On the practical CPA^D security of "exact" and threshold FHE schemes and libraries

Marina Checri, Renaud Sirdey, Aymen Boudguiga, Jean-Paul Bultel

Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France,
name.surname@cea.fr

Abstract. In their 2021 seminal paper, Li and Micciancio [17] presented a passive attack against the CKKS approximate FHE scheme and introduced the notion of CPA^D security. The current status quo is that this line of attacks does not apply to "exact" FHE. In this paper, we challenge this status quo by exhibiting a CPA^D key recovery attack on the linearly homomorphic Regev cryptosystem, which easily generalizes to other xHE schemes such as BFV, BGV and TFHE, showing that these cryptosystems are not CPA^D secure in their basic form. We also show that existing threshold variants of BFV, BGV and CKKS are particularly exposed to CPA^D attackers and would be CPA^D-insecure without proper smudging noise addition after partial decryption. Finally, we successfully implement our attack against several mainstream FHE libraries and discuss a number of natural countermeasures as well as their consequences in terms of FHE practice, security and efficiency. The attack itself is quite practical as it typically takes less than an hour on an average laptop PC, requiring a few thousand ciphertexts as well as up to around a million evaluations/decryptions, to perform a full key recovery.

1 Introduction

Since its inception more than ten years ago, Fully Homomorphic Encryption has been the subject of a lot of research towards more efficiency and better practicality. From a security perspective, however, FHE still raises a number of questions and challenges. In particular, all the FHE usable in practice, BFV[7,15], BGV[8], CKKS[10] and TFHE[11], achieve only CPA-security. Although it is well-known that malleability is contradictory with CCA2 security, building efficient FHE constructions achieving some degree of CCA security (e.g. CCA1) remains a very important open challenge. Being stuck at the CPA level, FHE is thus completely insecure (and trivially so) as soon as the adversary is granted access to a decryption oracle. In this context, Li and Micciancio [17] were the first to study the security of FHE against a slight, seemingly benign extension of CPA security where the adversary is granted access only to a highly constrained decryption oracle which accepts only genuine ciphertexts or ciphertexts derived from genuine ciphertexts by means of genuine homomorphic operations. The intuition is that, given a FHE scheme $S = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, if the adversary knows

m, f as well as $c = \mathsf{Enc}(m)$, granting her access to $\mathsf{Dec}(\mathsf{Eval}(f, c))$ should not raise any issue since she can compute $f(m)$ by herself and, by definition of FHE,

$$\mathsf{Dec}(\mathsf{Eval}(f, \mathsf{Enc}(m))) = f(m) \tag{1}$$

is supposed to hold for all m in the plaintext domain of S. At first glance, it appears that this constrained oracle does not provide more information to the adversary than she can compute on herself and, as such, that this CPA^D security is implied by or even equivalent to CPA security. Unfortunately, Li and Micciancio demonstrated that these intuitions are not true for approximate FHE schemes such as CKKS, for which it turns out that neither $\mathsf{Dec}(\mathsf{Enc}(m)) = m$ nor (1) hold (with high probability) and where the differences $\mathsf{Dec}(\mathsf{Enc}(m)) - m$ or

$$\mathsf{Dec}(\mathsf{Eval}(f, \mathsf{Enc}(m))) - f(m)$$

leak the LWE noises in the ciphertexts, resulting in the ability for the adversary to easily and practically recover the secret decryption key of the scheme. They further demonstrated the attack practicality on most mainstream libraries implementing CKKS. To the best of our knowledge, the current consensus in the state-of-the-art is that this line of attack does not apply to the other schemes such as BFV, BGV or TFHE which are "marketed" as non-approximate. In fact, as we shall later see, "non-approximate" does not mean "exact", and these schemes, at least in their basic forms, are practically vulnerable to CPA^D adversaries.

1.1 Contributions summary

This paper contributions are as follows:

- We start by exhibiting a key recovery attack on the Regev scheme and its simple RLWE variant in the CPA^D model. We also experimentally demonstrate that this attack is practical on several state-of-the-art LWE parameter sets achieving better than 128 bits security. In a nutshell, our attack consists in carefully controlling the noise growth in some encryptions of 0, obtained by repeatedly summing ciphertexts with themselves, in order to probe the threshold at which decryption errors start occurring. This eventually allows us to recover the absolute value of the LWE noise present in some ciphertexts, with a practically large enough probability. We then extend our attack toolbox with a procedure to test whether two ciphertexts for which the noise absolute values have been determined, have noises with the same sign. Using this toolbox, we can eventually generate two linear systems of equations in $\mathbb{Z}_q$ with one of the two giving the secret decryption key. In doing the attack, we perform only valid requests with respect to the CPA^D security game, which is formally recalled.
- We then port our attack to the BFV, BGV and TFHE schemes and successfully apply it to several of the mainstream libraries which implement these schemes, namely SEAL/BFV, SEAL/BGV, OpenFHE/BFV, OpenFHE/BGV,

Lattigo/BFV including in the threshold setting, and TFHElib. In all these cases, we successfully perform a full key recovery in a few minutes to a bit more than an hour, except for TFHElib for which the attack is much faster due to the smaller parameters usually necessary for TFHE to achieve 128 bits security. Remarkably, HElib/BGV resisted our attack due to its built-in noise upper bound tracking feature and, most importantly, the fact that its decryption function generates no output when the noise bound of a ciphertext exceeds a certain threshold.

- In addition, we take a closer look at multiparty threshold FHE setups for BFV and BGV, where encryption is performed relative to a global public key and decryption is done collaboratively by entities which possess only shares of a global secret decryption key which has to remain unknown to them. We show that this setup is intrinsically exposed to CPA^D adversaries, allowing any decrypting party to retrieve the global secret key unless either the underlying FHE scheme achieves CPA^D security or a countermeasure against CPA^D attacks is included. We then argue that the noise smudging technique with proper λ-independent variance, necessary in the security proofs for threshold constructions based on BFV or BGV, acts as a natural countermeasure against such CPA^D attacks.
- We discuss several natural and effective countermeasures applicable to different schemes (BFV/BGV vs TFHE), which essentially consist in strategies to ensure that a CPA^D adversary only has $\mathrm{neg}(\lambda)$ probability of observing exploitable decryption errors. Additionally, we investigate the impact that these countermeasures have on the choice of parameters for the underlying FHE scheme. As they generally require using "larger" parameters, e.g. larger ciphertext-modulus (which may also imply using larger dimensions) for BFV/BGV or larger dimensions and/or tighter decomposition basis and precisions for TFHE, we also experimentally illustrate the performance impact induced on the homomorphic calculations when these countermeasures are used to achieve CPA^D security.
- As a last contribution, we analyze the consequence of the relationship between CPA^D security and threshold FHE constructions when the CKKS scheme is used as the underlying FHE scheme. In particular, we show that in existing constructions, either the CPA^D insecurity of CKKS leads to an insecure threshold scheme or that decrypted results have to be so flooded with noise that their precision is jeopardized. This significantly reduces CKKS's attractiveness for use in the threshold setting.

1.2 Paper organization

This paper is organised as follows: we first recall the CPA^D security game in Sect. 2. We then detail in Sect. 3 the principles of our attack starting from vanilla Regev and its RLWE variant. Sect. 4 then adapts the attack to BFV, BGV and TFHE and provides experimental results showing the attack practicality on some of the mainstream libraries implementing them. Lastly, Sect. 5 discusses the implications of our attack on existing threshold variants of BFV/BGV. We

conclude the paper by a discussion on countermeasures and their implications in terms of FHE efficiency (Sect. 6).

2 Background on CPA^D security

In this section, we recall the CPA^D security game [17]. It is a left-or-right game in which an adversary attempts to recover a random bit with a significant advantage over guessing.

Given an encryption scheme

$$\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}),$$

with plaintext domain $\mathcal{P}$ and ciphertext domain $\mathcal{C}$, an adversary $\mathcal{A}$ and value λ for the security parameter, the game is parameterized by a bit $b^* \xleftarrow{\$} \{0,1\}$, unknown to $\mathcal{A}$, and an initially empty state S of message-message-ciphertext triplets:

- Key generation: Run $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and give ek to $\mathcal{A}$. Note that this security game works identically in the non-public key setting, only ek is not revealed to $\mathcal{A}$.
- Encryption request: When $\mathcal{A}$ queries $(\texttt{test\_messages}, m_0, m_1)$, where $m_0, m_1 \in \mathcal{P}$, compute $c = \mathsf{Enc}_{\mathsf{ek}}(m_{b^*})$, give c to $\mathcal{A}$ and do

$$S := [S; (m_0, m_1, c)].$$

- Evaluation request: When $\mathcal{A}$ queries $(\texttt{eval}, f, l_1, \ldots, l_K)$, with K the arity of function f and $\forall i : l_i \in [\![1, |S|]\!]$, compute

$$m'_0 = f(S[l_1].m_0, \ldots, S[l_K].m_0),$$

and

$$m'_1 = f(S[l_1].m_1, \ldots, S[l_K].m_1),$$

as well as

$$c' = \mathsf{Eval}(f, S[l_1].c, \ldots, S[l_K].c),$$

where $S[l_i].e$ is the element $e \in \{m_0, m_1, c\}$ of the state S for the l_i-th request. Then, we update S with the $|S|+1$ entry (m'_0, m'_1, c') as follows:

$$S := [S; (m'_0, m'_1, c')]$$

- Decryption request: When $\mathcal{A}$ queries $(\texttt{ciphertext}, l)$ $(l \in [\![1, |S|]\!])$ proceed as follows: if $S[l].m_0 \neq S[l].m_1$ then return $\perp$ to $\mathcal{A}$, otherwise return her $\mathsf{Dec}_{\mathsf{dk}}(S[l].c)$.
- Guessing stage : After polynomially many interleaved encryption, evaluation and decryption requests, $\mathcal{A}$ outputs $(\texttt{guess}, b)$. The outcome of the game is determined as follows: if $b = b^*$ then $\mathcal{A}$ wins the game. Otherwise, $\mathcal{A}$ looses the game.

