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 computations on encrypted data. The Cheon-Kim-Kim-Song (CKKS) scheme, an instantiation of approximate FHE, is particularly effective for privacy-preserving machine learning applications over real and complex numbers. Although CKKS offers clear efficiency advantages, confusion persists around accurately describing applications in FHE libraries and securely instantiating the scheme for these applications, particularly after the key recovery attacks by Li and Micciancio (EUROCRYPT'21) for the $\mathsf{IND\text{-}CPA^D}$ setting. There is presently a gap between the application-agnostic, generic definition of $\mathsf{IND\text{-}CPA^D}$, and efficient, application-specific instantiation of CKKS in software libraries, which led to recent attacks by Guo et al. (USENIX Security'24).

To close this gap, we introduce the notion of *application-aware homomorphic encryption* (AAHE) and devise related security definitions. This model corresponds more closely to how FHE schemes are implemented and used in practice, while also identifying and addressing the potential vulnerabilities in popular libraries. We then provide an *application specification language* (ASL) and formulate guidelines for implementing the AAHE model to achieve $\mathsf{IND\text{-}CPA^D}$ security for practical applications of CKKS. We present a proof-of-concept implementation of the ASL in the OpenFHE library showing how the attacks by Guo et al. can be countered. Moreover, we show that our new model and ASL can be used for the secure and efficient instantiation of exact FHE schemes and to counter the recent $\mathsf{IND\text{-}CPA^D}$ attacks by Cheon et al. (CCS'24) and Checri et al. (CRYPTO'24).

1 Introduction

Homomorphic encryption is a cryptographic primitive that enables the evaluation of certain computations over encrypted inputs, without intermediate decryptions. Its most powerful form, Fully Homomorphic Encryption (FHE), allows the evaluation of arbitrary arithmetic or boolean circuits, and has considerably advanced since Gentry's breakthrough in 2009 [23].

Today, there are several families of efficient FHE schemes, which can be divided according to 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) [8] and Brakerski/Fan-Vercauteren (BFV) [7, 20], or when homomorphically performing bitwise/small plaintext-space operations: Ducas-Micciancio (DM/FHEW) [18], Chillotti-Gama-Georgieva-Izabachene (CGGI/ TFHE) [13], Lee-Micciancio-Kim-Choi-Deryabin-Eom-Yoo (LMKCDEY) [37]. The approximate FHE family allows for small errors to corrupt the least significant bits of the message.[1] The Cheon-Kim-Kim-Song (CKKS) [12] 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.

However, the efficiency of the CKKS scheme comes at a cost. First, *correct decryption* now must be replaced by a more involved notion of *approximate correctness* which requires the scheme parameters to be set so that the decrypted output is not "too far" from the expected cleartext output. Second, the error corrupting the decrypted output can be used in certain passive attacks to recover the secret key.

$\mathsf{IND\text{-}CPA^D}$ 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

[1]In this sense, an approximate encryption scheme does not satisfy the standard correctness property as it may never produce exact decryptions. So, its correctness error is usually not negligible. On the other hand, "exact" schemes with high correctness error probability are not good approximate schemes because, when wrong, they may produce results that are very far from the correct one. Neither approximate nor exact schemes with non-negligible correctness error should be considered secure encryption schemes according to the standard definition, which implicitly assumes exact correctness.

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.

However, there are vulnerabilities arising from incomplete passive security definitions. In particular, IND-CPA-security (access only to an encryption oracle) is too weak for approximate FHE schemes. Li and Micciancio [38] 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 ciphertext error and use it to compute the secret key. To better capture the passive security of approximate FHE schemes, the authors introduced a new definition, $\mathsf{IND\text{-}CPA^D}$, which additionally gives the adversary access to an evaluation oracle and limited access to a decryption oracle for outputs of the evaluation oracle.

To counter this attack, Li et al. [39] showed how to post-process the raw decryption output of the CKKS scheme to achieve $\mathsf{IND\text{-}CPA^D}$ security, by adding Gaussian noise (or flooding noise), whose magnitude depends on the worst-case[2] error growth of the homomorphic computation.

Most of the FHE libraries that implement CKKS added mitigations to the Li-Micciancio attack. For instance, Microsoft SEAL [48] included a disclaimer advising against sharing the decrypted CKKS ciphertexts. OpenFHE [45], HElib [29], and HEAAN [28] implemented the Gaussian flooding technique, whereas Lattigo [36] 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 [39]. Of interest to this paper is *static noise estimation*, which 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, it is 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 [15, 16, 44]. This approach is supported by the OpenFHE [45], HElib [29], Lattigo [36] and HEAAN [28] (under restricted conditions) libraries. All libraries allow sophisticated users to further enhance these protective measures by estimating desired output precision and establishing tighter bounds for the flooding noise.

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

Attacks by Guo et al. Recently, Guo et al. [25] 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 [25], "worst-case" refers not only to the input choice, but also to encryption randomness. The authors focus on the OpenFHE library and illustrate two attacks.

In their first attack, the circuit corresponding to the addition of n independent inputs is used to estimate the noise for decryption. However, in the computation phase, the attacker provides the addition circuit of *n copies of the same* input. 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, 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, they are still two different circuits. In their second attack, the authors of [25] specify a circuit with n inputs in the noise estimation phase, and a circuit with $n' \gg n$ inputs in the evaluation phase. In both cases, different circuits are expected to produce different errors, and existing FHE libraries seem to design the noise added in the decryption phase to be valid for the same circuit.

Clearly, there are deficiencies in the existing FHE libraries' description of the supported application specifications for which security is guaranteed during evaluation. For instance, the allowed circuit description in the OpenFHE library is not sufficiently granular, as demonstrated by recent attacks, i.e., rather than the concrete circuit description, OpenFHE allows taking as input the maximum number of operations on a given level. However, there is also a misunderstanding around the idea of worst-case noise estimation and $\mathsf{IND\text{-}CPA^D}$-security. Generic $\mathsf{IND\text{-}CPA^D}$-security requires that noise estimation be performed over all circuits which satisfy the desired level of correctness, and use the obtained maximum bound in the decryption mechanism. Therefore, even using the worst-case estimates for the circuit C as suggested in [25] would not necessarily ensure generic $\mathsf{IND\text{-}CPA^D}$-security, as there might be other circuits for which this noise is not sufficient. This signals a second shortcoming in the literature, which resulted in the attacks from [25]. In this work, we address both issues.

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 [3] 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 [17, 41]. Moreover, concurrent works [10, 11] propose key recovery attacks similar to [25] that take advantage of incorrect decryption results in exact FHE schemes. Hence, a more refined definition for exact FHE schemes (exact in the given application 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. Achieving function privacy for the popular FHE schemes requires expensive procedures such as superpolynomial noise flooding or bootstrapping [6, 19, 22, 33, 34]. The $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ definition does not include function privacy but can be extended to do so. Satisfying function privacy would address the key recovery attack outlined above, but the cost would be substantial.

On a separate note, 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 [2, 34].

Related work. We remark that the need to restrict the evaluation function of an (approximate) FHE scheme to functions over which the scheme is (approximately) correct is somewhat implicit in previous work. For example, this assumption is underlying in [38, Lemma 1] when proving the equivalence between $\mathsf{IND\text{-}CPA}$ and $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ for FHE schemes with exact decryption. This was already observed in [35, Theorem 6.5], where [38, Lemma 1] is rephrased using a circuit class to make this assumption explicit. This is related to our notion of application-aware security, but there are important differences: in [35], the circuit class is a global parameter of the FHE scheme, and it is not part of the syntax of the key generation algorithm. On the other hand, in our application-aware security model, the user provides a set of functions (as part of the application specification) as an extra input to the key generation algorithm, which will use it to fine-tune the parameter generation. Moreover, [35] defines the circuit class as a set of functions mapping *ciphertexts* to *ciphertexts*. While functions between ciphertexts are required for noise estimation, it is problematic to include this in the definitions (see Appendix D for a detailed discussion). In our work, we address the same concerns in an abstract and more satisfactory way by explicitly using an *Application Specification Language* (ASL), which easily maps to (well-defined) functions on messages, but may include additional information, such as directives to the evaluation function on where to apply bootstrapping.

