the attack is *not* a valid adversary to the application-aware FHE. Showing that an encryption scheme can be broken using an invalid adversary does not demonstrate that the scheme is insecure, since no security (or even correctness) claim is made about invalid adversaries. Rather, it should be considered as a warning against misusing the encryption scheme to carry out a homomorphic computation that it was not designed to handle. In the subsequent sections, we will show that the IND-CPA and IND-CPA$^{\mathsf{D}}$ security in the application-aware model hold as expected against valid adversaries, describe how libraries instantiate the application-aware model, and illustrate how the recent attacks in the literature use adversaries that do not follow the application-aware model.

4 Equivalence between IND-CPA and IND-CPA$^{\mathsf{D}}$ for Application-Aware Schemes

We now adapt the results of [37, 38] to the application-aware model. The proofs are deferred to Appendix A.5.

4.1 Exact Schemes

The equivalence between IND-CPA and IND-CPA$^{\mathsf{D}}$ security for exact FHE schemes can be extended from its generic formulation [37, Lemma 1] to the application-aware model. As expected, for an allowed application class, as long as the scheme satisfies exact correctness, then the decryption oracle does not give any new information to the adversary.

Theorem 1. *Let $\mathcal{E}$ be a correct[4] application-aware exact homomorphic scheme for application App $\subseteq \bar{\mathcal{L}}$. $\mathcal{E}$ is IND-CPA-secure if and only if it is IND-CPA$^{\mathsf{D}}$-secure.*

4.2 Approximate Schemes

The starting point of the transformation to achieve IND-CPA$^{\mathsf{D}}$-security is an application-aware approximate FHE scheme $\widetilde{\mathcal{E}} = (\mathcal{E}, \mathsf{Estimate})$. The scheme is assumed to satisfy only a IND-CPA-security notion and the correctness property. In our setting, the relevant correctness notion is that of static approximate correctness (Definition 9). The transformation from [38], described in Algorithm 1, uses a *mechanism* M to define new KeyGen' and Dec' algorithms, producing a new scheme $M[\widetilde{\mathcal{E}}] = (\mathsf{KeyGen}', \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}')$. The mechanism M_t is simply a randomized algorithm that adds some flooding noise, parameterized by t, to the output of the (IND-CPA$^{\mathsf{D}}$-insecure) decryption function $\mathsf{Dec}_{\mathsf{sk}}$. The amount of noise required in the decryption algorithm to achieve IND-CPA$^{\mathsf{D}}$-security is quantified in [38] by the notion of ρ-KLDP (Kullback-Leibler Differential Privacy, see Appendix A.3), for a sufficiently small value of ρ.

We remark that the input scheme $\widetilde{\mathcal{E}}$ is required to satisfy the static notion of approximate correctness with respect to the Estimate function given as input to the transformation, in order for the output scheme $M[\widetilde{\mathcal{E}}]$ to be secure. $M[\widetilde{\mathcal{E}}]$ will also satisfy approximate correctness, but with respect to a different (typically larger) $\mathsf{Estimate}' = M_t[\mathsf{Estimate}]$ function, which includes the additional error introduced by the mechanism M_t. However, since the definition of IND-CPA$^{\mathsf{D}}$ security (Definition 10) does not involve the estimation function, we will not be concerned with $\mathsf{Estimate}'$. Determining $\mathsf{Estimate}'$ is important to assess the quality of the output and usefulness of an application that performs secure approximate computations on encrypted data, but it is not directly relevant to security. What is critical for security is that the original (IND-CPA-secure) scheme is correct with respect to the original Estimate function, used to determine the parameter t used by the security mechanism M_t. The formal security statement is given in the following theorem.

Theorem 2. *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be an approximate FHE scheme with normed message space $\mathcal{M}$ and application space from $\bar{\mathcal{L}}$. Let $\mathsf{Estimate} : 2^{\bar{\mathcal{L}}} \to \mathbb{R}_{\geq 0}$ be an efficiently computable function such that $\widetilde{\mathcal{E}} = (\mathcal{E}, \mathsf{Estimate})$ be application-aware statically approximate. Let M_t be a $\rho-$KLDP mechanism on $\widetilde{\mathcal{M}}$, where $\rho \leq 2^{-\kappa-7}$. If $\mathcal{E}$ is $(\kappa+8)$-bit secure in the application-aware IND-CPA game, then $M[\widetilde{\mathcal{E}}]$ is κ-bit secure in the application-aware IND-CPA$^{\mathsf{D}}$-game.*

Li et al. [38] also illustrate how to use the notion of *bit-security* to achieve IND-CPA$^{\mathsf{D}}$-security with a lower amount of DP noise than in Theorem 2. Concretely, having (c,s)-security corresponds to ensuring c bits of computational security for the computational cryptographic primitives and s bits of statistical security for the unconditional cryptographic primitives used in a cryptosystem. The idea is that the statistical security level s cannot be lowered by the adversary simply by investing more running time: any adversary running in time T will have advantage at most $2^{-\min(s, c/\log T)}$ in breaking the scheme. So, it is often acceptable to use $s < c$. Since the statistical parameter s directly influences the additional noise used by the mechanism M_t, this results in a scheme $M[\widetilde{\mathcal{E}}]$ which is approximately correct with respect to a better $\mathsf{Estimate}'$ function, and produces higher quality results. However, it is important to understand that the adversary running time does not affect the statistical security level s only as long as the adversary makes the same number of decryption queries. This is the case, for example, in Theorem 2, which uses a security definition where the adversary is limited to a single computation/decryption. This is typically not an issue in applications of approximate FHE schemes, where the application can control the number of decryption queries. Issuing ℓ decryption queries (e.g., as in the fully adaptive security definition) allows the adversary to gain a 2^{ℓ} factor in both statistical and computational security. So, while $s = 60$ (or even lower values) may be acceptable in some applications that make a single or small number of decryption queries, it can result in a total break in applications where the same key is used to perform a large number (say, 2^{30}) of homomorphic evaluations.

Let CKKS denote an instantiation of the application-secure scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ with algorithms corresponding to the CKKS algorithms form [10, 30]. Practi-

[4] Recall that a scheme is correct if it satisfies Definition 3 or Definition 4 with $\mathsf{Estimate}(\bar{C}) = 0$, and that, like all search games, this requires decryption errors to have negligible probability. This theorem provides no security guarantees for "exact" encryption schemes that are *not correct*.

Algorithm 1 Application-aware $M[\widetilde{\mathcal{E}}]$ for App.

$\mathsf{KeyGen}'(\kappa, \mathsf{App}) :=$

1: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\kappa, \mathsf{App})$
2: $t \leftarrow \mathsf{Estimate}(\mathsf{App})$
3: $\mathsf{sk}' = (\mathsf{sk}, t)$
4: return $(\mathsf{sk}', \mathsf{pk})$

$\mathsf{Dec}'_{\mathsf{sk}'}(\mathsf{ct}) :=$

1: return $M_t(\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}))$

cally, M_t from Theorem 2 is instantiated via a discrete Gaussian mechanism (Definition 18 in Appendix A.3). Specifically, in Algorithm 1 for CKKS, for a positive σ, $\mathsf{Dec}'_{\mathsf{sk}'}(\mathsf{ct}) = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}) + \mathcal{N}_{\mathbb{Z}^n}(0, \sigma^2 \cdot t^2 \cdot \mathbf{I}_n)$. To capture the dependency on σ, we denote the corresponding CKKS scheme instantiation from Algorithm 1 as $M[\widetilde{\mathsf{CKKS}}]_\sigma$. Using the bit-security notion, one can obtain the following result for an IND-CPA$^{\mathsf{D}}$-secure instantiation, adapted from [38], which can be extended to ℓ-decryptions queries in the fully adaptive model.

Theorem 3. *If CKKS is $(c + \log_2 24)$-bit application-aware IND-CPA-secure, then, for $\sigma = \sqrt{12} \cdot 2^{s/2}$, $M[\widetilde{CKKS}]_\sigma$ is (c,s)-bit application-aware IND-CPA^D-secure.*

5 Practical Guidelines for Application-Aware Homomorphic Encryption

Recall that homomorphic encryption schemes are only *passively*-secure. Viewing the outsourced computation as a protocol between parties (with the roles of encryptor, evaluator and decryptor), parties are assumed to be *honest-but-curious*, meaning they do not deviate from the protocol design. Under Definition 10, this translates to the attacker not being allowed to submit invalid ciphertexts or functions not part of the application App selected during key generation. Therefore, users should ensure they adequately follow these specifications when working with FHE schemes.

However, misuses of cryptography can occur in practice, and one could try to make the use of the library less error-prone. Instead of relying simply on the user expertise and discipline, the library can make the application specification App an explicit parameter of the key generation procedure, store App as part of the key or evaluation context, and then implement the appropriate checks when the user makes calls to the encryption, evaluation and decryption functions. In the following, we provide a practical description of the secure application-aware FHE schemes from Section 4, as well as instructions for the FHE libraries' users and developers.