A number of points should be emphasized with respect to the above game. First, the decryption oracle accepts only ciphertexts from the game state which are necessarily well-formed. That is, it accepts only ciphertexts either produced by an encryption oracle via an encryption request or derived by the evaluation oracle via an evaluation request, so derived by correctly applying homomorphic operators to well-formed ciphertexts. As such, the above game does not capture any CCA aspects. Second, when $S[l].m_0 = S[l].m_1$ it is important that the decryption oracle returns $\mathsf{Dec}_{\mathsf{dk}}(S[l].c)$ and not $S[l].m_0$ or $S[l].m_1$. For exact FHE, this has no impact since $\mathsf{Dec}_{\mathsf{dk}}(S[l].c) = S[l].m_0 = S[l].m_1$. And in this case, $\mathcal{A}$ learns nothing she does not already knows, so CPA^D security coincide with CPA security. However, for approximate or non-exact FHE, even when $S[l].m_0 = S[l].m_1$, we may have that $\mathsf{Dec}_{\mathsf{dk}}(S[l].c) \neq S[l].m_0$ and $\mathsf{Dec}_{\mathsf{dk}}(S[l].c) \neq S[l].m_1$. Thus, for approximate or non-exact FHE, the decryption oracle grants $\mathcal{A}$ access to information she cannot compute on her own, resulting or not in a guessing advantage depending on whether or not the cryptosystem at hand is CPA^D secure. As a last remark, let us also emphasize that, in the above game, $\mathcal{A}$ has control on the homomorphic calculations that are performed as f is included in the evaluation request.

3 A CPA^D key recovery attack on Regev

In this section, after recalling the specification of the Regev cryptosystem, we describe how a CPA^D attacker can retrieve the absolute value of the LWE noise in some ciphertexts and then find ciphertexts with noises of the same sign. After enough such ciphertexts have been obtained, the adversary is then able to recover the scheme secret key by solving two linear systems.

3.1 The Regev cryptosystem

We first start by considering the simple Regev cryptosystem [20]. Without loss of generality, we describe only the symmetric variant, which is parametrized by a dimension n, an integer q and a probability distribution χ_σ on $\mathbb{Z}_q$ with standard deviation σ. Plaintexts are elements of $\mathbb{Z}_t$, with $t = 2$, and ciphertexts are elements of $\mathbb{Z}_q^n \times \mathbb{Z}_q$. *These notations will be used consistently throughout the paper.* The scheme is then defined as follows:

- KeyGen: pick a secret key $s \in \mathbb{Z}_q^n$ uniformly at random.
- Enc: given a plaintext $m \in \mathbb{Z}_2$, pick $a \in \mathbb{Z}_q^n$ uniformly at random, pick e in $\mathbb{Z}_q$ according to χ_σ, and return (a, b) with $b = \langle a, s\rangle + \left\lfloor \frac{q}{2} \right\rfloor m + e$.
- Dec: given a ciphertext $c = (a, b)$, return $\left\lceil \frac{2}{q}(b - \langle a, s\rangle) \right\rfloor \mod 2$.

For this cryptosystem, ciphertexts such that

$$\frac{q}{4} \leq b - \langle a, s\rangle < \frac{3q}{4}$$

decrypt to 1. Other ciphertexts decrypt to 0. It is well-known that the Regev cryptosystem is additively homomorphic as given two ciphertexts (a, b) and (a', b') of messages m and m', the ciphertext $(a + a', b + b')$ is an encryption of $m + m'$, provided the total error absolute value is bounded by $q/4$.

3.2 Extracting the noise amplitude of a given ciphertext

In this section, we show how to determine the absolute value of the noise of an encryption of 0 by means only of valid CPA^D security game requests. Given a ciphertext c, we denote by $\mathsf{idx}(c)$, its index in the game state S. Note that $\mathcal{A}$ can mirror state S on her side.

Extracting the additive depth k. The first step consists in asking for an encryption of 0 via a request of the form $(\texttt{test\_messages}, 0, 0)$. $\mathcal{A}$ then gets $c_0 = (a, \langle a, s\rangle + e)$, e being of course unknown to her. By means of CPA^D game requests of the form $(\texttt{eval}, \mathsf{sum}, \mathsf{idx}(c_i), \mathsf{idx}(c_i))$ for $i \in [\![0, k-1]\!]$, $\mathcal{A}$ can obtain ciphertexts $c_{i+1} = c_i + c_i$ for $i \in [\![0, k-1]\!]$, and decrypt them via CPA^D game request of the form $(\texttt{ciphertext}, \mathsf{idx}(c_i))$. In particular,

$$c_k = (2^k a, \langle 2^k a, s\rangle + 2^k e). \tag{2}$$

$\mathcal{A}$ then stops:

- either when c_{k-1} decrypts to 0 and c_k decrypts to 1, in which case she can conclude that
$$\frac{q}{2^{k+2}} \le |e| < \frac{q}{2^{k+1}}, \tag{3}$$
- or when c_k still decrypts to 0 with $2^k > \frac{q}{4}$. In this latter case, she can conclude that $|e| = e = 0$ and gets one linear equation in s. The probability of this happening depends on the initial noise distribution.

Note that unless c_0 is identified as noise free, given $1 \le \alpha < \frac{q}{4}$ such that $\alpha = \sum_{k=0}^{\lceil \log_2(\alpha)\rceil} \alpha_k 2^k$, via one of more CPA^D game $(\texttt{eval}, \mathsf{sum}, \ldots)$ requests , $\mathcal{A}$ obtains ciphertext

$$c^{(\alpha)} := \sum_{k:\alpha_k=1} c_k,$$

so that

$$c^{(\alpha)} = (\alpha a, \langle \alpha a, s\rangle + \alpha e).$$

The parameter α_*. The adversary can start a dichotomic search for the value α_* such that $c^{(\alpha_*)}$ duly decrypts to 0 while $c^{(\alpha_*+1)}$ decrypts to 1 and conclude that

$$\frac{q}{4(\alpha_* + 1)} \le |e| < \frac{q}{4\alpha_*}. \tag{4}$$

Determination of |e|. It follows that $|e|$ is uniquely determined when

$$\left\lceil \frac{q}{4(\alpha_* + 1)} \right\rceil = \left\lfloor \frac{q}{4\alpha_*} \right\rfloor \tag{5}$$

and this occurs (Sect. A) when

$$|e| < \frac{\sqrt{q}}{2}. \tag{6}$$

or, more generally, when $|e| < \sqrt{\frac{q}{2t}}$ (for $t \geq 2$). So, at this point, the bottom line is whether or not this is likely to occur often enough to lead to a practical attack, considering commonly used FHE parameters. For example, for TFHE, $q = 2^{32}$ and $\sigma = 2^{17}$ are often used for binary plaintexts. Then, ciphertexts with noise below $\sqrt{q/4} = 2^{15}$ will have their noise completely determined. A rough calculation tells us that this will be the case for around 20% of the ciphertexts. This order of magnitude is consistent with the experiments in Table 1. For BFV and BGV, the tendency generally is to use quite large moduli, e.g. 2^{200}, and small standard deviations, e.g. $\sigma = 3.2$. With this kind of parameters, condition (6) is satisfied with overwhelming probability. *Note that condition* (6) *plays no role in the concrete attack, only condition* (5) *does.*

Finally, let us emphasize again that all the decrypted values used in this section are obtained by means of CPA^D game requests of the form

$$(\texttt{ciphertext}, \mathsf{idx}(c^{(\alpha)}))$$

as all involved ciphertexts are by construction registered in the game state S. The magnitude of the noise in our original ciphertext c_0 is then determined, when the above procedure succeeds, by means only of valid CPA^D game requests.

3.3 Identifying ciphertexts with noise of equal sign

We now consider two ciphertexts c_0 and c_0' for which $|e| > 0$ and $|e'| > 0$ (as well as α_* and α_*') have been determined following the previous section. Now, if $\alpha_*|e| + \alpha_*'|e'| > \frac{q}{4}$, $\mathcal{A}$ obtains ciphertext $c^{(\alpha_*)} + c^{(\alpha_*')}$ via a CPA^D game `eval` request, similarly to the previous section. If this latter ciphertext decrypts to 1, she can conclude that e and e' have the same sign, since $\alpha_* e$ and $\alpha_*' e'$ have added up over $\frac{q}{4}$. Otherwise, when 0 is obtained as a decryption, $\alpha_*' e'$ partially cancelled $\alpha_* e$ and the resulting noise magnitude remained below $\frac{q}{4}$.