Finally, recent and concurrent works focusing on definitions towards active security for FHE such as [1, 9, 40, 49] are orthogonal to our work, but also take into consideration some concept of application specification. For instance, the FuncCPA definition [1] is in a similar vein as our application-aware model, but with the crucial difference that the adversary submits a vector of potentially maliciously crafted ciphertexts (instead of plaintexts) and a function to an oracle, which returns an encryption of the function applied on the decryptions of the submitted ciphertexts. Achieving this kind of malicious security requires a notion of circuit privacy and a strong sanitization procedure. The definition of maliciously secure verifiable FHE from [49] also specifies a function at key generation time. Moreover, works such as [40] assume schemes to be correct in the same way that we do, explicitly prohibiting attacks such as [10].

1.1 Our Contribution

There is a major gap between the generic $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ definition and the use of approximate FHE in practice. To achieve compliance with the generic definition, impractically large parameters would need to be used. The practical implementations of approximate FHE in common FHE libraries typically work with more efficient parameter sets and implicitly assume that these parameters can be used only for specific applications. However, no or insufficient validators are implemented in FHE libraries to ensure the above. This led to misinterpretation of the proper usage of FHE libraries, and resulted in attacks like [10, 11, 25].

Our work closes this gap by introducing the notion of *Application-Aware Homomorphic Encryption*, related definitions, and guidelines for the practical use of $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-secure FHE, including an application specification language. We summarize our contributions in the following.

First, we present the notion of application-aware homomorphic encryption schemes and devise related security definitions, which correspond more closely to how homomorphic encryption schemes are implemented and used in practice. Application-aware FHE 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.

Second, we formulate guidelines for implementing the application-aware FHE model in practice and, to this end, we introduce an application specification language. We discuss how this model can be supported by FHE libraries, e.g., by 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.

Third, we provide a proof-of-concept implementation of an example ASL in OpenFHE showing how the attacks by Guo et al. [25] can be countered. Moreover, our application-aware definitions and ASL provide a useful tool to understand the correct way to use FHE libraries and detect misuses. We also demonstrate that our application-aware definitions are applicable to both approximate and exact homomorphic encryption schemes, and we include a proof-of-concept implementation for ASL-based BFV in OpenFHE. 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 [10, 11]. Using ASL, the attacks in [10, 11] can be prevented by correctly describing the intended circuits and generating the appropriate noise parameters.

1.2 Organization

We describe the foundational concepts in Section 2. Section 3 introduces our new application-aware security model and defines its properties. We describe an application specification language in Section 4 and propose practical guidelines for implementing the application-aware model in Section 5. Section 6 examines the recent secret-key recovery attacks for both approximate and exact FHE schemes based on our new model. We summarize our contributions in Section 7 and outline future research topics. Further theoretical results on our model and attack descriptions are provided in the Appendix.

2 Preliminaries

We denote scalars as lowercase letters and vectors as lowercase boldface letters. We use $x \leftarrow X$ for general sampling from a distribution X.

2.1 Measuring Security

Following [39], we measure the security of cryptographic primitives using the bit-security framework of [43] and its extension [39, 42] to computational/statistical bit-security.

Cryptographic primitives are usually parametrized by a main security parameter κ, which can be either a discrete security level (e.g., Level 1–5 security) or a positive integer (in the asymptotic setting). The number of bits of computational $c(\kappa)$ or statistical security $s(\kappa)$ offered by a cryptographic primitive is a function of the main security parameter κ. The difference between c and s is that computational properties are based on the assumption that some computational problem is hard to solve, while statistical properties hold unconditionally. To take into account possible algorithmic improvements in solving the underlying problem (and the ability to improve an attack's success probability by investing computational effort) it is common practice to use higher values for c than for s. For primitives that use both computational assumptions and statistical security techniques, the concrete security level is specified by the pair of numbers (c,s). If a (purely computational) cryptographic primitive achieves c-bits of computational security, then it is (c,s)-secure for any s. Similarly, s-bits of statistical security imply (c,s)-security for any c.

When we say that an adversary has *negligible* advantage in breaking a primitive, we mean that the primitive achieves (c,s) security, for appropriately large values of c,s, e.g., $(c,s) = (128,64)$. In the asymptotic security setting this should be interpreted as achieving $(c(\kappa),s(\kappa))$-security, for appropriate functions $c(\kappa),s(\kappa) = \omega(\log\kappa)$ at least superlogarithmic, and most typically linear in κ, e.g., $c(\kappa) = \kappa$ and $s(\kappa) = \kappa/2$. In particular, any attack should have either running time exponentially large in κ, or success probability exponentially small in κ (or a combination of the two.)

Within the context of computational/statistical security, the distinction between correctness and security properties (and acceptable error levels) is somehow artificial, because (as demonstrated by recent attacks [10, 11, 25]) failure of correctness can have a major impact on security. Thus, correctness should be regarded as an integral part of a cryptographic definition, alongside with other security properties. More substantial is the difference between *computational* and *statistical* properties, which associate different concrete interpretations to the term *negligible*. Correctness usually holds unconditionally, and it is therefore a statistical property. In particular, it is enough for correctness to achieve s-bits of security for some $s \leq c$. On the other hand, computational security properties require c-bits. In both cases, we can say that the properties achieves (c,s)-bits of computational/statistical security.

In choosing the security level (c,s) of an application, one should remember that even in a statistical setting, the adversary can increase its success probability by repeating the attack. However, this typically involves interaction with the user (by issuing many encryption challenge queries), and can be controlled by the application (e.g., by placing a bound on the number of encryptions before requiring a rekeying). This should not be confused with computational security, where the adversary can increase it success probability by investing computational resources locally, without further interaction with the user. Still, for applications making a large number of oracle calls, one should scale both security parameters (c,s) appropriately. We refer the reader to [39, 42] for more details.

2.2 Correctness properties

We introduce general notations 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), defined in Appendix A. We will use search games to model correctness properties, in which case we say that a scheme is $\mathcal{G}$-correct for a game $\mathcal{G}$.

We first describe (homomorphic) encryption syntax, using the notation from [38, 39]. 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 $C(x_1,\dots,x_k) \in \mathcal{L}$ 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 $x_1,\dots,x_k \in \mathcal{M}$, $C \in \mathcal{L}_k$ and security parameter κ,*

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

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.[3] 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:*

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

[3]Let $\mathsf{Enc}_{\mathsf{pk}}(x)$ be a pathological encryption scheme that, if the input message equals the public key, it 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 may depend on pk or if the circuit C may depend on pk or 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, it is (almost) never the case that the correctness property is satisfied, so any adversary will typically achieve advantage 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 [39]), 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:*

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

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 either case, the adversary advantage in the (approximate) correctness game is always assumed to be negligible.

2.3 Generic Security Definitions

The standard definition of secure encryption (not only homomorphic) against passive adversaries is 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:*

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

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

An enhanced definition, called $\mathsf{IND\text{-}CPA^D}$ with decryption oracles, was proposed in [38] to properly model the security of *approximate* homomorphic encryption schemes. We remark that the decryption oracle introduced by the $\mathsf{IND\text{-}CPA^D}$ definition impacts the adversary's advantage in the statistical parameter. Here, we describe the non-adaptive version of the definition, corresponding to the common application scenario where a dataset $(x_1,\dots,x_n)$ is encrypted at the outset, then a homomorphic computation is performed on it, and finally the result of the homomorphic computation is decrypted. 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.