5.1 Application-Aware Approximate FHE

Definitions 8–10 and Algorithm 1 assume (implicit) validators which ensure the validity of the attacker's queries. However, in practice, homomorphic encryption implementations typically do not include any validity checks and rely on the user's discipline to avoid improper use of the library.

Protocol 2 makes the presence of the validators explicit and provides guidelines for correct usage of approximate FHE schemes in the IND-CPA$^{\mathsf{D}}$ setting with respect to an application class App. Compared to the notation of KeyGen$'$ from Section 4, in Protocol 2, we separate the part of deriving the public parameters pp and noise estimates $\{t_i\}$ from the secret key sk sampling and public key pk computation. Specifically, the protocol includes two phases: offline, when the noise estimates are computed and scheme parameters are found without using the secret key, and online, when the actual homomorphic computation is performed using the secret key (the secret key is used to derive the evaluation keys and to perform the decryption, and is not used by the evaluator). This explicit split in two phases removes the burden from the user to compute the parameters and only requires the user to specify the same application class in both offline and online phases.

The offline phase may require multiple iterations to achieve both the desired functionality/precision and the security work factor; more concretely in the case of CKKS, to find the ciphertext modulus Q and ring dimension N. The online phase may invoke one or more validators to check whether the executed computation belongs to App. Such validation can be performed during evaluation and/or during decryption.

If needed, Protocol 2 can be run for a set of representative samples, and the maximum noise over all these runs can be used to instantiate the parameters.

Application Specification. In approximate FHE, the application specification needs to include the description of supported computations as well as the bounds on the input messages, which makes it challenging to find a compact form for the specification. When CKKS bootstrapping is used, one has to also check that the probability of decryption failure during bootstrapping is small enough (see [4] for more details) and to stipulate the bootstrapping procedure as part of the application specification. The multiplicative depth is often a useful parameter in guiding the parameter selection during the offline phase, but it may often be insufficient by itself. Therefore, the guidelines provided by libraries typically recommend running full computations (in the estimation or execution mode) to obtain tight noise bounds and generate scheme parameters.

Library-Specific Guidelines. The OpenFHE and HElib libraries provide guidelines [29, 43] for configuring IND-CPA$^{\mathsf{D}}$-secure approximate homomorphic encryption, which informally correspond to Protocol 2. Both libraries follow the two-phase approach and require running full computations (step-by-step procedures) during the offline estimation phase.

During the offline phase, OpenFHE finds tight estimates for approximation noise by computing the variance over the slots corresponding to the imaginary part of the decrypted plaintext vector (these slots are set to zero during encoding). OpenFHE also implements the flooding noise estimation method proposed in [38] based on differential privacy.

HElib provides a ciphertext-specific noise tracking functionality that can be used to check whether the computation run during the online phase belongs to the application class App. Although the HElib validator cannot cover all possible ways of misusing the library for IND-CPA$^{\mathsf{D}}$-secure approximate homomorphic encryption, the validator can detect many instances of accidental CKKS misconfiguration.

Protocol 2 Application-Aware FHE scheme for App.

Offline Noise Estimation and Parameter Generation

Input: κ, App.

Output: pp.

1: Initialize pp for the given application App (using an optimistic value of lattice dimension).
2: Compute noise estimates t using current pp on representative inputs for all computations $\bar{C}$, for each $\bar{C} \in \mathsf{App}$: $\{t_i \leftarrow \mathsf{Estimate}(\mathsf{App})\}_{i \in O(C)}$.[a]
3: Update pp based on current t.
4: If current pp do not satisfy κ, update pp (increase the lattice dimension) and go to Step 2.

Online Execution

Input: $\mathsf{pp}, \mathsf{App}$ to all, $\bar{C}, \{m_i\}_{i \in I(C)}$ to the Encryptor.

Output: $\{\mathsf{Dec}'_{\mathsf{sk}}(\mathsf{ct}_i)\}_{i \in O(C)}$ to all.

1: The Decryptor runs $\mathsf{KeyGen}'(\mathsf{pp}, \mathsf{App})$ and outputs the public key pk (including the evaluation keys) and keeps the secret key sk private.
2: The Encryptor checks[b] that $\{m_i\}_{i \in I(C)} \in \mathsf{dom}(\bar{C})$, and, if so, computes the ciphertexts $\{\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_i)\}_{i \in I(C)}$ and sends them to the evaluator, along with $\bar{C}$.
3: The Evaluator checks[c] if $\bar{C} \in \mathsf{App}$ and if yes, runs $\{\mathsf{ct}_i \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\bar{C}, \{\mathsf{ct}_j\}_{j \in I(C)})\}_{i \in O(C)}$ and outputs it. Otherwise, it outputs $\perp$.
4: The Decryptor outputs $\{\mathsf{Dec}'_{\mathsf{sk}}(\mathsf{ct}_i)\}_{i \in O(C)}$ (noise checks may also be performed before outputting the result; the decryptor may also output $\perp$ if the current noise estimate is above the bound t).[d]

[a] $I(C)$ and $O(C)$ denote all inputs and all outputs of the circuit C.

[b] This check may be hard to enforce in practice if the messages m_i are provided as different encryption calls, unless the domain $\mathsf{dom}(\bar{C}) = M^{|I(C)|}$ restricts each message independently to the same set $M \subseteq \mathcal{M}$.

[c] Notice that the evaluation cannot check that $\{m_i\}_{i \in I(C)} \in \mathsf{dom}(\bar{C})$ because it is only provided for encrypted messages, and it does not have the decryption key. Any validity check on the encrypted messages needs to be performed at encryption time, unless other inputs are provided.

[d] We remark that if parameters are properly set, with negligible correctness error, failure of a noise bound check should not happen in practice. If it does, it should be interpreted as a critical error that the scheme parameters are not set properly and the scheme may provide no security guarantees. Checking for error bounds is good for security because it limits possible information leakage to only one bit.

Library Tools. Ultimately, the FHE libraries are responsible for following the application-aware model formulated in our work and providing clear guidelines for specifying the allowed application class. In turn, the users are responsible for complying with these specifications, as there is no current design that can capture all possible invalid actions by an adversary (e.g., submitting invalid ciphertexts or correlated ciphertexts for independent inputs). To aid the latter, libraries can provide helper capabilities, such as validators or noise estimators for application specifications, that can make it significantly easier to achieve the compliance in practice. One such tool is a ciphertext-specific noise estimation capability, similar to the one available in HElib, which can detect a large approximation error before the output is presented to the user. Another useful tool is a generator of a more compact description of an application class, which could be then used to validate whether a given computation satisfies the application specification. Such a validator may be replaced by static analysis of the user program, or, when $\mathsf{App} = \{\bar{C}\}$ contains just a single computation, one could store $\bar{C}$ during key generation as part of the evaluation key, and then use an evaluation function $\mathsf{Eval}_{\bar{C}}$ with the circuit $\bar{C}$ hardwired in it (while still checking that input data $\mathbf{x} \in \mathsf{dom}(\bar{C})$). Compilers are a promising tool in aiding with application specification.

5.2 Application-Aware Exact FHE

Protocol 2 can also be applied in the case of the exact FHE family. However, there are a couple of practical differences between exact and approximate FHE settings:

- The goal of the protocol in the exact setting is to guarantee correct decryption with negligible probability of failure. Therefore, the probability of failure may also be taken as a parameter in application specification. (Note that in the approximate setting, this parameter explicitly comes up only when bootstrapping is needed.)
- The message bounds are not needed in the application specification because all plaintext operations are performed over finite fields (i.e., modulo the plaintext modulus), which significantly simplifies the noise estimation.

Application Specification. As the message bounds are not needed, compact descriptions of application specifications can be used. For example, the BGV implementation of OpenFHE takes three parameters to describe the application class: the multiplicative depth, the maximum (over all levels) number of additions per level, and the maximum number of key switching operations per level. Using these three input parameters, OpenFHE finds all scheme parameters via the procedure described in [31, Sec. 4]. The probability of failure is not set by the user because the heuristic estimates used internally for BGV/BFV estimation are conservatively chosen to achieve negligible probability of failure. More concretely, the conservative expansion factor bound of $2\sqrt{N}$ is used for all multiplications of random polynomials, for the ring dimension N (see [24, Sec. 6]), resulting in the probability of decryption failure below 2^{-100}. OpenFHE finds all scheme parameters for BGV and BFV using analytical expressions; there is no need to run the full step-by-step procedure in contrast to the approximate FHE setting.

In HElib, a more complicated representation of application specification is supported for BGV. The concept of level is not

explicitly used and ciphertext-specific noise estimation using the canonical embedding (see [25] for details) is employed to make decisions on when to invoke modulus switching (or bootstrapping) as well as to enforce the correctness of the decryption output.