3.4 Finalizing key recovery

Using the techniques in the two previous sections, $\mathcal{A}$ can now proceed as follows, starting from an initial encryption of 0, c_0, obtained by means of a CPA^D game `test_messages` request, for which $|e| > 0$ is exactly determined (following

Sect. 3.2). She then looks for a second encryption of 0, c'_0 such that $|e'|$ is determined and checks if c'_0's noise e' has the same noise sign as c_0's noise e, as in Sect. 3.3. Note that ciphertexts identified as noise-free during the procedure in Sect. 3.2 are always kept at this stage. If not, $\mathcal{A}$ discards c'_0 and restarts, otherwise she keeps c'_0. $\mathcal{A}$ then repeats this procedure until she gets n ciphertexts $c_1, \ldots, c_n$ with known noise magnitudes $|e_1|, \ldots, |e_n|$ determined as in Sect. 3.2, and same noise signs (that is, either $e_i = |e_i|$ for each i or $e_i = -|e_i|$ for each i). Then, there only remains to solve the two linear systems

$$\begin{aligned} \langle a_1, s \rangle &= b_1 + |e_1| \\ \ldots &= \ldots \\ \langle a_n, s \rangle &= b_n + |e_n| \end{aligned}$$

and

$$\begin{aligned} \langle a_1, s \rangle &= b_1 - |e_1| \\ \ldots &= \ldots \\ \langle a_n, s \rangle &= b_n - |e_n|, \end{aligned}$$

getting two candidate values for the secret key s. She can then determine which of these two solutions is the correct key by asking a few more encryptions of 0 and attempting to decrypt them with both candidate secret keys, with the correct one always leading to correct decryptions. Once the correct key is identified, $\mathcal{A}$ trivially wins the CPA^D game by decrypting the outcome of a single request of the form $(\texttt{test\_messages}, 0, 1)$.

Note that, when the probability for χ_σ to generate a noiseless LWE pair is large enough, the attack can be carried out by considering only ciphertexts for which the procedure in Sect. 3.2 determines that $|e| = 0$. In this case, although the adversary has to examine more ciphertexts, only one linear system needs to be solved as each noiseless encryption of 0 directly gives a linear equation in the key.

3.5 First experimental results

Table 1 provides some statistics for a number of representative LWE parameter sets. These statistics have been obtained with a preliminary Python implementation of the attack on Regev. In the table, n, q, σ are the LWE parameters, λ the security level estimated with the lattice-estimator[1], P is the proportion of ciphertexts for which $|e|$ was determined following Sect. 3.2, P_0 the proportion of ciphertexts for which it was determined that $|e| = 0$, the other columns respectively provide the number of encryption, evaluation and decryption requests

[1] https://github.com/malb/lattice-estimator

done to achieve secret key recovery. Following these preliminary results, we are now ready to attempt to implement the attack against real-world libraries. We do so in Sect. 4.

n	q	σ	λ	P	P_0	# enc	# eval	# dec
636	2^{32}	2^{17}	97	0.235	≈ 0	5358	150437	150437
1024	2^{32}	2^{17}	175	0.226	≈ 0	8427	236324	236324
8192	2^{240}	3.19	82	≈ 1	0.13	14593	6512218	6512218
16384	2^{240}	3.19	218	≈ 1	0.16	29016	12924174	12924174

Table 1. Statistics obtained by a proof-of-concept implementation of the attack in Sect. 3. The security level λ is estimated using the lattice-estimator. Note that for the last two rows, we are always able to determine $|e|$, hence $P \approx 1$, since condition (6) is satisfied with overwhelming probability.

3.6 Adaptation to RLWE

Let us consider an elementary RLWE variant of Regev working over $R_q = \mathbb{Z}_q[X]/(X^n+1)$ [18].
Encryption: given $m_0, \ldots, m_{n-1} \in \mathbb{Z}_2^n$ return (a, b) where $b = a \cdot s + \left\lfloor \frac{q}{2} \right\rfloor m + e$ where $m(X) = \sum_{i=0}^{n-1} m_i X^i$.
Decryption: given (a, b), return $\left\lceil \frac{2}{q}(b - a \cdot s) \right\rfloor \mod 2$ where the rounding is performed coefficient-wise.

Essentially, this RLWE variant consists in grouping n LWE pairs in a single ciphertext, since the i-th ($i \in [\![0, n-1]\!]$) coefficient of polynomial b is

$$\sum_{j=0}^{i} a_{i-j} s_j - \sum_{j=i+1}^{n-1} a_{i+n-j} s_j + \left\lfloor \frac{q}{2} \right\rfloor m_i + e_i. \tag{7}$$

As such, our attack on Regev is straightforward to adapt to this RLWE setting by focusing on a single arbitrarily chosen coefficient, say i, of the involved polynomial. For example, illustrating this for the first steps of the attack, the adversary starts by asking for an encryption of the null polynomial 0 via a request of the form $(\texttt{test\_messages}, 0, 0)$. $\mathcal{A}$ then gets $c_0 = (a, a \cdot s + e)$, the polynomial e being of course unknown to her. Similarly to the LWE case, by means of CPA^D game requests of the form $(\texttt{eval}, \mathsf{sum}, \mathsf{idx}(c_i), \mathsf{idx}(c_i))$ for $i \in [\![0, k-1]\!]$, $\mathcal{A}$ can obtain ciphertexts $c_{i+1} = c_i + c_i$ for $i \in [\![0, k-1]\!]$, where

$$c_k = (2^k a, 2^k a \cdot s + 2^k e).$$

She stops:

– either when c_{k-1} decrypts to *a polynomial whose i-th coefficient is* 0 and c_k decrypts to *a polynomial whose i-th coefficient is* 1 (ignoring the values of

coefficients $j \neq i$ in the decrypted polynomials), in which case $\mathcal{A}$ can conclude that

$$\frac{q}{2^{k+2}} < |e_i| \leq \frac{q}{2^{k+1}},$$

– or when c_k still decrypts to *a polynomial whose i-th coefficient is* 0 with $2^k > \frac{q}{4}$, in which case she can conclude that $|e_i| = e_i = 0$ and straightaway obtain one linear equation in s following (7):

$$\sum_{j=0}^{i} a_{i-j} s_j - \sum_{j=i+1}^{n-1} a_{i+n-j} s_j = 0.$$

The rest of the attack works similarly to the LWE case. $\mathcal{A}$ performs RLWE CPA^D game requests until she gets n ciphertexts $c^{(1)}, \ldots, c^{(n)}$ with known noise magnitudes $|e_i^{(1)}|, \ldots, |e_i^{(n)}|$ and same noise signs *in the i-th coefficient of their noise polynomials*. She can then post-process these ciphertexts on her own to extract and solve the two linear systems ($\kappa \in \{0, 1\}$),

$$\begin{aligned} \sum_{j=0}^{i} a_{i-j}^{(1)} s_j - \sum_{j=i+1}^{n-1} a_{i+n-j}^{(1)} s_j &= b_i + (-1)^{\kappa} |e_i^{(1)}| \\ &\cdots \\ \sum_{j=0}^{i} a_{i-j}^{(n)} s_j - \sum_{j=i+1}^{n-1} a_{i+n-j}^{(n)} s_j &= b_i + (-1)^{\kappa} |e_i^{(n)}| \end{aligned}$$

with one of them giving s.

For simplicity's sake, we have described the most direct adaptation of our CPA^D attack on Regev to RLWE. Even though it is already highly practical in this form, there are a number of simple ways in which it can be optimized. In particular, the adversary may wish to exploit all the coefficients in a given RLWE ciphertext in an attempt to extract more than one linear equation for a given ciphertext. Indeed, given an RLWE encryption of the null polynomial $(a, a \cdot s + e)$, the above noise absolute value determination procedure can be repeated independently for each coefficient index i of the right-hand side polynomial of the RLWE pair, yielding, when condition (5) holds for this index, $|e_i|$. To determine whether the noises e_i and e_j for two coefficients $i \neq j$, for which $|e_i|$ and $|e_j|$ have been determined, have the same sign, $\mathcal{A}$ can employ the same technique as in Sect. 3.3 after a cyclic rotation of the coefficient obtained by a multiplication with the constant polynomial X^{j-i} (depending on the chosen plaintext encoding).

4 Adaptation to mainstream FHE and libraries

In this section, we show that BFV, BGV and TFHE are variants of either Regev or its simple RLWE variant that we described in the previous section and, as such, vulnerable to the CPA^D attack path which we have discussed. For each of these cryptosystems, we provide experimental results obtained when implementing the attack on some of the mainstream libraries which support it. For

all these cryptosystems, we only provide the minimum details required to understand the present paper and refer the reader to the cited references for their full specifications.