Definition 6 ($\mathsf{IND\text{-}CPA^D}$ 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* $\mathcal{C}$. $\mathsf{IND\text{-}CPA^D}$ security *is defined by the following decision game:*

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

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 ciphertexts. 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.

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, but 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. For approximate schemes (and some exact ones, like the GSW cryptosystem [24] and its Ring LWE adaptation [18]), the size of the input messages can also have an impact on the correctness/security properties. 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 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 FHE 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 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 correspond to how homomorphic encryption schemes should be implemented and used in practice. Our definitions apply both to exact and approximate homomorphic schemes. We focus here on the simplest yet general type of computations, where the 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 include the adaptive definitions in Appendix B.

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 $\mathit{App} \subseteq \bar{\mathcal{L}}$ *is a set of* *computations that admits a compact description.*

We define an application App[4] 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. 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 $\mathbf{x} \in \mathsf{dom}(\bar{C})$). Then, $\mathsf{App} = \{\bar{C}\}$ is a singleton set, and can describe the application with a single circuit C and associated domain $\mathsf{dom}(\bar{C})$. We now define an application-aware homomorphic encryption scheme.

Definition 8. *An application-aware public-key homomorphic encryption scheme for application $\mathit{App} \subseteq \bar{\mathcal{L}}$ 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 $\mathit{App} \subseteq \bar{\mathcal{L}}$ 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.[5]

Definition 9 (Static 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* $\mathsf{Estimate} : 2^{\bar{\mathcal{L}}} \to \mathbb{R}_{\geq 0}$ *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:*

```
$\mathsf{Expr}^{\mathsf{approx},\widetilde{\mathcal{E}}}[\mathcal{A}](\kappa) : \mathit{App} \leftarrow \mathcal{A}(\kappa)$
    $(\mathit{sk}, \mathit{pk}) \leftarrow \mathsf{KeyGen}(\kappa, \mathit{App})$
    $\mathbf{x} \leftarrow \mathcal{A}(\mathit{pk})$
    $\mathit{ct}_i \leftarrow \mathsf{Enc}_{\mathit{pk}}(x_i)$ for $i = 1, \ldots, n$
    $\bar{C} \leftarrow \mathcal{A}(\mathit{ct}_1, \ldots, \mathit{ct}_n)$
    if $\bar{C} \notin \mathit{App}$ or $\mathbf{x} \notin \mathit{dom}(\bar{C})$ then abort
    $\mathit{ct}' \leftarrow \mathsf{Eval}_{\mathit{pk}}(\bar{C}, \boldsymbol{ct})$
    $y \leftarrow \mathsf{Dec}_{\mathit{sk}}(\mathit{ct}')$
    if $\|y - \bar{C}(\mathbf{x})\| > \mathsf{Estimate}(\bar{C})$
    then return 1 else return 0.
```

Definition 10 (Application-aware $\mathsf{IND\text{-}CPA^D}$ 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 $\mathsf{IND\text{-}CPA^D}$ security *is defined by the following decision game:*

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

The exact FHE correctness and $\mathsf{IND\text{-}CPA}$-security definitions (Definitions 3 and 5) can be also trivially extended to the application-aware model.

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. Here we detail 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. For (i), typically used for BGV, BFV and CKKS (and the main focus of our paper), the leveled computations desired to be evaluated should represent the application in the KeyGen algorithm, as described so far. For (iii), chiefly used for DM, CGGI and LMKCDEY, the application should be specified as gates with the bootstrapping procedure (and the 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

[4]A class of applications 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 $\mathbf{x} \in \mathcal{M}^k$ so $\|x_i\| \leq \mu$ for all i. A more complete description is provided in Section 4.

[5]Exact schemes correspond to setting $\mathsf{Estimate}(\bar{C}) = 0$, i.e., no approximation is allowed in the final result of the computation.

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/security properties) 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 show 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 Appendix C, we show that the $\mathsf{IND\text{-}CPA}$ and $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ security in the application-aware model hold as expected against valid adversaries.

4 Application Specification Languages

Since the interface of an application-aware FHE scheme requires the user to provide the KeyGen algorithm with an application specification, any library supporting the application-aware model requires a formal, well-specified mechanism to describe applications. In other words, we need an *application specification language (ASL)* which can be used to describe the application, readily maps to the operations supported by FHE schemes and can be easily parsed by the library to determine the scheme parameters.

In this section we describe an example language that can be used to describe simple, but representative applications, such as of the type used in the recent attacks described in Section 6. We demonstrate the feasibility of the approach by implementing the language in the OpenFHE library.

As in previous subsections, we consider applications $\mathsf{App} = \{\bar{C}\}$ consisting of a single function. In our sample ASL, a function operates on values which can be integers in the case of exact schemes and reals in approximate schemes (the effective precision will be determined by the scaling factor parameter), and is described simply by a sequence of operations $[1 : op_1, 2 : op_2, \ldots, n : op_n]$, each preceded by the corresponding line number, and possibly taking one or more parameters. Each operation can be one of the following:

- ℓ : $\mathsf{input}\ a_\ell\ b_\ell$. This instruction represents an input to the function, consisting of a value x_ℓ in the range $[a_\ell, b_\ell]$ of the appropriate data type. The parameters can be any two values in the message space of the encryption scheme, with $a_\ell \leq b_\ell$. The number k of input instructions in a program represents the number of inputs to the function C. We may assume without loss of generality that all the input instructions occur at the beginning of the program, i.e., they use line numbers $\ell = 1, \ldots, k$.
- ℓ : $\mathsf{add}\ i_\ell\ j_\ell$. Add up the values computed by the instructions at lines i_ℓ and j_ℓ. Here $i_\ell, j_\ell < \ell$ are two positive integers representing line numbers, and are required to be smaller than the current line ℓ. It is possible to use the same index $i_\ell = j_\ell$ to double a value.
- ℓ : $\mathsf{mul}\ i_\ell\ j_\ell$. Multiply the values computed by the instructions at lines $i_\ell, j_\ell < \ell$. It is possible to use the same index $i_\ell = j_\ell$ to square a value.

The program represents a circuit $C(x_1, \ldots, x_k)$ with k input wires (represented by the input instructions,) and $n - k$ addition and multiplication gates (represented by add and mul.) Each line number ℓ represents either an input or the output wire of a gate. The domain is given by the cartesian product $\prod_{\ell \in L}[a_\ell, b_\ell]$, where L is the set of all line numbers containing an input instruction. The program is evaluated on a list of inputs $x_1, \ldots, x_k$ in this domain in the obvious way, assigning a value to each line number: x_i for the input gates, and computing the sum or product of two previous lines i_ℓ, j_ℓ for the add and mul gates. The final output of the program is given by the last instruction, computing x_n. As a shorthand notation for parsing the $\mathsf{App} = \{\bar{C}\}$ description in ASL, we will also use $\bar{C} = \left[\mathsf{I}(C), \mathsf{dom}(\bar{C}), C, \mathsf{O}(C)\right]$, where $\mathsf{I}(C)$ and $\mathsf{O}(C)$ denote all inputs and all outputs of the circuit C. For some examples of programs written in this simple language, see Section 6.

This language describes the circuit in a manner that can be used by KeyGen and Eval to compute the noise estimates and evaluate the corresponding ciphertext circuit, respectively. In other words, this ASL can be used also to parse the scheme-specific mapping to a circuit over ciphertexts (both of these algorithms have to use the same mapping). Note that the inputs thus correspond to independent encryptions and each gate to the corresponding homomorphic operation.