Library Tools. As before, although libraries cannot prevent all possible ways of misuse, there are several tools that could help users minimize the chances of unsafe library use.

First, libraries should clearly describe the application specifications in the user API to generate the parameter set. This would minimize the risk of generating a parameter set that is not compliant with the desired application class.

Second, libraries can implement validators to detect invalid computations during run-time. For instance, for BGV, OpenFHE could check whether the depth, maximum number of additions, or maximum number of key switching operations is exceeded. Alternatively, a ciphertext-specific noise estimator, like the one implemented in HElib, could be used.

Third, the probability of decryption failure for a single bootstrapping operation can be exposed as an input parameter for certain schemes where the probability of failure has a major impact on the efficiency, e.g., DM or CGGI. The motivation for exposing this parameter is to support larger circuits (where the default bootstrapping probability of failure may be too high for the current application class). This application-specific configurability can be used to achieve better efficiency while still providing a negligibly small probability of failure for a given application class (ensuring this is required to prevent an attack described in Section 6.2). OpenFHE already provides a parameter generation tool [42] for DM, CGGI, and LMKCDEY that takes the bootstrapping probability of failure as an input argument. Finally, in practice, the number of evaluated gates (under a single key) should be accounted for by choosing a smaller failure probability for the single bootstrapping operation, such that the union bound over all gates yields an acceptable failure probability.

6 Discussion of Key Recovery Attacks

We briefly summarize the Li-Micciancio attack [37], as all attacks on CKKS, BGV and BFV from [8, 9, 23] are based on the same methodology.

Let us consider a toy version of symmetric-key CKKS based on the Ring LWE hardness problem (see Appendix A.4), where the encoding and decoding are considered errorless (the attack can be extended to the efficient CKKS scheme used in practice [10, 30]). Let the secret key be $\mathsf{sk} = (1, s)$, where $s \leftarrow \{0, -1, 1\}^N$ is sampled from the uniform ternary distribution. The encryption of a (possibly encoded) message is $\mathsf{Enc}_{\mathsf{sk}}(\mathsf{m}) = (a, b) \in R_Q^2$, where $a \leftarrow R_Q$ and $b = a \cdot s + e + \mathsf{m}$, for $e \leftarrow \mathcal{N}(0, \sigma)$ with support R_Q. To decrypt a ciphertext of form $\mathsf{ct} = (a, b)$ encrypting m, one performs $\mathsf{Dec}_{\mathsf{sk}}((a, b)) = b - a \cdot s \bmod Q$. An attacker can specify $\mathsf{m} = 0$ to the encryption oracle to obtain $\mathsf{ct} = (a, b)$, where $b = a \cdot s + e$, then nothing or the identity function to the evaluation oracle, and can finally request the decryption of ct from the decryption oracle, which returns $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}) = e \bmod Q$. The attacker retrieves $b - e = a \cdot s \bmod Q$. Making N such queries allows the adversary to form a system of linear equations in the secret s with high probability. When a is invertible, as few as a single query is sufficient to recover the secret key.

The gist of the attack is retrieving the error from the decryption query, which can be used, along with public information such as the ciphertexts, to recover the secret key. This implies that the basic CKKS scheme is not IND-CPA$^{\mathsf{D}}$-secure. Li et al. [38] further analyzed the IND-CPA$^{\mathsf{D}}$ definition and introduced a mechanism for achieving this security level for CKKS, through estimating and adding Gaussian noise during the decryption procedure such that the decryption query output does not reveal any useful information, described in Section 4.2. However, they did not formally include the estimation procedure and its relation to the evaluated function class in the definition, something which is done by libraries like OpenFHE and HElib that implement their security countermeasures. In Sections 3 and 4, we clarified the IND-CPA$^{\mathsf{D}}$ definition in the context of practical use cases of user applications, and gave precise formulations of application-aware security statements that support the use of the libraries. Software libraries that implement approximate FHE schemes with the countermeasures proposed in [38], or instantiate exact FHE schemes with parameters that satisfy appropriate correctness bounds (Section 5), satisfy the application-aware notion of IND-CPA$^{\mathsf{D}}$-security as described in Sections 3 and 4, and should be immune to the Li-Micciancio attack [37] and its variants.

Recently, a number of other works have extended the attack of [37] either (i) to defeat the security countermeasures for approximate FHE [23] or (ii) to break exact FHE schemes [8, 9]. As in the original attack [37], these works use an IND-CPA$^{\mathsf{D}}$ adversary that extracts the LWE encryption noise via decryption queries, and then uses this information to recover the secret key or break the indistinguishability of the scheme. In [8, 9, 23], this is achieved by exploiting queries to the evaluator or decryptor that are valid according to Definition 6, but *invalid* according to Definition 10. In the case of approximate FHE, this bypasses the intended effect of the noise flooding mechanism proposed in [23]. In the case of exact schemes [8, 9], this breaks the equivalence between IND-CPA and IND-CPA$^{\mathsf{D}}$-security. Note that in both cases the attack violates the assumptions of the security results from Theorems 1 and 2.

In this section we describe the attacks [8, 9, 23] using our application-aware security definitions. This serves two purposes: one is to show that these attacks highlight the dangers associated to using the libraries improperly rather than a vulnerability in the schemes or in the libraries implementing them. The other is to show how application-aware security

can be used to explain the security guarantees offered by FHE schemes and provide robust guidelines on the use of the libraries to avoid the pitfalls of [8,9,23]. We also stress that our formalism is only one of the ways to withstand these attacks: one could always select larger parameters in a non-systematic way, but we consider it to be more worthwhile to use a formalism tailored towards the application, which limits the misuse of libraries and leads to more efficient parameter sets.

6.1 Attacks on Approximate FHE

Guo et al. [23] proposed two attacks which have the goal of injecting a smaller noise in the decryption procedure than required by Theorem 2. This allows the attacker to retrieve sufficient information about the original noise in order to recover the secret key under some parameter settings.

We now translate the attacks from [23] to the language of the application-aware IND-CPA$^{\mathsf{D}}$ from Section 3. The attacks are not adaptive, meaning the attacker does not use the results of previous queries before submitting new ones, so we can use the simplified definition of IND-CPA$^{\mathsf{D}}$. (For completeness, in Appendix A.6, we show the formulation under the adaptive definition as well.)

In the case of the first attack (called "average-case estimation attack" in [23]), the attacker specifies an application $\mathsf{App} = \{\bar{C}\}$ on n inputs, described by the circuit $C(x_1,\ldots,x_n) = x_1 + \ldots + x_n$, for which the parameters and noise estimate for the differentially private mechanism are being computed.[5] The attacker then asks for the encryption of inputs $x_1,\ldots,x_n$, but specifies the function[6] $C'(x_1,\ldots,x_n) = x_1 + \ldots + x_1$ for evaluation. Despite the fact that when $x_i = x_1$, for $i = 2,\ldots,n$, the outputs of the two computations are the same, the computations $\bar{C}'$ and $\bar{C}$ are different, and, importantly, $\bar{C}' \notin \mathsf{App}$. This means $\bar{C}'$ is not a valid query according to Definition 10. We remark there is nothing wrong about the circuit C' itself, and a user may want legitimately evaluate C' on encrypted inputs. But, if so, it should include $\bar{C}'$ in the application specification $\mathsf{App} = \{\bar{C},\bar{C}'\}$ during key/parameter generation. (See below for additional discussion about this.) The same holds for a different computation which computes the addition recursively (in a tree shape), also explored in [23].

In the case of the second attack (called "empirical noise attack" in [23]), the attacker now specifies for the runtime evaluation the circuit $C''(x_1,\ldots,x_{n'}) = x_1 + \ldots + x_{n'}$, for $n' \neq n$, while still using $\mathsf{App} = \{\bar{C}\}$ defined above. But $\bar{C}'' \neq \bar{C}$ and $\bar{C}'' \notin \mathsf{App}$, also rendering this query invalid according to Definition 10.

The authors of [23] suggest that in order to avoid attacks, one should always use worst-case noise estimates. But from the description above it should be clear that the real issue exploited by their attack is not the difference between average-case and worst-case error estimates. Choosing the scheme parameters based on worst-case error estimates for $\bar{C}$, and then evaluating $\bar{C}'$ homomorphically using the same key, based on the ad-hoc analysis that $\bar{C}$ and $\bar{C}'$ produce similar worst-case noise estimates, is error-prone and theoretically unjustified. If the user also wants to evaluate $\bar{C}'$, it is both easier and less error-prone to include $\bar{C}'$ in App at key generation time, and let the library choose the parameters accordingly. Moreover, while $\bar{C}$ and $\bar{C}'$ have the same worst-case noise bounds, this is not the case for other circuits like $\bar{C}''$. In any case, if $\bar{C}' \notin \mathsf{App}$, one cannot invoke Theorem 2 and claim generic IND-CPA$^{\mathsf{D}}$-security. This is true even if the differentially-private mechanism applied in decryption uses worst-case noise bounds over $\bar{C}$ and $\bar{C}'$. The reason for this is because the generic IND-CPA$^{\mathsf{D}}$-security definition (Definition 6) requires noise bounds over *all* possible circuits allowed by the scheme's parameters.