4.1 BFV

With BFV[7,15], to encrypt a polynomial message m with the public key $\mathsf{pk} = (p_0, p_1) = \left([-(a \cdot \mathsf{sk} + e)]_q, a\right) \in {R_q}^2$, one must sample $r \xleftarrow{\$} R_2$, $e_0, e_1 \xleftarrow{\$} \chi_\sigma$ and return $c = \left([\Delta \cdot m + p_0 \cdot r + e_0]_q, [p_1 \cdot r + e_1]_q\right)$, where $\Delta = \lfloor \frac{q}{t} \rfloor$. An encryption of 0 is then $c = (c_0, c_1) = \left([p_0 \cdot r + e_0]_q, [p_1 \cdot r + e_1]_q\right)$. Thus, modulo q we have

$$\begin{aligned} c_0 &= -(a \cdot \mathsf{sk} + e) \cdot r + e_0 \\ &= -(a \cdot r) \cdot \mathsf{sk} + e \cdot r + e_0 + e_1 \cdot \mathsf{sk} - e_1 \cdot \mathsf{sk} \\ &= -c_1 \cdot \mathsf{sk} + (e \cdot r + e_0 + e_1 \cdot \mathsf{sk}) \end{aligned}$$

Therefore, a BFV ciphertext is on Regev form $b' = a' \cdot \mathsf{sk} + e'$, where $b' = c_0$, $a' = -c_1$ and $e' = e \cdot r + e_0 + e_1 \cdot \mathsf{sk}$. As discussed in Sect. 3.6, if we only target one coefficient of the polynomial c_0, we obtain an LWE instance and can proceed with the attack as defined on Regev.

SEAL The SEAL[2] library implements the BGV, BFV and CKKS schemes, allowing us to choose default security settings of 128, 192 or 256 bits. So, we first took a shot at the BFV cryptosystem with parameters giving a security of 128 or 256 bits. To carry out the attack, we need to access the plaintext polynomial's coefficients after decryption, which is easily done with this library as it overloads the [ ] operator in C++. The practical attack follows exactly the theoretical pattern: we try to recover the noise of ciphertexts of 0, which can be done when condition $|e| < \sqrt{\frac{q}{2t}}$ (cf (6)) is verified. This condition is almost always satisfied with the default BFV parameters in SEAL, which uses a q that is very large compared to the noise deviation for fresh ciphertexts. For example, 109 bits for $n = 4096$ and 128-bit security and a standard deviation of the Gaussian used to generate the noise equal to $\sigma = 3.2$.

When carrying out the attack, we thus systematically find the absolute value of the noise of the ciphertexts of interest with the dichotomic procedure in Sect. 3.2. As an example, we ran the attack with parameters $n = 4096$, $\log_2(q) = 58$, $\sigma = 3.2$, giving a security of 227 bits according to the lattice-estimator. For this set of parameters, we systematically recovered the absolute value of the noise, enabling us to generate the n linear equations required for key recovery in about 1 m 20 s. The number of calls to the encryption oracle was 7393, and the number of calls to both the evaluation and decryption oracles was 664138. These and other results can be found in Table 2.

[2] https://github.com/microsoft/SEAL version 4.1 [21]

OpenFHE In order to compare the libraries, we also attempted the attack on BFV as implemented in OpenFHE[3] with the default parameters given for 128- and 256-bit security. As described previously, we need to recover the coefficients of the plaintext polynomial after decryption.

OpenFHE offers several plaintext encoding methods. To perform the attack, we chose the simplest path, which consists in using the most direct encoding, namely COEF_PACKED_ENCODING, in which messages are directly put into the plaintext polynomial coefficients, similarly to Sect. 3.6. After decryption, the resulting plaintext is represented as a NativePoly, and we only have to extract the coefficient of interest. Note that the attack can be adapted to the other encoding modes available in OpenFHE. As for SEAL, we also instrumented the library so as to check that the correct noises were duly recovered.

To run the attack on BFV with the OpenFHE library, we choose parameters generated by default by the library, providing a security of 256 bits. For this security, we have to choose an n at least equal to 16384. So we choose $n = 16384$, and the library then uses as default parameters a q such that $\log_2(q) = 120$ and $\sigma = 3.19$. As with SEAL, with this set of parameters, we systematically recover the absolute value of the noise. For a security of 256 bits, the attack takes about 1 h 15 m 30 s. This relatively large time, compared to SEAL, is mostly attributable to the comparatively larger dimension used in the parameter set. As for the number of calls to the oracles, there are around 32700 for the encryption oracle and around 6630736 for the evaluation and decryption oracles. These results are available in table 2.

Lattigo We also implemented our attack targeting the Lattigo library (version 5). We tested it on several sets of parameters, ensuring security from 95 to 217 bits according to the lattice-estimator.

Unlike the OpenFHE library, Lattigo[4] does not propose several encoding types. It uses batching by default and transforms the plaintext polynomial using an INTT. To remain within the boundaries of a CPA^D attack, we duly used the encoding function provided by Lattigo prior to encryption, as well as the decoding function following each decryption. For BFV, Lattigo decryption is done via calls to functions Decrypt which removes $a \cdot s$, and then Decode which scales, rounds and performs a NTT to recover the plaintext polynomial. Hence, in order to recover the coefficients to which the attack is applied, we had to post-process the Decrypt/Decode output by an inverse NTT, as would have been done by a real attacker.

We first ran the attack on "small" parameters, referred to as PN11QP54 and PN12QP101pq in version 4 of the library, and still supported in the latest version 5. These parameters are given for 128 bits of security, but achieve around 100 bits of security according to the latest lattice-estimator. For these, we systematically found the absolute value of the noise. For example, for $n = 2048$, $\log_2(q) = 54$ and $\sigma = 3.2$, the attack runs in less than 50 s and finds the 2048 linear equations

[3] https://github.com/openfheorg/openfhe-development version 1.1.2 [5]
[4] https://github.com/tuneinsight/lattigo version 5 [1]

in 4086 calls to the encryption oracle, and 274638 calls to the evaluation and decryption oracles.

However, when we tried to carry out our attack with larger parameters, we could not determine the exact error, but we did find an interval containing two possible absolute values for the error. After investigation, we attribute this to slight inaccuracies in the decryption function when using large parameters. Although we were not able to exactly pinpoint the root cause of this behavior in the library, we were able to reproduce that behaviour within an independent Python script in which down scaling was done using Decimal rather than Fraction. This slightly disrupted our attack as condition (5) was never satisfied, leading to two distinct values, with only one of them giving the absolute noise of the ciphertext. Still, we experimentally noticed that when the ciphertext had a negative noise, the absolute value of this noise was equal to the left-hand bound, and that when the ciphertext had a positive noise, this noise was equal to the right-hand bound. Having noticed that this phenomenon occurred systematically, we adapted our attack without deviation from the CPA^D context. The CPA^D requests to the oracles remain identical. The only difference for the attacker, who remains passive, is in the post-processing. The attacker constructs two systems of linear equations, the first with all noises assumed to be negative, taking the left-hand bound of (5) as the absolute value of the error, and the second with all noises assumed to be positive, taking the right-hand bound of (5). With this slight adaptation, we still determine the n equations needed to find the secret key. Of course, in a real-world attack, the adversary can invest (as we did) as much time as needed in reverse engineering of the target library in order to adapt to its peculiarities. For example, we eventually were able to run our attack against parameter set $n = 4096$, $\log_2(q) = 60$ and $\sigma = 3.2$, which achieves a security of 217 bits according to the lattice-estimator. This last set of parameters allows us to find the desired set of linear equations in less than 1 m 30 s. For this set, the number of calls to the encryption oracle is 7983, and the number of calls to the decryption oracle is 326947.

Finally, the Lattigo library also supports threshold schemes. In this case, an additional so-called smudging noise is added during the multi-party decryption process. Depending on its deviation, σ_{smg}, this noise may or may not prevent our attack. Still, when its deviation is set to 0, an option which is offered by older versions of the library (up to version 4), the attack works without modifications. When that noise standard deviation is small, e.g. using the default $\sigma_{\text{smg}} = 3.2$ in version 5 of the library, *we have been able to perform the attack without modifications*. However, in this latter case, we experimentally observed that condition (5) is not always satisfied but always yields the correct value of $|e|$ when it is. For example, for parameters $n = 4096$, $\log_2(q) = 101$, $\sigma = 3.2$, $\sigma_{\text{smg}} = 3.2$ and 5 parties (thus leading to an additional noise of deviation $3.2\sqrt{5}$), the attack examines 11595 (vs 9814 in the single key case, hence without smudging) ciphertexts, performs 1137984 (vs 1071497) calls to the evaluation/decryption oracle and an overall time of 18 m 20 s (vs 6 m 40 s). The latter cost difference is

imputable to both the larger number of ciphertexts examined as well as the fact that collaborative decryption is more expensive than single-key decryption.

When σ_{smg} is a large constant but smaller than a certain λ-dependent bound, e.g. $\sigma_{\text{smg}} = 2^{30}$ as suggested in examples provided within the library, we are still able to perform a slower variant of the attack via the identification of 0-noise LWE pairs (see Sect. 3.4). However, when σ_{smg} is a carefully chosen λ-dependent constant, then the attack is prevented. We further discuss these two cases in full detail in Sect. 5, which is dedicated to threshold variants of BFV and BGV following the Lattigo blueprint.