The domain description is relevant for noise estimation in approximate schemes (and certain exact schemes, like GSW). We remark that using the range is only one possible example; one can specify a distribution instead or even provide concrete samples for the inputs. In this ASL example, used as a basis for the prototype implementation in OpenFHE, we chose the range as a middle ground between simplicity and effectiveness. Furthermore, this language can be extended in a number of ways, by including:

- An explicit $\mathsf{output}\, i_\ell$ instruction, to allow for programs that output more than one value. For h output instructions, C would be a function with output $(y_1, \ldots, y_h)$.
- More operations, as supported by the FHE scheme being implemented. These could be either specialized versions of add, mul, e.g., to double or square a number, or genuinely new operations.
- Explicit support for SIMD operations, where “wires”

carry a vector of values, on which additions and multiplications (or other operations) are performed in parallel.

- Additional operations on vectors, like the permutation (rotation) operations implemented using automorphisms and key-switching by some FHE schemes.
- A special unary function computation operation eval $T_\ell i_\ell$, where T_ℓ is a function on a small domain represented by a table, e.g., to represent "functional bootstrapping".
- Special identity functions for "no-operation" instruction $\ell :$ id i_ℓ, which simply copies a value from line i_ℓ to line ℓ.

While instructions of the last kind serve no useful purpose on the plaintext data, they can be used to represent, e.g., an invocation of the bootstrapping algorithm, giving more control to the expert user on where and how often bootstrapping is performed. Similarly, such instructions can be used to describe advanced optimizations such as lazy relinearization.

5 Practical Guidelines for Application-Aware Homomorphic Encryption

Recall that homomorphic encryption schemes are only *passively*-secure. 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 should make the use of FHE libraries less error-prone using the application-aware model formulated in our work. Instead of relying simply on the user expertise, FHE libraries can make the application specification App an explicit parameter of the key generation procedure, store App as part of the key/evaluation context, and then implement appropriate checks when the user makes calls to the encryption, evaluation and decryption functions. We remark that from a practical perspective, compilers are a promising solution to implement the application-aware model in FHE libraries. In the following, we provide a practical description of the secure application-aware FHE schemes, specify validators and instructions for the FHE libraries' users and developers.

5.1 Application-Aware Approximate FHE

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

Protocol 1 makes the presence of the validators explicit and provides guidelines for the correct usage of approximate FHE schemes in the IND-CPA$^{\mathsf{D}}$ setting with respect to an application class App. The transformation from [39] for IND-CPA$^{\mathsf{D}}$, described in Appendix C.2, uses a *mechanism* to define new KeyGen$'$ and Dec$'$ algorithms. In Protocol 1 we separate the derivation of the public parameters pp and noise estimates $\{t_i\}$ from the secret key sk sampling and public key pk computation in KeyGen$'$. 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 (sk is used to derive the evaluation keys and to perform the decryption, and is not used by the evaluator). This explicit split into 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; concretely for 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. Concretely, during evaluation, the library API should call a ValidateCircuit procedure, which, given an application (described in our ASL), determines if it is allowed, i.e., the gates of the evaluated circuit satisfy the application specification. Checking that the input values belong to the function domain is more challenging because the input to Eval is encrypted. Therefore, during encryption, the library API should call a ValidateEncryption procedure to ensure that the inputs are in the correct domain. This check may be hard to enforce in practice if the inputs are provided as different encryption calls, unless the domain $\mathsf{dom}(\bar{C}) = M^{|\mathsf{I}(C)|}$ restricts each message independently to the same set $M \subseteq \mathcal{M}$. To alleviate this, encryption should also be provided with an index ℓ for each input such that it can check the range as specified in App, i.e., check that an input m with index ℓ is in the range $[a_\ell, b_\ell]$ before encryption. The encryption could also tag the output ciphertext with the index ℓ, to indicate that it is the encryption of a valid input for position ℓ. Then, ValidateCircuit could also check that the input ciphertexts are properly tagged with the indexes $\ell = 1, \ldots, k$. Note that this also implies that the ciphertexts passed as input to Eval are independently computed, fresh encryptions of the input values, as required by our application-aware security definition. Finally, if an alternative run-time estimation is desired, the accumulated noise can be estimated during Enc and Eval, and a noise check can be performed during Dec using a ValidateDecryption procedure.

We reiterate that these validators are relevant in the passive security model, normally used by FHE, and rely on Eval being called with public key and input ciphertexts that have been honestly computed. (An active adversary could easily modify the tags of the ciphertexts output by Enc, add tags to ciphertexts output by Eval, or come up with invalid ciphertexts on its own, and go around all the checks performed by the library.) These checks are still useful to detect possible (honest) misuses that do not satisfy the application-aware model and that

Protocol 1 Application-Aware FHE scheme for App.

Offline Noise Estimation and Parameter Generation

```
Input: $\kappa, \mathsf{App}$ (in ASL).
Output: $\mathsf{pp}$.
1: Initialize $\mathsf{pp}$ for the given application $\mathsf{App}$ (using an optimistic value of lattice dimension). Parse $\mathsf{App} = \bigcup_i \bar{C}_i$. Each $\bar{C}_i$ is specified as $\mathsf{I}(C_i)$, $\mathsf{dom}(\bar{C}_i)$, $C_i$, and $\mathsf{O}(C_i)$.
2: Compute noise estimates $t$ using current $\mathsf{pp}$ on representative inputs for all computations $\bar{C}_i$, for each $\bar{C}_i \in \mathsf{App}$: $\{t_i \leftarrow \mathsf{Estimate}(\mathsf{App})\}_{i\in \mathsf{O}(C_i)}$.
3: Update $\mathsf{pp}$ based on current $t$.
4: If current $\mathsf{pp}$ do not satisfy $\kappa$, update $\mathsf{pp}$ (increase the lattice dimension) and go to Step 2.
```

Online Execution

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

[a]We remark that if parameters are properly set, 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.

can lead to a key recovery attack even by a passive adversary.

Application Specification. In approximate FHE, the application specification needs to include the description of supported computations as well as a compact description of the input messages, for instance, their range. The multiplicative depth is commonly used in guiding the parameter selection during the offline phase, but it may often be insufficient by itself as demonstrated by the attacks discussed in Section 6. When CKKS bootstrapping is used, one has to also check that the probability of decryption failure during bootstrapping is negligible (see [5] for more details) and to stipulate the bootstrapping procedure as part of the application specification. Our proposed ASL addresses the challenges related to the application specification for CKKS.

Current Library Limitations. The guidelines provided by libraries typically recommend running full computations (in the estimation mode) to obtain tight noise bounds and generate scheme parameters. Here we focus on OpenFHE and HElib as they both describe concrete guidelines [30, 47] for configuring specifically $\mathsf{IND\text{-}CPA^D}$-secure approximate homomorphic encryption. Both libraries follow the two-phase (first estimate on test data, then evaluate on actual encrypted data) approach and require running full computations (step-by-step procedures) during the offline estimation phase.

OpenFHE finds tight estimates during the offline phase 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 so only real inputs are supported). OpenFHE also implements the flooding noise estimation method proposed in [39] based on differential privacy. However, there is no check that the same circuit is used for both noise estimation and evaluation, which enabled the attacks in [25]. OpenFHE only takes the multiplicative depth, scaling factor, representative test data, and optionally ring dimension to describe the application. Then it relies on the user to implement the evaluation procedure and provide encrypted inputs to it. The procedure itself is not formalized and there is some ambiguity in how the inputs are fed to the procedure. In this work, we propose to use ASLs to eliminate the ambiguity and provide a unique way to describe the circuits and inputs during estimation and evaluation phases.