Instead, for application-aware IND-CPA$^{\mathsf{D}}$-security (Definition 10), one can clearly define and focus on a specific computation class App. The practical significance of this model is that one can thus compute smaller parameters (leading to more efficient implementation), as long as only valid computations are performed, and still achieve application-aware IND-CPA$^{\mathsf{D}}$-security. Importantly, this also allows the use of non-worst-case noise bound estimation, and refutes the claim from [23] that any usage of non-worst-case estimates is insecure. The estimation should be performed globally over the class of allowed computations (with confidence intervals over the noise introduced by the FHE schemes operations), but it does not have to account for disallowed computations.

From the perspective of Section 5, these attacks violate Protocol 2 as the computation they run during the online phase does not belong to the application class App specified during the offline estimation phase. Crucially, it is the responsibility of the libraries such as OpenFHE to clarify the guidelines for the application specification, if it should refer to the same computation during offline and online phases.

Finally, another claim from [23] against non-worst-case estimates is that "the user in possession of the secret key may lack prior knowledge of the function to be evaluated, as could occur in cases involving private circuits". Presuming the function is private falls under the function-privacy model. We discussed in Section 3 that function-privacy requires a different definition of IND-CPA$^{\mathsf{D}}$, both in the generic and application-aware models. Satisfying these new definitions would require different estimations than in the non-function-private model, and the existing libraries do not claim security in the function-private model.

[5] The noise estimate for the encrypted computation is not performed worst-case over the ciphertexts input to the homomorphic evaluation of $\bar{C}$, but using heuristics assuming independently honestly generated ciphertexts; however, this is actually not relevant to our discussion of the application-aware model.

[6] Technically, the attack in [23] did not specify a function, but was described informally as adding n copies of the first ciphertext $\mathsf{ct}_1 = \mathsf{Enc}_{\mathsf{pk}}(x_1)$ obtained from the encryption queries. But since in an IND-CPA$^{\mathsf{D}}$ attack the adversary can only choose input messages x_i and not ciphertexts ct_i, the correct (and only) way to add a ciphertext to itself $\mathsf{ct}_1 + \ldots + \mathsf{ct}_1$ is to use a circuit that reuses the same (encrypted) input like C'.

6.2 Attacks on "Exact" FHE schemes

By definition, exact FHE schemes (Definition 3) are a special case of approximate FHE schemes with a perfect estimation function $\mathsf{Estimate}(\mathsf{App}) = 0$ that leaves no space for approximation errors. There is still a way in which decryption may deviate from recovering the input message: a decryption failure, and the Li-Micciancio attack [37] extends to such schemes. According to Definition 3, decryption failures should occur with at most negligible probability (see Remark 1). However, if a cryptographic library is misconfigured or improperly used, decryption failures may occur with noticeable probability and be exploited in attacks.

There are several folkore attacks on schemes such as BGV/BFV where decryption is allowed despite an overflown ciphertext error. We briefly describe in the following such an attack from [2], as part of a discussion following the responsible disclosure of the attack in [37].

Consider a toy version of the BGV scheme with plaintext space $\mathbb{Z}_p$ and ciphertext space R_Q, with the secret key $\mathsf{sk} = (1,s)$, where $s \leftarrow \{0,-1,1\}^N$ is sampled from the uniform ternary distribution. The encryption of a (possibly encoded) message $\mathsf{Enc}_{\mathsf{sk}}(\mathsf{m}) = (a,b) \in R_Q^2$, where $a \leftarrow R_Q$ and $b = a \cdot s + p \cdot e + \mathsf{m}$, for $e \leftarrow \mathcal{N}(0,\sigma)$ with support R_Q. To decrypt a ciphertext of form $\mathsf{ct} = (a,b)$, one performs $\mathsf{Dec}_{\mathsf{sk}}((a,b)) = b - a \cdot s \bmod p$. An attacker submits the message $\mathsf{m} = 0$ to the encryption oracle, resulting in a ciphertext $\mathsf{ct} = (a,b)$ with randomly sampled a and $b = a \cdot s + p \cdot e \bmod Q$. The attacker then requests the evaluation of a circuit adding the input to itself $p^{-1} - 1 \bmod Q$ times and finally asks for the resulting ciphertext $\mathsf{ct}' = \mathsf{ct} + \ldots + \mathsf{ct} = (a',b')$. Note that $\mathsf{ct}' = (a \cdot p^{-1} \bmod Q, (a \cdot s + p \cdot e) \cdot p^{-1} \bmod Q)$. As such, $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}') = (p \cdot e) \cdot p^{-1} \bmod p = e \bmod p$. It is clear that when $e < p$, then one can recover the secret key via linear algebra. The attack can also be extended for when $e \geq p$.

What makes this attack possible is allowing for as many additions as to lead to an incorrect decryption result (and implicitly, to a scheme that does not satisfy exact correctness even probabilistically, with negligible failure probability). Note that in this attack, p and Q are specified first, and the number of additions required depends on their values. However, in Definitions 8–10, one specifies the application class in the key generation algorithm. This would translate to the user specifying the addition circuit, which fixes the number of inputs and the number of addition gates, and obtaining public parameters that are correct with respect to this computation (one can specify multiple circuits, but all have the number of inputs and gates fixed). Then, during run-time, only the evaluation of this computation would be allowed, which with high probability, disallows adding a value for $p^{-1} - 1 \bmod Q$ times.

Recently, Checri et al. [8] and Cheon et al. [9] proposed similar key recovery attacks against OpenFHE and other libraries. Their attacks on BGV/BFV schemes fix the parameters of the schemes—implicitly, by specifying a computation class App which returns parameters for achieving exact correctness for App—and then using these parameters to perform a different computation $\bar{C} \notin \mathsf{App}$. This computation $\bar{C}$ is chosen such that $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{pk}}(\bar{C}, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)) \neq \bar{C}(x_1, \ldots, x_n)$, for $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_i)$, $i = 1, \ldots, n$, see [8, 9] for more concrete details. Since $\bar{C} \notin \mathsf{App}$, neither application-aware correctness nor $\mathsf{IND\text{-}CPA^D}$-security is guaranteed. Therefore, there is no reason to expect encryption to be secure. The attacks in [8, 9] demonstrate that this lack of security is not just a (well-known, but theoretical) possibility, but a concrete threat in practice.

Concretely, this is a good example of the risks of not following Protocol 2. The attacks in [8, 9] against OpenFHE go through not because of the use of average-case noise estimation instead of worst-case estimation (for addition in BGV/BFV, OpenFHE actually uses worst-case estimation), but because the circuit to be evaluated is not specified correctly. In the case of the attack in [9] the proper OpenFHE use of BGV/BFV for this scenario would require the user to supply the number of additions before generating the parameters using SETEVALADDCOUNT, or an equivalent multiplicative depth using SETMULTIPLICATIVEDEPTH. One can check that, when this is done correctly, a larger parameter set is generated by OpenFHE than the one used for the attack. The same can be done for the attack in [8] where the circuit is purely made of addition gates. Additionally, to reduce the number of additions, [8] uses an optimization involving rotations, which should also be accounted for when specifying the allowed application class via SETKEYSWITCHCOUNT, as it affects the noise estimation bound.

The works [8, 9] also describe attacks against the schemes implemented in the TFHE-rs [45] and TFHELib [12] libraries, which fall in a different category. CGGI/TFHE is a DM/FHEW-like cryptosystem that allows to evaluate arbitrary (boolean) circuits performing bootstrapping after each gate. In this context, the set of functions $\mathcal{L}$ supported by the scheme does not represent entire applications, but individual gates, which are combined together to evaluate a complex function. Since bootstrapping is applied after every gate, this should make the library easier to use, and parameter configuration less error-prone. The attacks in [8, 9] use the default parameters of specific libraries. However, these attacks do not necessary apply to custom parameters and/or other libraries, e.g., the FHEW/TFHE implementation in OpenFHE allows the user to generate a custom parameter set with a user-defined bootstrapping probability of failure that corresponds to negligible decryption error even for large circuits.

The attack in [8] exploits the fact that TFHE (like essentially all lattice-based cryptosystems) is linearly homomorphic and supports the evaluation of addition operations (in fact, exclusive-or, or addition modulo 2) very efficiently, without resorting to bootstrapping. Then, by evaluating a huge number of additions (beyond what can be supported by the selected scheme parameters—TFHELib [12] does not auto-

matically apply bootstrapping after a gate evaluation)—one can trigger decryption errors and recover the secret key. Conceptually, this attack is similar to the attacks described before: since addition is performed without bootstrapping, the maximum number of homomorphic additions before bootstrapping should be specified at key generation time, and taken into account during parameter generation. Our application-aware security definition allows only the evaluation of circuits that respect that bound. Alternatively, as the number of additions approaches the allowed limit, the library or user may inject a bootstrapping operation to reset the noise to acceptable levels.