4.2 BGV

For BGV[8], we remark that since the attack requires only homomorphic additions, level switchings can be ignored.

In BGV, to encrypt $m \in R_p$, one must sample $u \xleftarrow{\$} R_p$ and $e_1, e_2 \xleftarrow{\$} \chi_\sigma$ then create a level-L BGV ciphertext $c = \left(u \cdot \mathsf{pk}_{L,0} + m + p \cdot e_0, u \cdot \mathsf{pk}_{L,1} + p \cdot e_1\right)$, where the public key at level L is $\mathsf{pk}_L = (\mathsf{pk}_{L,0}, \mathsf{pk}_{L,1}) = (a_L \cdot \mathsf{sk}_L + p \cdot e_L, -a_L)$. A level-$\ell$ encryption of 0 is $c = (c_0, c_1) = \left(u \cdot \mathsf{pk}_{\ell,0} + m + p \cdot e_0, u \cdot \mathsf{pk}_{\ell,1} + p \cdot e_1\right)$. Then,

$$\begin{aligned} -c_0 &= -(a_\ell \cdot \mathsf{sk}_\ell + p \cdot e_\ell) \cdot u + p \cdot e_0 \\ &= -(a_\ell \cdot u) \cdot \mathsf{sk}_\ell + p \cdot e_\ell \cdot u + p \cdot e_0 + p \cdot e_1 \cdot \mathsf{sk}_\ell - p \cdot e_1 \cdot \mathsf{sk}_\ell \\ &= c_1 \cdot \mathsf{sk}_\ell - (p \cdot e_\ell \cdot u + p \cdot e_0 + p \cdot e_1 \cdot \mathsf{sk}_\ell) \end{aligned}$$

Thus, a BGV ciphertext follows Regev's construction $b' = a' \cdot \mathsf{sk} + e'$, where $b' = -c_0$, $a' = c_1$ and $e' = p \cdot e_\ell \cdot u + p \cdot e_0 + p \cdot e_1 \cdot \mathsf{sk}_\ell$. As for the BFV case, we can focus on only one coefficient of c_0 to obtain an LWE instance and go back to the Regev case.

SEAL Again, we targeted the SEAL library to perform the attack on BGV, using a set of default parameters giving 256-bit security according to the library. Without surprise, the attack on BGV works similarly to that on the BFV cryptosystem for that library (see Sect. 4.1).

With the parameters $n = 4096$, $\log_2(q) = 58$, $\sigma = 3.2$, (227 bits of security according to the lattice-estimator), we always find the absolute value of the noise of a ciphertext of interest. We generated the 4096 linear equations in less than a minute, calling the encryption oracle 8183 times and the evaluation and decryption oracles 661647 times. These results and others are provided in table 2.

OpenFHE The attack on the OpenFHE implementation of BGV works similarly to that of BFV. The main difference is that a CryptoContextBGVRNS context is generated instead of a CryptoContextBFVRNS context. The polynomial coefficients are recovered in the same way as for BFV. After decryption, we get the NativePoly from the decrypted plaintext and extract the coefficient of interest. We carried out this attack on parameters offering 128-bit and 256-bit security.

As with BFV, we systematically found the absolute noise's value and were thus able to recover the linear equations required for the attack.

As an example, we used the parameters generated by default by the library: $n = 16384$, $\log_2(q) = 120$ and $\sigma = 3.19$. With these parameters, we systematically found the absolute value of the noise and thus carried out the attack in approximately 1 h 8 m 50 s. The number of calls to the evaluation and decryption oracles is 3475514 while that of the encryption oracle is 32779. We present these results and others in Table 2.

HElib To the best of our knowledge, HElib[5] is the only library that performs *noise level monitoring and in fact blocks decryption* when the estimated noise level is deemed too large to result in a correct decryption.

Specifically, upon decrypting a Ctxt with the function SecKey::Decrypt(), the function Ctxt::isCorrect() compares the Ctxt's noise level estimate noiseBound against its ciphertext modulus. If Ctxt::isCorrect() returns false, SecKey::Decrypt() exits with a warning and without running the decryption algorithm.

To work in this context, our attack would require to generate ciphertexts which actually fail to correctly decrypt before they are flagged by Ctxt::isCorrect(). But the parameters in HElib have been chosen so that this happens with a very small probability which does not depend on λ. The implication is that one cannot determine the (additive) depth k on ciphertexts, as in (3), let alone the value of α_* defined in (4). As a result, HElib prevents us from extracting even the magnitude of the noise e in the initial ciphertext.

In conclusion, thanks to this mechanism HElib appears to be the only library that is immune to our attack in its present form. This hints at a possible countermeasure to mitigate the attack by monitoring noise deviations in ciphertexts and choosing instance parameters rendering the probability of incorrect decryption negligible in the security parameter λ, rather than a very small λ-independent probability as presently done in the library. This will be further discussed in Sect. 6.

We note however that, while the exact determination of $|e|$ seems out of reach against HElib, the possibility remains open to reduce the range of possible values for $|e|$ and thus effectively reduce the security level. This deserves further investigations.

4.3 TFHE

The TFHE encryption scheme was proposed in 2016 [11] and updated in [13]. It introduces the TLWE problem as an adaptation of the LWE problem to the Torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$, and specifies the most efficient bootstrapping operation in the literature [6].

TFHE relies on three structures to encrypt plaintexts defined over $\mathbb{T}$, $\mathbb{T}_n[X]$ or $\mathbb{Z}[X]/(X^n + 1)$. In this work, we are only interested in *TLWE samples* that

[5] https://github.com/homenc/HElib version 2.2.0

serve for encrypting messages in $\mathbb{T}$. A pair (a, b) is a valid TLWE sample if $a \xleftarrow{\$} \mathbb{T}^n$ and $b \in \mathbb{T}$ satisfies $b = \langle a, s \rangle + e$, where $s \xleftarrow{\$} \mathbb{B}^n$ is the secret key, and $e \xleftarrow{(0,\sigma^2)} \mathbb{T}$ is a Gaussian noise. Then, (a, b) is a fresh encryption of 0.

Let $\mathcal{M} \subset \mathbb{T}$ be the discrete message space[6]. To encrypt a message $m \in \mathcal{M} \subset \mathbb{T}$, we add $(0, m)$ to a fresh TLWE sample (a, b), to obtain the ciphertext $c = (a, b + m)$. In the following, we refer to an encryption of m with the secret key s as a TLWE ciphertext noted $[\![m]\!] = c \in \mathrm{TLWE}_s(m)$.

To decrypt a sample $c \in \mathrm{TLWE}_s(m)$, we compute its *phase* $\phi(c) = b - \langle a, s \rangle = m + e$. Then, we round it to the nearest element of $\mathcal{M}$.

We note that TFHE is an adaptation of the Regev encryption scheme to $\mathbb{T}$, and so it is an additive homomorphic encryption scheme. That is, if we add $c_1 \in \mathrm{TLWE}_s(m_1)$ to $c_2 \in \mathrm{TLWE}_s(m_2)$, we get $c \in \mathrm{TLWE}_s(m_1 + m_2)$.

TFHElib TFHElib[7] is an open source library providing the original TFHE implementation. It supports two discretizations for $\mathbb{T}$ either on 32 or 64 bits[8]. That is, in practice, a TLWE sample corresponds to a Regev encryption with $q = 2^{32}$ (or $q = 2^{64}$). TFHElib allows adding ciphertexts without bootstrapping. So, it provides the elementary operations for implementing the attack described in Sect. 3. Indeed, we implemented this attack targeting TFHElib, with various typical parameter sets, and we succeeded, in less than 1 second, in finding the number of linear equations required for key recovery (as described in Table 2). Note that the attack runs much faster for TFHElib since TFHE usually requires relatively small dimensions (typically below 1024) and moduli (e.g. 2^{32}) which fit in native machine words to achieve above 128 bits security. This is in opposition to the typical parameters in the BFV/BGV realm, which lead to much larger dimensions (e.g. 16384) and moduli over more than 100 bits.

[6] In practice, to encrypt a message $m \in \mathbb{Z}_t$, we encode it in $\mathbb{T}$ as $\frac{m}{t}$ then we add it to a fresh TLWE sample as: $(a, b + \frac{m}{t})$. That is, $\mathcal{M} = \{0, \frac{1}{t}, \cdots, \frac{t-1}{t}\}$

[7] https://tfhe.github.io/tfhe/ [12]

[8] The 64-bit implementation of TFHE is less maintained when compared to the 32-bit one. So, when we refer to TFHElib in this paper, we will be referring to 32-bit implementation.