The guidelines provided by HElib [30] have similar limitations: HElib's noise estimation mechanism, which provides tight estimates for BGV, can become highly inaccurate for CKKS as soon as the message magnitudes significantly deviate from unity [4]. Hence, the noise estimation mechanism cannot be used to check that circuits provided during the evaluation phase match those used during the estimation phase.

Proof-of-concept implementation in OpenFHE. We proposed an ASL tailored to CKKS to check that compatible circuits are used during the estimation and execution phases. The ASL for CKKS requires a granular description of inputs so that the test data used during the estimation phase represent well the encrypted data supplied during the evaluation phase. Concretely, an example of such description is that for each input, the ASL supplies the range. These input-specific parameters are then used by the library to generate test data during the estimation phase (note that initializing the input vectors to the boundary values yields the worst-case bounds).

We provide a proof-of-concept implementation of this method in OpenFHE and demonstrate it for the case corresponding to the attacks in [25]. At a high level, our implementation adds SetEvalCircuit (for a single circuit) and SetEvalCircuits (if multiple circuits are supported) to parameter generation, ValidateCircuit to check that the circuit used for evaluation is compatible with the circuits set during parameter generation, and EvaluateCircuit to evaluate the full circuit after validating it. All these methods take a circuit definition (using the example ASL) as an input parameter.

Note that the ValidateEncryption and ValidateDecryption capabilities can also be added, but we focused specifically on EvaluateCircuit in our proof-of-concept implementation as the latter is sufficient to circumvent the attacks [25].

5.2 Application-Aware Exact FHE

Protocol 1 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. First, the goal of the protocol in the exact setting is to guarantee correct decryption with negligible probability of failure. The probability of failure is taken as a parameter in the key generation, in the form of the statistical security parameter. Second, for schemes such as BFV/BGV, 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.

Current Library Limitations. 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 (a similar procedure is used for BFV). Using these three input parameters, OpenFHE finds all scheme parameters via the procedure described in [32, Sec. 4], using analytical expressions. The probability of failure does not need to be 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 [26, Sec. 6]), resulting in the probability of decryption failure below 2^{-100}. Nevertheless, the current approach in OpenFHE (using only the mentioned three parameters) cannot uniquely describe all applications, which made the attacks in [10, 11] possible. Our proposed ASL addresses this problem. Moreover, no validator such as ValidateCircuit is currently used to determine that the parameters were generated for the same (or compatible) circuit that is being evaluated.

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 [27]) is employed to make decisions on when to invoke modulus switching (or bootstrapping), as well as to enforce the correctness of the decryption output. Using this tight noise estimation mechanism for BGV, HElib already provides the functionality corresponding to ValidateDecryption in Protocol 1, which gives it empirical protection against the attacks [10, 11].

Proof-of-concept implementation in OpenFHE. Instead of relying on the three parameters to describe BGV/BFV circuits, we add the support of ASL to OpenFHE. The new methods introduced to OpenFHE are similar to the CKKS case except for two major differences. First, we add EstimateCircuit to compute a tight estimate for a given circuit. This estimation functionality can be used to verify the compatibility of the circuits during the evaluation phase without explicitly specifying them during parameter generation (thus extending the functionality as compared to the CKKS case). Second, the input ranges do not need to be explicitly given (as they can be automatically derived from the plaintext modulus for most applications). We keep explicit input ranges in our proof-of-concept implementation for generality (as they could be potentially needed in scenarios with bootstrapping, where multiple plaintext moduli can be used). Our proof-of-concept implementation supports only BFV, but can also be extended to BGV by using a different noise estimation capability (already provided in OpenFHE).

While the safeguards described above are intended to protect against attacks exploiting invalid circuits, recent works have also identified vulnerabilities related to the incorrect setting of parameters with respect to the (c,s)-security. We give more details on the latter in Appendix E.3.

6 Discussion of Secret Key Recovery Attacks

We briefly summarize the Li-Micciancio key-recovery attack [38], as all attacks on CKKS, BGV and BFV from [10, 11, 25] 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.3), where the encoding and decoding are considered errorless (the attack can be extended to the efficient CKKS scheme used in practice [12, 31]). 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 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 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 $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-secure. Li et al. [39] further analyzed the $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ 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. However, they did not formally include the estimation procedure and its relation to the evaluated function class in the definition, something which needs to be done by libraries like OpenFHE and HElib that implement their security countermeasures.

In Section 3 and Appendix C, we clarified the $\mathsf{IND\text{-}CPA^D}$ definition in the practical context of user applications, and gave precise formulations of application-aware security statements in support of FHE libraries. Consequently, FHE libraries that implement approximate FHE schemes with the countermeasures proposed in [39], or instantiate exact FHE schemes with parameters that satisfy appropriate correctness bounds for specified circuits, satisfy the application-aware notion of $\mathsf{IND\text{-}CPA^D}$-security and should be immune to the Li-Micciancio attack [38] and its variants.

Recently, a number of works have extended the attack of [38] to either (i) defeat the security countermeasures for approximate FHE [25] or (ii) break exact FHE schemes [10, 11]. As in the original attack, these works use an $\mathsf{IND\text{-}CPA^D}$ 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 [10, 11, 25], this is achieved by exploiting queries to the evaluator or decryptor that are valid according to Definition 6, but *invalid* according to Definition 10. For approximate FHE, this bypasses the intended effect of the noise flooding mechanism. For exact schemes, this breaks the equivalence between $\mathsf{IND\text{-}CPA}$ and $\mathsf{IND\text{-}CPA^D}$-security. (Note that these attacks violate the assumptions of Theorems 1 and 2 in Appendix C.)

In this section, we describe the attacks [10, 11, 25] using our application-aware security definitions and the simple ASL from Section 4. This serves two purposes. One is to show that these attacks highlight the dangers associated to both currently insufficient library specifications and to using the libraries improperly, rather than a vulnerability in the schemes or in the implementations. 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 [10, 11, 25].

6.1 Attacks on Approximate FHE schemes

Guo et al. [25] proposed two attacks with the goal of injecting a smaller noise in the decryption procedure than required. This allows the attacker to retrieve sufficient information about the original noise in order to recover the secret key.

We now translate the attacks from [25] to the application-aware $\mathsf{IND\text{-}CPA^D}$ formalism from Section 3 using the ASL in Section 4. 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 $\mathsf{IND\text{-}CPA^D}$. (In Appendix E.1, we show the formulation under the adaptive definition as well, which was how it was described in [25].)

In the first attack (called "average-case estimation attack"), the attacker specifies $\mathsf{App} = \{\bar{C}\}$ 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. Using ASL, App would be specified as in (1). In the attack, the authors use messages equal to 0 in both estimation and evaluation; we use a non-degenerate range for generality. For $n = 1,000$, the noise estimate and ciphertext modulus are 13 and 72 bits, respectively.