On the other hand, the attack in [9] exploits a weakness of the TFHE-rs library: the choice of a fairly large correctness error of 2^{-40} or even 2^{-17} (for Concrete-python). Note that such parameters are not correct according to Definition 3, which requires decryption errors to have negligible probability. Selecting such parameters to maximize performance allows [8, 9] to trigger decryption errors and mount a key recovery attack.

7 Concluding Remarks

In this work, we proposed a framework for secure and efficient configuration of approximate FHE schemes by introducing the concept of application-aware FHE and its associated security definitions. Our framework addresses the current confusion surrounding the secure instantiation of the CKKS scheme in practice, especially after recent secret-key recovery attacks which highlighted the practical limitations of the generic IND-CPA$^{\mathsf{D}}$ model. Unlike generic and potentially hard-to-satisfy security models, our application-aware security model reflects the real-world use of FHE. We provide practical guidelines for FHE developers and users to achieve IND-CPA$^{\mathsf{D}}$ security in the application-aware setting. We also demonstrate that our application-aware model can be used to securely instantiate exact FHE schemes.

We see this work as a first step in establishing the practical procedures for the secure, efficient use of FHE in the IND-CPA$^{\mathsf{D}}$ setting. In the future, we envision multiple tools that could help FHE users to enforce the application-aware model. For instance, more compact application specifications could be developed for approximate FHE. Automated validators that check that a specific computation belongs to the allowed application class could also be useful. Online noise estimation tools could provide a mechanism to detect unsafe use of a library. An important research problem is to reduce the cost of noise flooding, which currently requires increasing the CKKS scaling factor by 30 or more bits.

Note that while FHE has a great potential for privacy-preserving computations, realizing it in practice brings about many challenges. First, library developers aim for better usability to hide complicated details of underlying FHE schemes. However, these simplified interfaces might increase the chance of library misconfiguration and misuse. Second, the honest-but-curious assumption in the FHE security model is hard to satisfy in practice. Although cryptography provides tools such as authentication, commitments, and zero-knowledge proofs to ensure adherence to established protocols, these solutions are often too computationally expensive in the context of FHE applications [19], and are actively being researched. A more practical alternative are legal auditing and other non-cryptographic approaches, which can offer valuable complementary measures.

Acknowledgements. The authors would like to thank Nicholas Genise for helpful discussions on the CKKS noise estimation and IND-CPA$^{\mathsf{D}}$ security.

References

[1] Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., and Wichs, D. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT 2012* (Apr. 2012), D. Pointcheval and T. Johansson, Eds., vol. 7237 of *LNCS*, Springer, Heidelberg, pp. 483–501.

[2] Bergamaschi, F., Cheon, J. H., Dai, W., Halevi, S., Kim, A., Kim, D., Laine, K., Li, B., Micciancio, D., Papadimitriou, A., Polyakov, Y., Shoup, V., Song, Y., and Vaikuntanathan, V. Personal Communication, 2020. Email thread on October 30, 2020.

[3] Blatt, M., Gusev, A., Polyakov, Y., and Goldwasser, S. Secure large-scale genome-wide association studies using homomorphic encryption. *Proceedings of the National Academy of Sciences 117*, 21 (2020), 11608–11613.

[4] Bossuat, J.-P., Mouchet, C., Troncoso-Pastoriza, J., and Hubaux, J.-P. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Advances in Cryptology – EUROCRYPT 2021* (Cham, 2021), A. Canteaut and F.-X. Standaert, Eds., Springer International Publishing, pp. 587–617.

[5] Bourse, F., del Pino, R., Minelli, M., and Wee, H. FHE circuit privacy almost for free. In *CRYPTO 2016, Part II* (Aug. 2016), M. Robshaw and J. Katz, Eds., vol. 9815 of *LNCS*, Springer, Heidelberg, pp. 62–89.

[6] Brakerski, Z. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO 2012* (Aug. 2012), R. Safavi-Naini and R. Canetti, Eds., vol. 7417 of *LNCS*, Springer, Heidelberg, pp. 868–886.

[7] Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012* (Jan. 2012), S. Goldwasser, Ed., ACM, pp. 309–325.

[8] Checri, M., Sirdey, R., Boudguiga, A., Bultel, J.-P., and Choffrut, A. On the practical CPAD security of "exact" and threshold FHE schemes and libraries. Cryptology ePrint Archive, Paper 2024/116, 2024. https://eprint.iacr.org/2024/116.

[9] Cheon, J. H., Choe, H., Passelègue, A., Stehlé, D., and Suvanto, E. Attacks against the INDCPA-D security of exact FHE schemes. Cryptology ePrint Archive, Paper 2024/127, 2024. https://eprint.iacr.org/2024/127.

[10] Cheon, J. H., Kim, A., Kim, M., and Song, Y. S. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT 2017, Part I* (Dec. 2017), T. Takagi and T. Peyrin, Eds., vol. 10624 of *LNCS*, Springer, Heidelberg, pp. 409–437.

[11] Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *ASIACRYPT 2017, Part I* (Dec. 2017), T. Takagi and T. Peyrin, Eds., vol. 10624 of *LNCS*, Springer, Heidelberg, pp. 377–408.

[12] CHILLOTTI, I., GAMA, N., GEORGIEVA, M., AND IZABACHÈNE, M. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

[13] COSTACHE, A., CURTIS, B. R., HALES, E., MURPHY, S., OGILVIE, T., AND PLAYER, R. On the precision loss in approximate homomorphic encryption. Cryptology ePrint Archive, Report 2022/162, 2022. https://eprint.iacr.org/2022/162.

[14] COSTACHE, A., NÜRNBERGER, L., AND PLAYER, R. Optimisations and tradeoffs for HElib. In *CT-RSA 2023* (Apr. 2023), M. Rosulek, Ed., vol. 13871 of *LNCS*, Springer, Heidelberg, pp. 29–53.

[15] D'ANVERS, J.-P., VERCAUTEREN, F., AND VERBAUWHEDE, I. The impact of error dependencies on ring/mod-LWE/LWR based schemes. In *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019* (2019), J. Ding and R. Steinwandt, Eds., Springer, Heidelberg, pp. 103–115.

[16] DUCAS, L., AND MICCIANCIO, D. FHEW: Bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT 2015, Part I* (Apr. 2015), E. Oswald and M. Fischlin, Eds., vol. 9056 of *LNCS*, Springer, Heidelberg, pp. 617–640.

[17] DUCAS, L., AND STEHLÉ, D. Sanitization of FHE ciphertexts. In *EUROCRYPT 2016, Part I* (May 2016), M. Fischlin and J.-S. Coron, Eds., vol. 9665 of *LNCS*, Springer, Heidelberg, pp. 294–310.

[18] FAN, J., AND VERCAUTEREN, F. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

[19] FRANKLE, J., PARK, S., SHAAR, D., GOLDWASSER, S., AND WEITZNER, D. J. Audit: Practical accountability of secret processes. *Cryptology ePrint Archive* (2018).

[20] GENTRY, C. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[21] GENTRY, C. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC* (May / June 2009), M. Mitzenmacher, Ed., ACM Press, pp. 169–178.

[22] GENTRY, C., SAHAI, A., AND WATERS, B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. Cryptology ePrint Archive, Report 2013/340, 2013. https://eprint.iacr.org/2013/340.

[23] GUO, Q., NABOKOV, D., SUVANTO, E., AND JOHANSSON, T. Key recovery attacks on approximate homomorphic encryption with non-worst-case noise flooding countermeasures. In *Usenix Security* (2024).

[24] HALEVI, S., POLYAKOV, Y., AND SHOUP, V. An improved rns variant of the bfv homomorphic encryption scheme. In *Topics in Cryptology – CT-RSA 2019* (Cham, 2019), M. Matsui, Ed., Springer International Publishing, pp. 83–105.

[25] HALEVI, S., AND SHOUP, V. Design and implementation of helib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481, 2020. https://eprint.iacr.org/2020/1481.

[26] HAN, K., HONG, S., CHEON, J. H., AND PARK, D. Logistic regression on homomorphic encrypted data at scale. In *Proceedings of the AAAI conference on artificial intelligence* (2019), vol. 33, pp. 9466–9471.

[27] HEAAN v2.1. https://github.com/snucrypto/HEAAN, Dec 2020. SNUCRYPTO.

[28] HElib v2.3. https://github.com/homenc/HElib, Jul 2023. IBM.