Library	Scheme	Parameters					Proportion of ctxt with $\lvert e\rvert$ recovered	Proportion of noisefree ctxt	Nb of encrypt.	Nb of lin. eq.	Time	Nb of eval./decrypt.
		λ	n	$\log_2(q)$	σ	t						
SEAL	BFV	95	4096	109	3.2	1024	1	6250/232858	7979	4096	$2m50s$	927163
	BFV	227	4096	58	3.2	1024	1	1481/860557	7393	4096	$1m20s$	664138
	BGV	227	4096	58	3.2	1024	1	124/65405	8183	4096	$52s$	661647
OpenFHE	BFV	128	8192	120	3.19	1024	1	69/48929	16235	8192	$19m30s$	3357143
	BFV	256	16384	120	3.19	1024	1	173/130535	32700	16384	$75m30s$	6630736
	BGV	128	8192	69	3.19	1024	1	59/32811	16405	8192	$18m30s$	1690360
	BGV	256	16384	71	3.19	1024	1	80/65559	32779	16384	$68m50s$	3475514
HElib	BGV	187	8096	108	3.2	-	-	-	-	-	-	-
TFHElib	TFHE	97	630	32	2^{17}	2	1295/5427	0	5427	630	$0.245s$	203879
	TFHE	128	700	32	81604.378	2	1363/3678	0	3678	700	$0.195s$	146072
	TFHE	128	1024	32	81604.378	4	2070/5608	0	5608	1024	$0.412s$	223098
	TFHE	128	1024	32	279.172	16	2021/2041	11/2041	2041	1024	$0.237s$	131783
Lattigo	BFV	95	4096	109	3.2	65537	1	785/29241	7860	4096	$5m50s$	908795
	BFV	98	2048	54	3.2	65537	1	69/24518	4086	2048	$46s$	274638
	BFV	106	4096	101	3.2	65537	1	829/32260	9814	4096	$6m40s$	1071497
	BFV	106	8192	202	3.2	65537	1	457/23943	9960	8192	$52m00s$	3129570
	BFV	217	4096	60	3.2	65537	1	828/31934	7983	4096	$1m25s$	326947

Table 2. Summary of experimental results for Sect. 4

5 Remarks on threshold homomorphic encryption

One of the existing multi-user approaches is the Threshold (also called Multi-Party) Homomorphic Encryption [2,3]. It allows users to encrypt their data using a joint public key, constructed from their individual public keys. The decryption phase is collaborative, such that no user holds the associated global private key. In this setting, no user should be able to extract information on another user's or the joint private key from the public or shared knowledge (i.e. the joint public key or even the encrypted messages under this joint key).

Thus, in a threshold approach, users can encrypt their data using this joint public key and perform collaborative decryption without knowledge of the global secret key. Now, if the underlying scheme is CPA^D insecure, an adversary may be able to retrieve the decryption key from the knowledge of several triplets $\{m, c, \mathsf{Dec}(c)\}$, where c is an encryption of m. In the collaborative threshold decryption setup, each user is first given the ciphertext c to decrypt. This ciphertext is encrypted under the global secret key that he or she does not know. At the end of the collaborative decryption protocol, each user is granted access to $\mathsf{Dec}(c)$. Assuming the messages are also known, every user is clearly in the position of a CPA^D attacker on the global secret key. Therefore, either CPA^D security or a countermeasure against CPA^D attacks is a must-have in the threshold setting.

5.1 The case of Lattigo's blueprint

Now, we illustrate this with the K-out-of-K threshold scheme from [19] based on core RLWE-based homomorphic encryption and which serves as a basis for the Lattigo library. Our notations for the rest of this section are the ones of [19].

Let CRS be the uniform distribution in R_q, i.e., elements sampled from this distribution are uniformly distributed and the same for all K parties $P_i \in \mathcal{P}$. In addition with the usual public parameters for RLWE, this multi-user scheme requires a public polynomial p_1 with coefficients sampled from the CRS [9].

The resulting scheme is then specified by a tuple MBFV = (EncKeyGen, RelinKeyGen, KeySwitch, PubKeySwitch) that extends the BFV scheme with the following machinery:

- $\mathsf{EncKeyGen}(\mathsf{sk}_1, \ldots, \mathsf{sk}_K)$ constructs a collective encryption key from individual secret keys;
- $\mathsf{RelinKeyGen}(\mathsf{sk}_1, \ldots, \mathsf{sk}_K)$ constructs a collective relinearization key from individual secret keys;
- $\mathsf{KeySwitch}(\mathrm{ct}, \mathsf{sk}'_1, \ldots, \mathsf{sk}'_K, \mathsf{sk}_1, \ldots, \mathsf{sk}_K)$ re-encrypts a ciphertext ct from a collective public key $\mathsf{EncKeyGen}(\mathsf{sk}_1, \ldots, \mathsf{sk}_K)$ to a collective public key $\mathsf{EncKeyGen}(\mathsf{sk}'_1, \ldots, \mathsf{sk}'_K)$ using private knowledge;
- $\mathsf{PubKeySwitch}(\mathrm{ct}, \mathsf{pk}', \mathsf{sk}_1, \ldots, \mathsf{sk}_K)$ re-encrypts a ciphertext ct from a collective public key $\mathsf{EncKeyGen}(\mathsf{sk}_1, \ldots, \mathsf{sk}_K)$ to another public key pk', using only public knowledge.

The protocol then runs as follows. Each party P_i constructs its individual public key $p_{0,i} \coloneqq -p_1\mathsf{sk}_i + e_i$ from its individual private key sk_i. The joint public key $\mathsf{EncKeyGen}(\mathsf{sk}_1, \ldots, \mathsf{sk}_K)$ is then (p_0, p_1) with $p_0 = \sum_{P_j \in \mathcal{P}} p_{0,j}$. We do not detail how the joint re-linearization key is constructed as our attacks require only homomorphic additions.

To perform a collaborative key switch $\mathsf{KeySwitch}(\mathrm{ct}, \mathsf{sk}'_1, \ldots, \mathsf{sk}'_K, \mathsf{sk}_1, \ldots, \mathsf{sk}_K)$ on a ciphertext (recall Sect. 4.1)

$$\mathrm{ct} = (c_0, c_1) = (\Delta \cdot m + p_0 \cdot r + e_0, p_1 \cdot r + e_1),$$

each party P_i (owner of the secret keys $s_i \coloneqq \mathsf{sk}_i$ and $s'_i \coloneqq \mathsf{sk}'_i$) picks a smudging noise e_i and discloses

$$h_i \coloneqq (s_i - s'_i)c_1 + e_i$$

to the other parties. The smudging noise is sampled from a discrete Gaussian distribution with (*very large* λ-dependent) variance $\sigma^2_{\mathrm{smg}} = 2^\lambda \sigma^2_{\mathrm{ct}}$, where σ^2_{ct} is the ciphertext noise variance (which has to be monitored during FHE evaluations) and λ is, as usual, the target security level. The output re-encrypted ciphertext $\mathrm{ct}' \coloneqq (c_0 + \sum_{P_j \in \mathcal{P}} h_j, c_1)$ can then be computed by any party.

Now, observe that the key switching protocol $\mathsf{KeySwitch}$ with private inputs $(s_i, 0)$ (that is, $s'_i = 0$ for all party i) is actually a collaborative decryption protocol. To perform this collaborative decryption, each party P_i picks a smudging noise e_i as above and discloses $h_i \coloneqq s_i c_1 + e_i$ to the other parties. The plaintext is then obtained by each party by computing $c_0 + \sum_{P_j \in \mathcal{P}} h_j$.

Note that this collective decryption is actually equivalent to a decryption under the key $s \coloneqq \sum_{P_i \in \mathcal{P}} s_i$, which is not known by any unique party. Hence, each party is actually a potential CPA^D attacker on s. Indeed, each party can obtain ciphertexts by performing encryption with the joint public key, evaluation requests on these ciphertexts, and collaborative decryption requests on them, which is equivalent to decryption requests under the associated global private key s that she does not know.

As discussed, the smudging technique, as introduced by Asharov et al. [4] consists in flooding the partial decryptions with a fresh noise term of large λ-dependent variance. This technique is necessary in the security proofs for these threshold schemes. However, identifying concrete attack paths that would be mitigated by smudging was left as an open problem in [19]. From the smudging lemma [4], Mouchet et al. show (proof of Lemma 3 of [19]) that when e_{smg} is indeed picked from a discrete Gaussian distribution of variance $\sigma^2_{\mathrm{smg}} = 2^\lambda \sigma^2_{\mathrm{ct}}$, the distribution of $e_{\mathrm{ct}} + e_{\mathrm{smg}}$, where e_{ct} is the ciphertext noise, is statistically indistinguishable from the one of e_{smg}. Getting back to the attack in Sect. 3, and stated in LWE terms for simplicity sake, this means that ciphertext c_k (recall equation (2)) flooded with smudging noise i.e.,

$$c_k^{(\mathrm{smg})} = (2^k a, \langle 2^k a, s \rangle + 2^k e + e_{\mathrm{smg}}), \tag{8}$$

where $\sigma_{\mathrm{smg}} = 2^k \sigma \sqrt{K} 2^{\frac{\lambda}{2}}$ and $\sigma_{\mathrm{ct}} = 2^k \sigma$, is indistinguishable from ciphertext

$$(2^k a, \langle 2^k a, s \rangle + e_{\mathrm{smg}}),$$

and, hence, that no information can be extracted on e from $c_k^{(\mathrm{smg})}$ with more than $\mathrm{neg}(\lambda)$ advantage.