The attacker then asks for the encryption of input x_1 and specifies the function $C'(x_1) = x_1 + \ldots + x_1$ for evaluation (keeping the same number of inputs as above, it could also be $C'(x_1, \ldots, x_n) = x_1 + \ldots + x_1$). Under ASL, this specification would look as in (2). Naturally, the parameters for $\bar{C}'$ are 18 and 77 bits, respectively, which are 5 bits ($\approx 0.5 \log n$) larger than the ones estimated for $\bar{C}$.

$$\begin{array}{rlll} 1: & \mathsf{input} & -1 & 1, \\ 2: & \mathsf{input} & -1 & 1, \\ & \vdots & & \\ n: & \mathsf{input} & -1 & 1, \\ n+1: & \mathsf{add} & 1 & 2, \\ n+2: & \mathsf{add} & n+1 & 3, \\ & \vdots & & \\ 2n-1: & \mathsf{add} & 2n-2 & n. \end{array} \tag{1}$$

$$\begin{array}{rlll} 1: & \mathsf{input} & -1 & 1, \\ 2: & \mathsf{add} & 1 & 1, \\ 3: & \mathsf{add} & 2 & 1, \\ & \vdots & & \\ n+1: & \mathsf{add} & n & 1. \end{array} \tag{2}$$

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. An implementation of $\mathsf{ValidateCircuit}$ would disallow the evaluation of $\bar{C}'$. The same holds for computing the addition recursively, via doubling, also explored in [25], that can be specified as

$$\begin{array}{rlll} 1: & \mathsf{input} & -1 & 1, \\ 2: & \mathsf{add} & 1 & 1, \\ 3: & \mathsf{add} & 2 & 2, \\ & \vdots & & \\ \log n+1: & \mathsf{add} & \log n & \log n. \end{array} \tag{3}$$

In the case of the second attack (called "empirical noise attack"), the attacker specifies for the run-time evaluation the circuit $C''(x_1, \ldots, x_{n'}) = x_1 + \ldots + x_{n'}$, for $n' \neq n$ (in ASL, the specification looks like (1) but with n replaced by n'), while still using $\mathsf{App} = \{\bar{C}\}$ defined in (1). But $\bar{C}'' \neq \bar{C}$ and $\bar{C}'' \notin \mathsf{App}$, rendering this query invalid according to Definition 10.

The authors of [25] 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 better to include $\bar{C}'$ in App at key generation time, and let the library choose the parameters accordingly, as done in our proof-of-concept implementation. 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 $\mathsf{IND\text{-}CPA^D}$-security. This is true even if the differentially-private mechanism applied in decryption uses worst-case noise bounds over $\bar{C}$ and $\bar{C}'$.

Instead, for application-aware $\mathsf{IND\text{-}CPA^D}$-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 $\mathsf{IND\text{-}CPA^D}$-security. Importantly, this also allows the use of non-worst-case noise bound estimation, and refutes the claim from [25] that any usage of non-worst-case estimates is insecure, as long as the estimation is performed globally over the class of allowed computations.

Finally, from the perspective of Section 5, these attacks violate Protocol 1, 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 application specifications, and validate the use of the same computation during offline and online phases.

6.2 Attacks on Exact FHE schemes

Exact FHE schemes (Definition 3) are a special case of approximate FHE schemes with a perfect estimation function $\mathsf{Estimate}(\mathsf{App}) = 0$ (no approximation error). In the case of decryption failures, these schemes can still deviate from recovering the exact message, enabling the Li-Micciancio attack [38]. 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 may 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 describe such an attack [3] in Appendix E.2. What makes this attack possible is allowing for as many additions (whose number depends on the ciphertext modulus) as to lead to an incorrect decryption result. 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. Then, during run-time, only the evaluation of this computation is allowed, which returns correct decryptions with high probability.

Recently, Checri et al. [10] and Cheon et al. [11] 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, corresponding to (1), which returns parameters for achieving exact correctness for App–and then using these parameters to perform a different computation $\bar{C}' \notin \mathsf{App}$, which can be specified as in (3). This computation $\bar{C}'$ is chosen such that for $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_i)$, $i = 1, \ldots, n$, it holds that $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{pk}}(\bar{C}', \mathsf{ct}_1, \ldots, \mathsf{ct}_n)) \neq \bar{C}(x_1, \ldots, x_n)$. Concretely, for the attack in [11] with plaintext modulus $p = 65,537$, ring dimension $N = 8,192$ and 44 doubling operations, the circuit $\bar{C}$ specified in OpenFHE, corresponding to (1), leads to a ciphertext modulus of 37 bits and a noise estimate of 19 bits, whereas the implemented circuit, corresponding to (3), leads to a required ciphertext modulus of 75 bits and a noise estimate of 57 bits. But since $\bar{C} \notin \mathsf{App}$, neither application-aware correctness nor $\mathsf{IND\text{-}CPA^D}$-security is guaranteed. The attacks in [10, 11] show that this lack of security is not just a (well-known, but theoretical) possibility, but a real threat in practice. We defer more details to Appendix E.2.

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. 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 $\mathsf{IND\text{-}CPA^D}$ 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 $\mathsf{IND\text{-}CPA^D}$ security in the application-aware setting. We also demonstrate that our application-aware model can be used to securely instantiate exact FHE schemes.

This work is a first step in establishing the practical procedures for the secure, efficient use of FHE in the $\mathsf{IND\text{-}CPA^D}$ setting. In the future, more expressive/compact application specification languages could be developed. Improving the implementation of automated validators for testing that computations are in the allowed application class is also desirable.

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 the underlying complicated details. 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 [21], and are still actively being researched. Legal auditing and other non-cryptographic approaches could offer valuable complementary measures.

References

[1] Akavia, A., Gentry, C., Halevi, S., and Vald, M. Achievable CCA2 relaxation for homomorphic encryption. In *TCC 2022, Part II* (Nov. 2022), E. Kiltz and V. Vaikuntanathan, Eds., vol. 13748 of *LNCS*, Springer, Cham, pp. 70–99.

[2] 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, Berlin, Heidelberg, pp. 483–501.

[3] 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.

[4] Bergamaschi, F., Halevi, S., Halevi, T. T., and Hunt, H. Homomorphic training of 30,000 logistic regression models. In *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings* (Berlin, Heidelberg, 2019), Springer-Verlag, p. 592–611.

[5] Bossuat, J.-P., Mouchet, C., Troncoso-Pastoriza, J. R., and Hubaux, J.-P. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *EUROCRYPT 2021, Part I* (Oct. 2021), A. Canteaut and F.-X. Standaert, Eds., vol. 12696 of *LNCS*, Springer, Cham, pp. 587–617.

[6] 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, Berlin, Heidelberg, pp. 62–89.

[7] 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, Berlin, Heidelberg, pp. 868–886.

[8] 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.

[9] Canard, S., Fontaine, C., Phan, D. H., Pointcheval, D., Renard, M., and Sirdey, R. Relations among new CCA security notions for approximate FHE. Cryptology ePrint Archive, Report 2024/812, 2024.

[10] Checri, M., Sirdey, R., Boudguiga, A., and Bultel, J.-P. On the practical CPA^D security of "exact" and threshold FHE schemes and libraries. In *CRYPTO 2024, Part III* (Aug. 2024), L. Reyzin and D. Stebila, Eds., vol. 14922 of *LNCS*, Springer, Cham, pp. 3–33.

[11] Cheon, J. H., Choe, H., Passelègue, A., Stehlé, D., and Suvanto, E. Attacks against the IND-CPA^D security of exact FHE schemes. In *ACM CCS 2024* (Oct. 2024), B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds., ACM Press, pp. 2505–2519.

[12] 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, Cham, pp. 409–437.

[13] 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, Cham, pp. 377–408.

[14] Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

[15] 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.

[16] 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, Cham, pp. 29–53.

[17] 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, Cham, pp. 103–115.

[18] 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, Berlin, Heidelberg, pp. 617–640.

[19] 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, Berlin, Heidelberg, pp. 294–310.

[20] Fan, J., and Vercauteren, F. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[21] Frankle, J., Park, S., Shaar, D., Goldwasser, S., and Weitzner, D. J. Practical accountability of secret processes. In *USENIX Security 2018* (Aug. 2018), W. Enck and A. P. Felt, Eds., USENIX Association, pp. 657–674.

[22] Gentry, C. *A fully homomorphic encryption scheme*. Stanford university, 2009.

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

[24] 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.

[25] 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* (Aug. 2024), D. Balzarotti and W. Xu, Eds., USENIX Association.