[29] Security of Approximate-Numbers Homomorphic Encrypt. https://github.com/homenc/HElib/blob/master/CKKS-security.md, 2024. [Online; accessed 7-Feb-2024].

[30] KIM, A., PAPADIMITRIOU, A., AND POLYAKOV, Y. Approximate homomorphic encryption with reduced approximation error. In *CT-RSA 2022* (Mar. 2022), S. D. Galbraith, Ed., vol. 13161 of *LNCS*, Springer, Heidelberg, pp. 120–144.

[31] KIM, A., POLYAKOV, Y., AND ZUCCA, V. Revisiting homomorphic encryption schemes for finite fields. In *Advances in Cryptology – ASIACRYPT 2021* (Cham, 2021), M. Tibouchi and H. Wang, Eds., Springer International Publishing, pp. 608–639.

[32] KLUCZNIAK, K. Circuit privacy for FHEW/TFHE-style fully homomorphic encryption in practice. Cryptology ePrint Archive, Report 2022/1459, 2022. https://eprint.iacr.org/2022/1459.

[33] KLUCZNIAK, K., AND SANTATO, G. On circuit private, multikey and threshold approximate homomorphic encryption. Cryptology ePrint Archive, Report 2023/301, 2023. https://eprint.iacr.org/2023/301.

[34] Lattigo v5. https://github.com/tuneinsight/lattigo, Nov 2023. EPFL-LDS, Tune Insight SA.

[35] LEE, E., LEE, J.-W., LEE, J., KIM, Y.-S., KIM, Y., NO, J.-S., AND CHOI, W. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning* (2022), PMLR, pp. 12403–12422.

[36] LEE, Y., MICCIANCIO, D., KIM, A., CHOI, R., DERYABIN, M., EOM, J., AND YOO, D. Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In *EUROCRYPT 2023, Part III* (Apr. 2023), C. Hazay and M. Stam, Eds., vol. 14006 of *LNCS*, Springer, Heidelberg, pp. 227–256.

[37] LI, B., AND MICCIANCIO, D. On the security of homomorphic encryption on approximate numbers. In *EUROCRYPT 2021, Part I* (Oct. 2021), A. Canteaut and F.-X. Standaert, Eds., vol. 12696 of *LNCS*, Springer, Heidelberg, pp. 648–677.

[38] LI, B., MICCIANCIO, D., SCHULTZ, M., AND SORRELL, J. Securing approximate homomorphic encryption using differential privacy. In *CRYPTO 2022, Part I* (Aug. 2022), Y. Dodis and T. Shrimpton, Eds., vol. 13507 of *LNCS*, Springer, Heidelberg, pp. 560–589.

[39] MARINGER, G., FRITZMANN, T., AND SEPÚLVEDA, J. The influence of LWE/RLWE parameters on the stochastic dependence of decryption failures. In *ICICS 20* (Aug. 2020), W. Meng, D. Gollmann, C. D. Jensen, and J. Zhou, Eds., vol. 11999 of *LNCS*, Springer, Heidelberg, pp. 331–349.

[40] MURPHY, S., AND PLAYER, R. A central limit framework for ring-LWE decryption. Cryptology ePrint Archive, Report 2019/452, 2019. https://eprint.iacr.org/2019/452.

[41] OpenFHE v1.2. https://github.com/openfheorg/openfhe-development, Dec 2023. OpenFHE Org.

[42] OpenFHE Lattice Estimator. https://github.com/openfheorg/openfhe-lattice-estimator, 2024. [Online; accessed 7-Feb-2024].

[43] CKKS Noise Flooding. https://github.com/openfheorg/openfhe-development/blob/main/src/pke/examples/CKKS_NOISE_FLOODING.md, 2024. [Online; accessed 7-Feb-2024].

[44] Microsoft SEAL v4.1. https://github.com/Microsoft/SEAL, Jan. 2023. Microsoft Research, Redmond, WA.

[45] ZAMA. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.

A Appendix

A.1 More preliminaries

Definition 11 (Decision game)**.** *A* decision game $\mathcal{G}$ *is defined by an experiment* $\mathsf{Expr}_b^{\mathcal{G},\mathcal{S}}[\mathcal{A}]$ *parameterized by a bit* $b \in \{0,1\}$*, (encryption) scheme* $\mathcal{S}$ *and adversary* $\mathcal{A}$*, that on*

input a security parameter κ, runs a computation (using the algorithms of $\mathcal{S}$ and $\mathcal{A}$) and outputs a bit. The advantage $\mathsf{Adv}^{\mathcal{S}}_{\mathcal{G}}[\mathcal{A}](\kappa)$ of $\mathcal{A}$ in breaking the $\mathcal{G}$-security of $\mathcal{S}$ is

$$|\Pr\{\mathsf{Expr}^{\mathcal{G},\mathcal{S}}_0[\mathcal{A}](\kappa)=1\}-\Pr\{\mathsf{Expr}^{\mathcal{G},\mathcal{S}}_1[\mathcal{A}](\kappa)=1\}|.$$

The scheme $\mathcal{S}$ is $\mathcal{G}$-secure if for any efficient (probabilistic, polynomial time, stateful) adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathcal{S}}_{\mathcal{G}}[\mathcal{A}](\kappa)$ is negligible in κ.

Definition 12 (Search game)**.** *A* search game *$\mathcal{G}$ is defined by an experiment $\mathsf{Expr}^{\mathcal{G},\mathcal{S}}[\mathcal{A}]$ parametrized by a (encryption) scheme $\mathcal{S}$ and adversary $\mathcal{A}$, that on input a security parameter κ, outputs a bit. The advantage of $\mathcal{A}$ is simply the probability*

$$\mathsf{Adv}^{\mathcal{S}}_{\mathcal{G}}[\mathcal{A}](\kappa)=\Pr\{\mathsf{Expr}^{\mathcal{G},\mathcal{S}}[\mathcal{A}](\kappa)=1\}$$

that the experiment outputs 1. The scheme $\mathcal{S}$ is $\mathcal{G}$-secure if for any efficient (probabilistic, polynomial time, stateful) adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathcal{S}}_{\mathcal{G}}[\mathcal{A}](\kappa)$ is negligible in κ.

As a standard convention, if at any point in an experiment the adversary makes a syntactically incorrect query (e.g., indices out of range) or an invalid query (e.g., a circuit C not supported by the scheme), the experiment returns an error symbol $\perp$ in the case of a decision game and 0 in the case of search game.

A.2 Fully adaptive definitions

For simplicity, in the main body of the paper, we have considered applications where all input data is specified (and encrypted) in advance, and then a single homomorphic computation is performed on it. In practice, homomorphic encryption schemes (and libraries) allow to interleave encryption, evaluation and decryption queries, performing computations incrementally (possibly based on the result of decryption queries), reuse intermediate results of previous homomorphic computations, etc. In this section, we provide general definitions of correctness and security properties for this more general form of encrypted computations. We remark that, while the mathematical formalization of the properties in this general setting is somehow more complex (which is why we postponed it to the appendix), the essence of the definition is the same, and the main insights of our work can be already understood from the basic treatment of non-adaptive definitions.

The first thing that we need to generalize our definitions of application-aware correctness and security is a formalization of adaptive, incremental computations. Here, the set $\mathcal{L}$ of functions supported by a homomorphic encryption scheme should be understood as the set of basic operations that can be performed by a single call to Eval, and corresponding to the functions associated to the individual gates of a larger circuit representing the entire computation.

Definition 13. *Let $\mathcal{M}$ and $\mathcal{L}$ be the message space and (basic) function space of a homomorphic encryption scheme. A* computation trace *is a sequence of basic operations $[\mathsf{op}_1,\mathsf{op}_2,\ldots]$ where each op_i can be one of the following:*

- *an encryption query $\mathsf{E}(m)$, where $m\in\mathcal{M}$*
- *an evaluation query $\mathsf{H}(f,i_1,\ldots,i_k)$ where $f\colon \mathcal{M}^k\to\mathcal{M}$ is a function in $\mathcal{L}$ and $i_1,\ldots,i_k\in\{1,\ldots,i-1\}$ are indexes corresponding to previous E or H operations*
- *a decryption query $\mathsf{D}(j)$ where $j\in\{1,\ldots,i-1\}$ is the index of a previous E or H operations.*

Let Ops^* be the set of all computation sequences. An application *is specified by a subset $\mathsf{App}\subseteq\mathsf{Ops}^*$ of computation traces that is closed under prefixes, i.e., such that if $[\mathsf{op}_1,\ldots,\mathsf{op}_n]\in\mathsf{App}$, then $[\mathsf{op}_1,\ldots,\mathsf{op}_i]$ is also in App for all $i<n$.*

As usual, we assume that the set App admits a compact description, and not all possible applications (i.e., subsets of Ops^*) may be supported by a scheme. For example, App may be described by a single sequence of operations $\mathsf{op}_1,\ldots,\mathsf{op}_n$ where encryption operations $\mathsf{op}_i=\mathsf{E}(\mu_i)$ carry not a single message $m\in\mathcal{M}$ but a bound μ_i on the message size. This single sequence represents the set of all possible computation traces obtained by replacing each μ_i by any message $x_i\in\mathcal{M}$ satisfying the given size bound $\|x_i\|\le\mu_i$. Since the details of how App may be specified are scheme and application dependent, we formulate our definition using general set notation.