The smudging technique is then an efficient countermeasure, without which the above threshold scheme would be vulnerable to our attack, as long as the base FHE scheme is not CPA^D secure. Hence, smudging noise addition with the proper σ_{ct} and λ-dependent standard deviation should be used in a systematic way, and *not be made optional* by any library implementing the threshold homomorphic encryption scheme specified in [19]. For instance, as discussed in Sect. 4, the Lattigo library (version 5) uses a minimum value for σ_{smg} of 3.2, independently of σ_{ct} and λ, which is not sufficient to prevent our attack. Even when a large, yet λ-independent, deviation smudging noise is used (e.g. $\sigma_{\mathrm{smg}} = 2^{30}$ is suggested in parameter set examples for Lattigo) it is not enough to prevent CPA^D attacks as ours. In that case, we can still exploit the fact that it remains possible to identify LWE pairs with 0-noise, therefore yielding linear equations in the key, (as discussed in Sect. 3.2, 3.4 and 3.6 for RLWE where each coefficients of the b polynomial are considered independently). Indeed, when c_k, as defined in (2), still decrypts to 0 when $2^k > \frac{q}{2t}$, the adversary can conclude that its noise $e = 0$. Now, when a c_k with $e = 0$ is smudged as in (8) but with a noise e_{smg} with a λ-independent deviation, the probability that $c_k^{(\mathrm{smg})}$ does not decrypt to 0,

$$P\left(|e_{\mathrm{smg}}| > \frac{q}{2t}\right),$$

is independent of k and depends solely on σ_{smg}. The probability of this happening is then bounded by $\left(\frac{2t\sigma_{\mathrm{smg}}}{q}\right)^2$. For example, with typical Lattigo parameters $\log_2(q) = 101$, $t = 65537$ and $\sigma_{\mathrm{smg}} = 2^{30}$, this occurs with probability less than $3.08 \cdot 10^{-33}$. Also, since the typical initial ciphertext noise deviation is $\sigma = 3.2$, 0 values are generated quite often for the noise. We have been able to confirm this experimentally with the Lattigo library by being able to duly identify the 13 RLWE ciphertexts with 0 noise in their first coefficient over 600 generated ciphertexts. Of course such an attack will have to examine many more ciphertexts (around 50 times more), even though it would remain totally practical.

5.2 Other threshold schemes

There also exists variants of the scheme in [19] with a fixed parameter $T < K$, in which any T parties among the K ones are enough to perform a collaborative decryption. In such variants, the individual keys are constructed by means of Shamir's secret sharing, and the collective key is still a linear combination of them, constructed from a Lagrange interpolation polynomial. Hence, the conditions of the previous section are still met, and any one of the T decrypting parties is still a potential CPA^D adversary.

To the best of our knowledge, all the existing threshold constructions from HE schemes based on LWE and its variants are analogous to this one, with a collective decryption protocol equivalent to a simple decryption under an algebraic sum of individual private keys in some ring structure, making each party

a CPA^D attacker in a similar way. Hence, existing threshold schemes over BFV, BGV and CKKS may be insecure if no smudging is applied, due to the CPA^D insecurity of these schemes.

6 Attack mitigation strategies

In this section, we investigate a number of natural countermeasures.

6.1 Monitor&Block

The Monitor&Block mechanism introduced in HElib for reliable decryption, as described in Sect. 4.2, hints at a possible countermeasure for BFV/BGV implementations. Let us define the noise deviation budget, denoted B, as the standard deviation in ciphertexts obtained after a target algorithm has been executed over fresh ciphertexts. We then wish to choose the cryptosystem parameters such that a ciphertext with noise deviation B has probability $\mathrm{neg}(\lambda)$ to generate a decryption error. Then, ciphertext noise standard deviation must be monitored by the homomorphic operator implementation[9] and the decryption function has to provide no output for ciphertexts with noise deviation beyond the noise budget B. Note that noise deviation monitoring and blocked decryption thus become part of the cryptosystem specification and, as such, a CPA^D attacker cannot work around them. This has implications on the choice for the modulus q as we then want that (Sect. B)

$$\frac{q}{2t} \geq 2^{\frac{\lambda}{2}} B \tag{9}$$

Hence, this has implications on the choice of n and on the efficiency of the unitary homomorphic operations. Let us give a coarse idea for the cost of this countermeasure for BFV. For example taking $t = 65537$, $n = 16384$, $\sigma = 3.2$ and $\lambda \geq 128$ for a multiplicative depth of 4 leads to a bound $B \approx 1.53 \cdot 10^{68}$ (see Sect. B), thus, following (9), requiring a 289-bits q. When asked for parameters for multiplicative depth 4, OpenFHE however chooses a 240 bits q. Note that switching from a 240 to a 289 bits modulus does not require to increase n beyond 16384 to remain above 128 bits security. Still this modulus increase is far from computationally free as it increases the homomorphic multiplication cost from 24.6 ms to 33.4 ms leading to 35% increase (the cost of noise deviation monitoring being negligible). Table 3 provides more illustrative insights at the cost of this countermeasure for OpenFHE/BFV.

6.2 Bootstrapping

Another natural strategy is to bootstrap after each homomorphic operation, including additions, which is unusual except for TFHE. Since bootstrapping resets the noise variance to a preset value, decryption errors cannot occur anymore and

[9] By opposition to HElib which monitors upper bounds satisfied with high probability.

d	$\log_2(q)$	n	$\log_2(q)$	n	ratio
1	120	8192	131	8192	1,09
2	180	8192	181	8192	1,00
3	180	8192	237	16384	2,96
4	240	16384	289	16384	1,35
5	240	16384	341	16384	1,68
6	300	16384	392	16384	1,46
7	300	16384	444	16384	1,66
8	360	16384	516	32768	3,37
9	360	16384	570	32768	3,93
10	420	16384	624	32768	3,65

Table 3. Illustration of the performance cost of the Monitor&Block countermeasure for OpenFHE/BFV. Column d incidates the target multiplicative depth. The left $\log_2(q)$ and n columns indicate the parameters chosen by OpenFHE for 128 bits security ($t = 65537$). The right $\log_2(q)$ and n columns indicate the parameters required to ensure 128 bits security and a 2^{-128} probability of incorrect decryption when depth k is reached (following the estimates in Sect. B). The last column indicates the resulting *measured* performance cost ratio over 10000 BFV multiplications.

the scheme becomes exact with probability $1 - \mathrm{neg}(\lambda)$, as long as the bootstrapping itself has a probability of error in $\mathrm{neg}(\lambda)$. In this case, CPA security and CPA^D security are known to be equivalent [17]. Using systematic bootstrapping as a countermeasure also has efficiency implications as an efficient bootstrapping procedure is not always known for a given FHE scheme and, even when it does (e.g. for TFHE), the current trend in the FHE community is to try to avoid doing it too often. Also a tendency in the parameters usually used in TFHE-related papers is to have a probability of error independent of λ. Nevertheless, modifying the scheme parameters to go from a probability of error of, say, 2^{-80} to 2^{-128} (using the parameters from Sect. C) has a non-negligible impact on the bootstrapping time which increases from 40 to 60 ms (i.e. a 50% increase in computational burden). It is also interesting to note that bootstrapping may result in security improvements, even though it is usually considered as a weakness due to the circular security assumption.

6.3 Monitor&Smudge

As discussed in Sect. 5, for threshold schemes, we have shown that they are inherently subject to CPA^D adversaries. The very large λ-dependent variance smudging noise, which also depends on the ciphertext noise deviation, added on each partial decryption acts as an effective countermeasure against our attack. As a consequence, *noise smudging must not be optional* in FHE threshold scheme implementations. Unfortunately, this also has consequences on the performance of the FHE operators as, when smudging noise is added, one then has to use large enough moduli to ensure reliable decryption. Typically, given a noise deviation budget B (still defined as the standard deviation of the noise after a target algorithm has been executed on a fresh ciphertext, or an upper bound for it),

and following the K-out-of-K decryption procedure of Sect. 5 for e.g. BFV, final ciphertexts include a noise with variance

$$B^2(1 + K2^\lambda), \tag{10}$$

where K is the number of decrypting parties. It follows that q has to be (conservatively) chosen such that[10]

$$q \geq \frac{2tB\sqrt{1 + K2^\lambda}}{\sqrt{\epsilon}}$$

in order to achieve a probability of erroneous decryption of ϵ. Table 4 experimentally illustrates the additional cost of using the larger moduli for a range of multiplicative depths, a number of decrypting parties $K = 5$ and $\epsilon = 2^{-40}$. When smudging is used, ϵ is fixed based on reliability considerations and does not have to be dependent on λ since the smudging noise variance already depends on it. Note that the number of decrypting parties only has a relatively marginal impact on the modulus choice, at least in the small values regime, e.g. a few tens of parties. Lastly, we remark that adding smudging noise before decryption can also serve as a countermeasure to CPA^D attacks in the single-key setup. Comparing the "ratio" columns of Tables 3 and 4 however suggest that in the single key setting the Monitor&Block countermeasure is less costly than Monitor&Smudge.