[26] Halevi, S., Polyakov, Y., and Shoup, V. An improved RNS variant of the BFV homomorphic encryption scheme. In *CT-RSA 2019* (Mar. 2019), M. Matsui, Ed., vol. 11405 of *LNCS*, Springer, Cham, pp. 83–105.

[27] Halevi, S., and Shoup, V. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481, 2020.

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

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

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

[31] 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, Cham, pp. 120–144.

[32] Kim, A., Polyakov, Y., and Zucca, V. Revisiting homomorphic encryption schemes for finite fields. In *ASIACRYPT 2021, Part III* (Dec. 2021), M. Tibouchi and H. Wang, Eds., vol. 13092 of *LNCS*, Springer, Cham, pp. 608–639.

[33] Kluczniak, K. Circuit privacy for FHEW/TFHE-style fully homomorphic encryption in practice. Cryptology ePrint Archive, Report 2022/1459, 2022.

[34] Kluczniak, K., and Santato, G. On circuit private, multikey and threshold approximate homomorphic encryption. Cryptology ePrint Archive, Report 2023/301, 2023.

[35] KNABENHANS, C. Practical integrity protection for private computations. Master's thesis, ETH Zurich, 2022.

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

[37] 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, Cham, pp. 227–256.

[38] 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, Cham, pp. 648–677.

[39] 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, Cham, pp. 560–589.

[40] MANULIS, M., AND NGUYEN, J. Fully homomorphic encryption beyond IND-CCA1 security: Integrity through verifiability. In *EUROCRYPT 2024, Part II* (May 2024), M. Joye and G. Leander, Eds., vol. 14652 of *LNCS*, Springer, Cham, pp. 63–93.

[41] 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, Cham, pp. 331–349.

[42] MICCIANCIO, D., AND SCHULTZ-WU, M. Bit security: Optimal adversaries, equivalence results, and a toolbox for computational-statistical security analysis. In *TCC 2024, Part II* (Dec. 2024), E. Boyle and M. Mahmoody, Eds., vol. 15365 of *LNCS*, Springer, Cham, pp. 224–254.

[43] MICCIANCIO, D., AND WALTER, M. On the bit security of cryptographic primitives. In *EUROCRYPT 2018, Part I* (Apr. / May 2018), J. B. Nielsen and V. Rijmen, Eds., vol. 10820 of *LNCS*, Springer, Cham, pp. 3–28.

[44] MURPHY, S., AND PLAYER, R. A central limit framework for ring-LWE decryption. Cryptology ePrint Archive, Report 2019/452, 2019.

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

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

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

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

[49] VIAND, A., KNABENHANS, C., AND HITHNAWI, A. Verifiable fully homomorphic encryption. *arXiv preprint arXiv:2301.07041* (2023).

[50] 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 More preliminaries

A.1 Security Games

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 $\mathsf{Adv}_{\mathcal{G}}^{\mathcal{S}}[\mathcal{A}](\kappa)$ of $\mathcal{A}$ in breaking the $\mathcal{G}$-security of $\mathcal{S}$ is*

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

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

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

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 $\mathsf{Adv}_{\mathcal{G}}^{\mathcal{S}}[\mathcal{A}](\kappa)$ 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 Differential Privacy

Definition 13 (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 14 (Norm KLDP [39])**.** *For $t \in \mathbb{R}_{\geq 0}$, let $M_t : B \to C$ be a family of randomized algorithms, where B is a normed space with norm $\|\cdot\| : B \to \mathbb{R}_{\geq 0}$. 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 $\|x - x'\| \leq t$, it holds:*

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

Definition 15 (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.3 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 $(R_Q^2)^m$.

B 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 16. *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* $\mathit{App} \subseteq \mathsf{Ops}^*$ *of computation traces that is closed under prefixes, i.e., such that if* $[\mathsf{op}_1, \ldots, \mathsf{op}_n] \in \mathit{App}$*, 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* $\mathit{App}' = \{\bar{C}_1, \bar{C}_2, \ldots\}$ *introduced in Definition 8 correspond to a special case of Definition 16, 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* $(x_1, \ldots, x_k) \in \mathit{dom}(\bar{C}_i)$ *for some* $\bar{C}_i \in \mathit{App}'$*. 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 17 (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* $\mathsf{Estimate} : 2^{\bar{\mathcal{L}}} \to \mathbb{R}_{\ge 0}$ *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:*

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

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

- *If the sequence of queries issued so far* $[\mathsf{op}_1, \ldots, \mathsf{op}_i] \notin \mathit{App}$*, then abort the experiment with output* 0
- *if* $\mathsf{op}_i = \mathsf{E}(x_i)$*, then let* $\mathit{ct}_i \leftarrow \mathsf{Enc}_{\mathit{pk}}(x_i)$ *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* $\mathit{ct}_i \leftarrow \mathsf{Eval}_{\mathit{pk}}(f_i, \mathit{ct}_1, \ldots, \mathit{ct}_k)$ *using previously stored pairs* $(x_{i_j}, \mathit{ct}_{i_j})$*. 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* $\|y_i - x_j\| \leq \mathsf{Estimate}(App)$ *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 17 when restricted to the simple applications App' described in Remark 2.

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

Definition 18 ($\mathsf{IND\text{-}CPA^D}$ 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* $\mathsf{Estimate} : 2^{\bar{\mathcal{L}}} \to \mathbb{R}_{\geq 0}$ *be an efficiently computable function.* Application-aware $\mathsf{IND\text{-}CPA^D}$ 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 17.* 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* $\mathcal{A}$.

Function-privacy. While the $\mathsf{IND\text{-}CPA^D}$ 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 $\mathsf{IND\text{-}CPA^D}$ 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.

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

We now adapt the results of [38, 39] to the application-aware model.

C.1 Exact Schemes

The equivalence between IND-CPA and $\mathsf{IND\text{-}CPA^D}$ security for exact FHE schemes can be extended from its generic formulation [38, 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*[6] *application-aware exact homomorphic scheme for application* $App \subseteq \bar{\mathcal{L}}$. $\mathcal{E}$ *is IND-CPA-secure if and only if it is* $\mathsf{IND\text{-}CPA^D}$*-secure.*

Proof. First, application-aware $\mathsf{IND\text{-}CPA^D}$-security also 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 $\mathsf{Enc}_{\mathsf{pk}}$ call, and no other $\mathsf{Eval}_{\mathsf{pk}}$ or $\mathsf{Dec}_{\mathsf{sk}}$ 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 $1 \leq i \leq n$, 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.

[6] 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*.

[7] Technically, for the adversary to issue no evaluation and decryption calls one needs to use the fully adaptive Definition 18. 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.

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 $\mathcal{A}$:
$$\mathsf{ct}_j \leftarrow \begin{cases} \mathsf{Enc}_{\mathsf{pk}}(x_1), & \text{if } j < i \\ \mathsf{Enc}_{\mathsf{pk}}(x_0), & \text{if } j > i \\ \mathsf{Expr}_b^{\mathsf{cpa}}[\mathcal{B}^{(i)}], & \text{if } j = i. \end{cases}$$

- For the query $\bar{C}$, it lets $\mathsf{ct}' \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\bar{C}, \mathbf{ct})$ 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 $\mathcal{H}^{(i)} = \mathsf{Expr}_0^{\mathsf{cpa}}[\mathcal{B}^{(i)}]$ for $1 \le i \le n$ and $\mathcal{H}^{(n+1)} = \mathsf{Expr}_1^{\mathsf{cpa}}[\mathcal{B}^{(n)}]$. Note that by construction, $\mathcal{H}^{(i)} = \mathsf{Expr}_1^{\mathsf{cpa}}[\mathcal{B}^{(i-1)}]$ 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 $\mathsf{Expr}_b^{\mathsf{cpad}}[\mathcal{A}]$. This leads to having indistinguishability between $\mathcal{H}^{(1)}$ and $\mathsf{Expr}_0^{\mathsf{cpad}}[\mathcal{A}]$ and between $\mathcal{H}^{(n+1)}$ and $\mathsf{Expr}_1^{\mathsf{cpad}}[\mathcal{A}]$. Therefore, using a union bound over the hybrid distributions gives that the advantage of $\mathcal{A}$ in the IND-CPA$^{\mathsf{D}}$ 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 IND-CPA$^{\mathsf{D}}$ game for App is also negligible. □

C.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 an 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 [39], described in Algorithm 2, 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 [39] by the notion of ρ-KLDP (Kullback-Leibler Differential Privacy, see Appendix A.2), for a sufficiently small value of ρ.