Remark 2. *The basic applications $\mathsf{App}'=\{\bar{C}_1,\bar{C}_2,\ldots\}$ introduced in Definition 8 correspond to a special case of Definition 13, where App is the set of all computation traces of the form*

$$[\mathsf{E}(x_1),\ldots,\mathsf{E}(x_k),\mathsf{H}(C_i,1,2,\ldots,k),\mathsf{D}(k+1)]$$

such that $C_i\colon\mathcal{M}^k\to\mathcal{M}$ and $(x_1,\ldots,x_k)\in\mathsf{dom}(\bar{C}_i)$ for some $\bar{C}_i\in\mathsf{App}'$. Naturally, if C_i is specified by a circuit with gates in $\mathcal{L}$ (rather than a single function $C_i\in\mathcal{L}$), then the operation $\mathsf{H}(C_i,1,2,\ldots,k)$ should be replaced by a sequence of operations $\mathsf{H}(g_j,1,\ldots,k_j)$ corresponding to the individual gates of C_i.

Using this definition of computation we can generalize the definitions of correctness and security as follows.

Definition 14 (Approximate Correctness)**.** *Let $\mathcal{E}=(\mathsf{KeyGen},\mathsf{Enc},\mathsf{Dec},\mathsf{Eval})$ be an (approximate) FHE scheme with (normed) message space $\mathcal{M}$ and application space from $\bar{\mathcal{L}}$, and let $\mathsf{Estimate}:2^{\bar{\mathcal{L}}}\to\mathbb{R}_{\ge0}$ be an efficiently computable function. We say that the tuple $\widetilde{\mathcal{E}}=(\mathcal{E},\mathsf{Estimate})$ satisfies*

application-aware static approximate correctness *if it is correct for the following search game:*

$$\begin{aligned}\mathsf{Expr}^{\mathsf{approx},\widetilde{\mathcal{E}}}[\mathcal{A}](\kappa):\ & App \leftarrow \mathcal{A}(\kappa)\\ & (sk,pk) \leftarrow \mathsf{KeyGen}(\kappa, App)\\ & \mathcal{A}^{\mathsf{Ops}(\cdot)}(pk)\\ & \text{return } 0\end{aligned}$$

where $\mathsf{Ops}(\cdot)$ *is an oracle defined as follows. The oracle accepts* E,H *and* D *queries, and stores a pair* $(x_i, ct_i) \in \mathcal{M} \times \mathcal{C}$ *for each* E *or* H *query. Each time* $\mathcal{A}$ *issues a new query* op_i*:*

- *If the sequence of queries issued so far* $[\mathsf{op}_1,\ldots,\mathsf{op}_i] \notin App$*, then abort the experiment with output* 0
- *if* $\mathsf{op}_i = \mathsf{E}(x_i)$*, then let* $ct_i \leftarrow \mathsf{Enc}_{pk}(x_i)$ *and return* ct_i *to* $\mathcal{A}$
- *if* $\mathsf{op}_i = \mathsf{H}(f_i, i_1, \ldots, i_k)$*, then compute* $x_i = f_i(x_{i_1},\ldots,x_{i_k})$ *and* $ct_i \leftarrow \mathsf{Eval}_{pk}(f_i, ct_1, \ldots, ct_k)$ *using previously stored pairs* (x_{i_j}, ct_{i_j})*. Then store the new pair* (x_i, ct_i)*, and return* ct_i *to* $\mathcal{A}$*.*
- *if* $\mathsf{op}_i = \mathsf{D}(j)$*, then compute* $y_i \leftarrow \mathsf{Dec}_{sk}(ct_j)$ *using previously stored pair* (x_j, ct_j)*. If* $\|y_i - x_j\| \le \mathsf{Estimate}(App)$ *return* y *to* $\mathcal{A}$*. Otherwise terminate the experiment immediately with output* 1*.*

For simplicity, in the above definition we have used an Estimate function that outputs the same bound for all decryption queries. This can be easily generalized to an estimate function that allows difference decryption queries to be answered with a varying degree of accuracy. As before, our definition applies to both exact and approximate FHE schemes, where a scheme is exact when $\mathsf{Estimate}(\mathsf{App}) = 0$ is the perfect accuracy estimation function, so that when answering decryption queries it must be $y_i = x_j$.

Again, it can be seen that basic correctness from Definition 9 is a special case of Definition 14 when restricted to the simple applications App' described in Remark 2.

The definition of $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ security is generalized similarly.

Definition 15 ($\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ Security)**.** *Let* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be an (approximate) FHE scheme with (normed) message space* $\mathcal{M}$ *and application space from* $\bar{\mathcal{L}}$*, and let* $\mathsf{Estimate} : 2^{\bar{\mathcal{L}}} \to \mathbb{R}_{\ge 0}$ *be an efficiently computable function.* Application-aware $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ security *is defined by the following decision game:*

$$\begin{aligned}\mathsf{Expr}_b^{\mathrm{cpad}}[\mathcal{A}](\kappa):\ & App \leftarrow \mathcal{A}(\kappa)\\ & (sk,pk) \leftarrow \mathsf{KeyGen}(\kappa, App)\\ & b' \leftarrow \mathcal{A}^{Ops(\cdot)}(pk)\\ & \mathsf{return}(b').\end{aligned}$$

where $\mathsf{Ops}(\cdot)$ *is an oracle defined as follows. The oracle accepts* E,H*, and* D *queries.* H *and* D *queries are similar to Definition 14.* E *queries take the form* $\mathsf{op}_i = \mathsf{E}(x_i^0, x_i^1)$ *instead of* $\mathsf{E}(x_i)$*. For each such query let* op_i^b *be the corresponding encryption operation* $\mathsf{E}(x_i^b)$*. The oracle* $\mathsf{Ops}(\cdot)$ *stores a triplet* $(x_{i,0}, x_{i,1}, ct_i) \in \mathcal{M}^2 \times \mathcal{C}$ *for each* E *or* H *query. Each time* $\mathcal{A}$ *issues a new query* op_i*:*

- *If for either* $b = 0$ *or* $b = 1$*, the sequence of queries issued so far* $[\mathsf{op}_1,\ldots,\mathsf{op}_i]$ *satisfies* $[\mathsf{op}_1^b,\ldots,\mathsf{op}_i^b] \notin App$*, then abort the experiment with output* 0
- *if* $\mathsf{op}_i = \mathsf{E}(x_i^0, x_i^1)$*, then compute* $ct_i \leftarrow \mathsf{Enc}_{pk}(x_i^b)$*, store* (x_i^0, x_i^1, ct_i)*, and return* ct_i *to* $\mathcal{A}$
- *if* $\mathsf{op}_i = \mathsf{H}(f_i, i_1, \ldots, i_k)$*, then compute* $x_i^b = f_i(x_{i_1}^b, \ldots, x_{i_k}^b)$ *for both* $b \in \{0,1\}$*, and* $ct_i \leftarrow \mathsf{Eval}_{pk}(f_i, ct_1, \ldots, ct_k)$ *using previously stored pairs* $(x_{i_j}^0, x_{i_j}^1, ct_{i_j})$*. Then store the new triplet* (x_i^0, x_i^1, ct_i)*, and return* ct_i *to* $\mathcal{A}$*.*
- *if* $\mathsf{op}_i = \mathsf{D}(j)$*, then retrieve previously stored triplet* (x_j^0, x_j^1, ct_j) *and check that* $x_j^0 = x_j^1$*. If not, abort the experiment. Otherwise, compute* $y_i \leftarrow \mathsf{Dec}_{sk}(ct_j)$ *and return* y_j *to* $\mathcal{A}$*.*

A.3 Differential Privacy

Definition 16 (KL Divergence)**.** *Let* $\mathcal{P}$ *and* $\mathcal{Q}$ *be discrete distributions with common support* $\mathcal{X}$*. The Kullback-Leibler (KL) divergence between* $\mathcal{P}$ *and* $\mathcal{Q}$ *is* $D(\mathcal{P}||\mathcal{Q}) := \sum_{x\in\mathcal{X}} \mathcal{P}(x) \ln\left(\frac{\mathcal{P}(x)}{\mathcal{Q}(x)}\right)$*.*

Definition 17 (Norm KLDP [38])**.** *For* $t \in \mathbb{R}_{\ge 0}$*, let* $M_t : B \to C$ *be a family of randomized algorithms, where* B *is a normed space with norm* $\|\cdot\| : B \to \mathbb{R}_{\ge 0}$*. Let* $\rho \in \mathbb{R}$ *be a privacy bound. We say that the family* M_t *is* ρ*-Kullback-Leibler differentially private (*ρ*-KLDP) if, for all* $x, x' \in B$ *with* $\|x - x'\| \le t$*, it holds:*

$$D\left(M_t(x)||M_t(x')\right) \le \rho.$$

Definition 18 (Gaussian Mechanism)**.** *Let* $\rho > 0$ *and* $n \in \mathbb{N}$*. Define the (discrete) Gaussian Mechanism* $M_t : \mathbb{Z}^n \to \mathbb{Z}^n$ *be the mechanism that, on input* $\mathbf{x} \in \mathbb{Z}^n$ *outputs a sample from* $\mathcal{N}_{\mathbb{Z}^n}(\mathbf{x}, \frac{t^2}{2\rho}\mathbf{I}_n)$*.*

A.4 Ring Learning With Errors

Let N be a power two. Then, the polynomial ring $R := \mathbb{Z}[X]/(X^N + 1)$ is the $2N$-th cyclotomic field's ring of integers. Let $R_Q := R/QR$ be the ring with coefficients reduced modulo Q.