Algorithm 2 Application-aware $M[\widetilde{\mathcal{E}}]$ for App.

```
KeyGen'(κ, 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 $(\mathsf{sk}', \mathsf{pk})$
Dec'_{sk'}(ct) :=
1: return $M_t(\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}))$
```

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* $\bar{L}$. *Let* $\mathsf{Estimate} : 2^{\bar{L}} \to \mathbb{R}_{\ge 0}$ *be an efficiently computable function such that* $\widetilde{\mathcal{E}} = (\mathcal{E}, \mathsf{Estimate})$ *be application-aware statically approximate. Let* M_t *be a* ρ*−KLDP mechanism on* $\widetilde{\mathcal{M}}$*, where* $\rho \le 2^{-\kappa-7}$*. 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.*

Proof. The proof follows the same steps as the proof in Theorem 2 in [39], using similar modifications for the application-aware non-adaptive case as in the proof of Theorem 1. □

Li et al. [39] illustrated how to use the notion of bit-security [42, 43], described in Section 2.1, to achieve IND-CPA$^{\mathsf{D}}$-security with a lower amount of DP noise than in Theorem 2. 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 $s = 64$ (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 [12, 31]. Practically, M_t from Theorem 2 is instantiated via a discrete Gaussian mechanism (Definition 15 in Appendix A.2). Specifically, in Algorithm 2 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 2 as $M[\widetilde{\mathsf{CKKS}}]_\sigma$. Using the bit-security notion, one can obtain the following result for an $\mathsf{IND\text{-}CPA^D}$-secure instantiation, adapted from [39], 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 $\mathsf{IND\text{-}CPA}$-secure, then, for $\sigma = \sqrt{12} \cdot 2^{s/2}$, $M[\widetilde{\mathsf{CKKS}}]_\sigma$ is (c, s)-bit application-aware $\mathsf{IND\text{-}CPA^D}$-secure.*

Proof. The proof follows the proof of Corollary 2 in [39], with the correction mentioned in [47]. □

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 [39].

D Comparison with [35]

In [35], the notation $\mathcal{F}$-$\mathsf{IND\text{-}CPA^D}$ is used to capture the restriction of the evaluation function to a class of circuits $\mathcal{F}$. However, in [35, Theorem 6.5], the set $\mathcal{F}$ is a global parameter of the FHE scheme, and it is not part of the syntax of the key generation algorithm. Moreover, the function set $\mathcal{F}$ does not explicitly capture any restriction on the encrypted input values. Another difference between [35] and our work is that [35] defines $\mathcal{F}$ as a set of functions mapping ciphertexts to ciphertexts, with the intent to capture ciphertext management operations (such as bootstrapping, key switching, etc.), which cannot be described as pure functions of plaintext data. Using functions $F\colon \mathcal{C}^k \to \mathcal{C}$ that directly work on ciphertexts is problematic for a number of reasons: to start with, the set $\mathcal{F}$ only makes sense in the context of a specific encryption scheme, and one cannot use it to capture restricted functionalities, such as additive homomorphic encryption, in an abstract way. Perhaps more importantly, an arbitrary function $F\colon \mathcal{C}^k \to \mathcal{C}$ cannot be mapped to a well-defined function $f(m) = \mathsf{Dec}(F(\mathsf{Enc}(m))$ on messages, because Enc is randomized, and the same message may be mapped (homomorphically) to the encryption of different messages under F. On the other hand, our proposed ASL easily maps to (well-defined) functions on messages, and is also designed to include additional information, such as directives to the evaluation function on where to apply bootstrapping. However, this remains abstract, and how it maps to ciphertext is left to the definition of the Eval function which is part of the FHE scheme instance, rather than the abstract definition of FHE syntax and security.

E More on Section 6

E.1 Further comments on attacks in [25]

In [25], 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 18). The attack specifies the same circuit $C(x_1, \ldots, x_n) = x_1 + \ldots + x_n$ in the estimation and runtime 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 18.

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 $\{\mathsf{E}(x_1), \mathsf{H}(\bar{C}, 1, 1, \ldots, 1), \mathsf{D}(n+1)\}$ (or $\{\mathsf{E}(x_1), \ldots, \mathsf{E}(x_n), \mathsf{H}(\bar{C}, 1, 1, \ldots, 1), \mathsf{D}(n+1)\}$) which is not allowed in the application-aware model, since it has a different computation trace.

Another claim from [25] 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 $\mathsf{IND\text{-}CPA^D}$, 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.

E.2 Further comments on attacks in [10, 11]

Folkore attack on BFV/BGV. We briefly describe a folklore attack on schemes such as BGV/BFV where decryption is allowed despite an overflown ciphertext error. This discussion [3] happened following the responsible disclosure of the attack in [38].

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 the folklore 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, Q (and p) is specified first, and the number of additions required depends on this value. 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 is allowed, which with high probability, disallows adding a value for $p^{-1} - 1 \bmod Q$ times.

Attacks on OpenFHE. In particular, the attacks in [10, 11] are a good example of the risks of not following Protocol 1. The attacks 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 uses worst-case estimation), but because the circuit to be evaluated is not specified correctly. OpenFHE does not support a sufficiently granular application specification–see Section 5.2–in order to directly specify $\bar{C}'$ (in (2), (3)), for the doubling attack from [11], but only allowed specifications of circuits such as $\bar{C}$ (in (1)). The OpenFHE use of BGV/BFV for this scenario that would have prevented the attack would require the user to supply the number of additions of independent inputs (so 2^{44} instead of 44) before generating the parameters using SETEVALADDCOUNT, or an equivalent multiplicative depth using SETMULTIPLICATIVEDEPTH. The same could have been done for the attack in [10] where the circuit is purely made of addition gates. Additionally, to reduce the number of additions, [10] 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. Regardless, the correct way to prevent these attacks is to have the libraries support a sufficiently descriptive application specification language, such as what we proposed in Section 4, as to avoid any confusion on how to specify the circuits on which the parameter sets are being computed for.

E.3 Attacks related to FHEW/TFHE schemes

The works [10, 11] also describe attacks against the schemes implemented in the TFHE-rs [50] and TFHELib [14] libraries, which fall in a different category, related to the incorrect setting of parameters with respect to the (c,s)-security.

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 [10, 11] 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 [10] 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 [14] does not automatically 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 and using the proposed ASL to describe and validate the circuits would disallow the attack. 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 [11] also 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 [10, 11] to trigger decryption errors and mount a key recovery attack.

Concretely, with respect to the (c,s)-security, an incorrectly set probability of failure for the *whole* circuit can lead to key recovery attacks. The probability of decryption failure for a single bootstrapping operation can be exposed as an input parameter for certain schemes where it has a major impact on the efficiency, e.g., DM or CGGI. 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 below. OpenFHE provides a parameter generation tool [46] for DM, CGGI, and LMKCDEY that takes the bootstrapping probability of failure as an input argument. Finally, the number of evaluated gates (under a single key), which can be extracted using the ASL, 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.