The *Ring Learning With Errors* (Ring LWE) distribution with secret $s \in \mathbb{Z}^N$ under a distribution χ_s and error distribution χ, denoted as $\mathsf{RLWE}_s(N, Q, \chi)$, outputs pairs of form $(a, b) \in R_Q^2$, where $a \leftarrow R_Q$ and $b := a \cdot s + e$ for $e \leftarrow \chi$. The

decisional Ring LWE assumption with error distribution χ, secret distribution χ_s and m samples, states that for $s \leftarrow \chi_s$, the product distribution $\mathsf{RLWE}_s(N, Q, \chi)^m$ is computationally indistinguishable from the uniform distribution over $(R_Q^2)^m$.

A.5 Proofs of Section 4

Proof of Theorem 1. First, application-aware $\mathsf{IND\text{-}CPA^D}$-security implies application-aware IND-CPA-security, since for the application class App, the adversary in the IND-CPA definition is an $\mathsf{IND\text{-}CPA^D}$ adversary making an $\mathsf{Enc_{pk}}$ call, and no other $\mathsf{Eval_{pk}}$ or $\mathsf{Dec_{sk}}$ calls.[7]

In the reverse direction, assume towards a contradiction that $\mathcal{E}$ is application-aware IND-CPA-secure but not application-aware $\mathsf{IND\text{-}CPA^D}$-secure. Given an adversary $\mathcal{A}$ that breaks the $\mathsf{IND\text{-}CPA^D}$-security of $\mathcal{E}$ for an application App, we show how to build a series of adversaries $\mathcal{B}^{(i)}$ breaking the IND-CPA-security of $\mathcal{E}$, for $1 \le i \le n$, where n is the maximum number of inputs of computations inside App. We can only have equivalence for the same application class App, so both $\mathcal{A}$ and $\mathcal{B}^{(i)}$ will select the same App and computations $\bar{C}$ in the experiments.

The adversaries select an App based on the security parameter κ and receive $\mathsf{pk} \leftarrow (\kappa, \mathsf{App})$. Then each $\mathcal{B}^{(i)}$ runs $\mathcal{A}(\kappa, \mathsf{App}, \mathsf{pk})$ and answers its queries as follows:

- For each j'th encryption query (x_0, x_1), it stores the plaintexts and the computed ciphertexts, and returns to $\mathcal{A}$:
$$\mathsf{ct}_j \leftarrow \begin{cases} \mathsf{Enc_{pk}}(x_1), \text{ if } j < i \\ \mathsf{Enc_{pk}}(x_0), \text{ if } j > i \\ \mathsf{Expr}_b^{\mathrm{cpa}}[\mathcal{B}^{(i)}], \text{ if } j = i. \end{cases}$$
- For the query $\bar{C}$, it lets $\mathsf{ct}' \leftarrow \mathsf{Eval_{pk}}(\bar{C}, \mathbf{ct})$ if $\bar{C} \in \mathsf{App}, \bar{C}(\mathbf{x}_0) = \bar{C}(\mathbf{x}_1)$ and $\mathbf{x}_0, \mathbf{x}_1 \in \mathsf{dom}(\bar{C})$, and returns ct' to $\mathcal{A}$.
- For the decryption query for ct', it returns $\bar{C}(\mathbf{x}_0)$ to $\mathcal{A}$.

Finally, when $\mathcal{A}$ outputs bit b', $\mathcal{B}^{(i)}$ also outputs b'.

Define the following hybrid distributions $\mathcal{H}^{(i)} = \mathsf{Expr}_0^{\mathrm{cpa}}[\mathcal{B}^{(i)}]$ for $1 \le i \le n$ and $\mathcal{H}^{(n+1)} = \mathsf{Expr}_1^{\mathrm{cpa}}[\mathcal{B}^{(n)}]$. Note that by construction, $\mathcal{H}^{(i)} = \mathsf{Expr}_1^{\mathrm{cpa}}[\mathcal{B}^{(i-1)}]$ for $2 \le i \le n$. Using the exact correctness of $\mathcal{E}$ with respect to App, it holds that the decryption response from $\mathcal{B}^{(i)}$ to $\mathcal{A}$ are indistinguishable from those received by $\mathcal{A}$ in $\mathsf{Expr}_b^{\mathrm{cpad}}[\mathcal{A}]$. This leads to having indistinguishability between $\mathcal{H}^{(1)}$ and $\mathsf{Expr}_0^{\mathrm{cpad}}[\mathcal{A}]$ and between $\mathcal{H}^{(n+1)}$ and $\mathsf{Expr}_1^{\mathrm{cpad}}[\mathcal{A}]$. Therefore, using a union bound over the hybrid distributions gives that the advantage of $\mathcal{A}$ in the $\mathsf{IND\text{-}CPA^D}$ game is smaller than the sum over the advantages of the n adversaries $\mathcal{B}^{(i)}$ in the IND-CPA game. Given $\mathcal{E}$ was assumed to be IND-CPA-secure for App, the advantage of each $\mathcal{B}^{(i)}$ is negligible and n is polynomial in κ, therefore the advantage of $\mathcal{A}$ in the $\mathsf{IND\text{-}CPA^D}$ game for App is also negligible. □

Sketch-proof of Theorem 2. The proof follows the same steps as the proof in Theorem 2 in [38], using similar modifications for the application-aware non-adaptive case as in the proof of Theorem 1. □

Sketch-proof of Theorem 3. The proof follows the proof of Corollary 2 in [38], with the correction mentioned in https://github.com/openfheorg/openfhe-development/blob/main/src/pke/examples/CKKS_NOISE_FLOODING.md. □

For the fully adaptive version of the application-aware $\mathsf{IND\text{-}CPA^D}$ game, where the adversary can make multiple decryption queries, one has to parameterize Theorem 2 and Theorem 3 by the number of decryption queries ℓ, as done in [38].

A.6 More on Section 6

In [23], the attack is described using the adaptive definition of $\mathsf{IND\text{-}CPA^D}$ (we gave the definition of application-aware $\mathsf{IND\text{-}CPA^D}$ in Definition 15). The attack specifies the same circuit $C(x_1, \ldots, x_n) = x_1 + \ldots + x_n$ in the estimation and run-time evaluation, but in one the inputs are on different database indices, and in the other they are all at the same database index. This translates to using independent ciphertexts in the estimations but using correlated ciphertexts at run-time. The adversary does not have chosen-ciphertexts capabilities, so below we illustrate how this is achieved through the language of Definition 15.

Concretely, the computation trace specified by the attacker when choosing App is not the same as the computation trace specified to the evaluation oracle. In particular, the computation class is specified as $\mathsf{App} = \{\mathsf{E}(x_1), \ldots, \mathsf{E}(x_n), \mathsf{H}(\bar{C}, 1, 2, \ldots, n), \mathsf{D}(n+1)\}$. During the $\mathsf{IND\text{-}CPA^D}$ experiment, the attacker specifies a sequence of calls $\{\mathsf{E}(x_1), \ldots, \mathsf{E}(x_n), \mathsf{H}(\bar{C}, 1, 1, \ldots, 1), \mathsf{D}(n+1)\}$ which is not allowed in the application-aware model, since it has a different computation trace.

[7] Technically, for the adversary to issue no evaluation and decryption calls one needs to use the fully adaptive Definition 15. For the simplified Definition 9, the adversary is required to make exactly one evaluation and decryption call. In this case, one can require App to always contain a constant function mapping all $x \in \mathcal{M}$ to a fixed value $C(x) = 0$. This ensures $C(x_0) = C(x_1)$ is trivially satisfied. Then, the adversary can simply ignore the results ct', y of the trivial evaluation and decryption functions.