As a side remark, this CPA^D sensitivity diminishes the interest of using CKKS in the threshold setting, at least when following the blueprint in [19]. Indeed, since CKKS is CPA^D insecure, without smudging, the resulting threshold scheme is not secure (allowing decrypting parties to recover the global key via a CPA^D attack). When smudging is used, the resulting large-variance noise added on each partial decryption is most likely to jeopardize the precision of the post-decryption results. Indeed, stated in LWE terms for simplicity sake, CKKS encryption does not apply any scaling to the message i.e., generates ciphertext $(a, \langle a, s\rangle + m + e)$ rather than $(a, \langle a, s\rangle + \lfloor \frac{q}{2t} \rfloor m + e)$ in the other schemes. As a consequence, CKKS decryption function does not remove the noise term present in the ciphertexts[11]. Hence, when the Monitor&Smudge countermeasure is applied, the decryption function outputs the associated message flooded by a noise of standard deviation in $O(2^{\frac{\lambda}{2}})$ following (10). For instance, on 64 bits cleartext data, the presence of such a large noise would render the FHE computations results unexploitable.

[10] This bound is straightforward to derive from Chebychev inequality analogously to Sect. B. Other less conservative bounds can be obtained.

[11] This has desirable consequences. In particular, as it allows to use smaller parameters, this is one of the reasons why CKKS is very competitive in terms of performances. On the dark side however, this is also the very reason why CKKS was the first scheme deemed CPA^D insecure as the decryption by definition leaks the LWE noise...

d	$\log_2(q)$	n	$\log_2(q)$	n	ratio
1	120	8192	153	8192	1,28
2	180	8192	202	8192	1,12
3	180	8192	258	16384	3,22
4	240	16384	310	16384	1,45
5	240	16384	362	16384	1,79
6	300	16384	414	16384	1,55
7	300	16384	483	32768	3,99
8	360	16384	537	32768	3,70
9	360	16384	591	32768	4,30
10	420	16384	645	32768	3,95

Table 4. Illustration of the performance cost of the Monitor&Smudge countermeasure for OpenFHE/BFV and K-out-of-K decryption, with $K = 5$. The probability of decryption error ϵ is fixed here to 2^{-40}. Column d incidates the target multiplicative depth. The left $\log_2(q)$ and n columns indicate the parameters chosen by OpenFHE for 128 bits security ($t = 65537$). The right $\log_2(q)$ and n columns indicate the parameters required to ensure 128 bits security and to be able to decrypt with a $\epsilon = 2^{-40}$ probability of error when depth k is reached and appropriate smudging noise of st. dev. $B_k\sqrt{K}2^{\frac{\lambda}{2}}$ has been added. The last column indicates the resulting *measured* performance cost ratio over 10000 BFV multiplications.

7 Conclusion

First, let us recall that the FHE schemes which are studied in this paper are all proven secure with respect to the CPA security game *in which the adversary has no access to a decryption oracle*. It is well known, that all the schemes considered in this paper are trivially insecure with respect to the CCA(1) security game. The CPA^D security game with respect to which we define our attack grants the adversary access to a (very constrained) decryption oracle and, as such, grants him or her more power than allowed by the CPA game. As a consequence, there is no contradiction between the CPA security of the schemes considered in this paper and the existence of the attacks that we present.

However, the attacks presented in this paper demonstrate that, contrary to the present state-of-the-art informal status quo, the community working on non-approximate FHE schemes i.e., schemes for which the decryption function includes a mechanism to remove the ciphertext noise, cannot escape taking CPA^D security into account. Furthermore, CPA^D decryption oracles occur naturally in usual application scenarios for FHE. Consider for example an FHE aggregation server involved in a Federated Training protocol for a machine learning model (e.g., [22]) which will eventually be publicly released: it grants the server access to both the encryptions of the model parameters and, after public release, the associated decrypted values. Another example can be a client-server application where the client behaviour may betray decryption errors. Additionally, as we discussed, CPA^D decryption oracle also appear naturally in the threshold variants of FHE schemes which are very important in FHE practice as soon as multiple parties are involved.

Lastly, in the case of the non-approximate FHE schemes considered in this paper, we have shown that there exists a number of natural and simple countermeasures which allow to mitigate CPA^D attacks. These countermeasures essentially consist in ensuring that the adversary has neg(λ) probability of observing exploitable decryption errors. In such a case, the equivalence between CPA and CPA^D security [17] applies. However, these countermeasures have an impact on the parameters of the underlying FHE scheme and, as a result, a negative impact on the FHE calculation performances. For each countermeasure, we have given an idea of that performance impact illustrating the fact that, in qualitative terms, it is not negligible but not prohibitive. Indeed, we observed typical computational cost increases between 50% and 100%. When a larger ciphertext modulus is needed and in order to remain above 128 bits security, n may increase to the next power of 2. In such situations, we observed, in the worst case, an increase of the computational cost of 400%.

We have provided a first set of countermeasures as well as coarse estimates for their impact on schemes parameters and performances. In terms of perspectives, additional work is required to possibly propose more lightweight countermeasures (e.g., requiring for example non systematic bootstrapping for TFHE) and refine and optimize the parameter setting procedures to reduce as much as possible the extra performance cost imputable to CPA^D security in the case of non-approximate FHE.

Acknowledgments

The authors would like to thank Antoine Choffrut for providing key insights on HElib noise upper bound monitoring and blocked decryption mechanisms. The authors would also like to thank Akram Bendouka, Antoine Choffrut and Pierre-Emmanuel Clet for providing feedback that helped improving this manuscrit.

References

1. Lattigo v5. Online: https://github.com/tuneinsight/lattigo (Nov 2023), ePFL-LDS, Tune Insight SA
2. Aloufi, A., Hu, P., Song, Y., Lauter, K.: Computing Blindfolded on Data Homomorphically Encrypted under Multiple Keys: An Extended Survey (Aug 2020), http://arxiv.org/abs/2007.09270
3. Aloufi, A., Hu, P., Song, Y., Lauter, K.: Computing blindfolded on data homomorphically encrypted under multiple keys: A survey. ACM Comput. Surv. (2021). https://doi.org/10.1145/3477139
4. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold fhe. Annual International Conference on the Theory and Applications of Cryptographic Techniques pp. 483–501 (2012)
5. Badawi, A.A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan,

V., Zucca, V.: Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915 (2022)
6. Badawi, A.A., Polyakov, Y.: Demystifying bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/149 (2023)
7. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Advances in Cryptology – CRYPTO 2012. pp. 868–886 (2012)
8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
9. Canetti, R., Fischlin, M.: Universally composable commitments. Annual International Cryptology Conference pp. 19–40 (2001)
10. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology – ASIACRYPT 2017, pp. 409–437. Springer International Publishing
11. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT (2016)
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption library (August 2016), https://tfhe.github.io/tfhe/
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33** (04 2019)
14. Clet, P.E., Boudguiga, A., Sirdey, R., Zuber, M.: Combo: A novel functional bootstrapping method for efficient evaluation of nonlinear functions in the encrypted domain. In: AFRICACRYPT. pp. 317–343 (2023)
15. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. (2012)
16. Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. Tech. Rep. 2021/204, IACR ePrint (2021)
17. Li, B., Micciancio, D.: On the security of homomorphic encryption on approximate numbers. In: EUROCRYPT. pp. 648–677 (2021)
18. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: EUROCRYPT. p. 1–23 (2010)
19. Mouchet, C., Troncoso-Pastoriza, J., Bossuat, J.P., Hubaux, J.P.: Multiparty homomorphic encryption from Ring-Learning-with-Errors. In: PoPETS. pp. 291–311 (2021)
20. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC. pp. 84–93 (2005)
21. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL (Jan 2023), microsoft Research, Redmond, WA.
22. Sébert, A.G., Checri, M., Stan, O., Sirdey, R., Gouy-Pailler, C.: Combining homomorphic encryption and differential privacy in federated learning. In: IEEE PST. pp. 1–7 (2023)

A Threshold for exact noise determination

Let $T = \frac{q}{4}$ and consider a ciphertext with noise $|e|$. Following Sect. 3.2 we have $\alpha_* = \left\lfloor \frac{T}{|e|} \right\rfloor$ and $\alpha_* + 1 = \left\lfloor \frac{T}{|e|} + 1 \right\rfloor$. Condition (4) and (5) can then be respectively rewritten as

$$\frac{T}{\left\lfloor \frac{T}{|e|} + 1 \right\rfloor} \leq |e| < \frac{T}{\left\lfloor \frac{T}{|e|} \right\rfloor}.$$

and

$$\left\lceil \frac{T}{\left\lfloor \frac{T}{|e|} + 1 \right\rfloor} \right\rceil = \left\lfloor \frac{T}{\left\lfloor \frac{T}{|e|} \right\rfloor} \right\rfloor. \tag{11}$$

We are now looking under which condition on $|e|$ the RHS of the above equation is actually equal to $|e|$. Assume $T = k|e| + r$ with $0 \leq r < |e|$, then $\frac{T}{|e|} = k + \frac{r}{|e|}$ and $\left\lfloor \frac{T}{|e|} \right\rfloor = k$. Then,

$$\frac{T}{\left\lfloor \frac{T}{|e|} \right\rfloor} = \frac{k|e| + r}{k} = |e| + \frac{r}{k}.$$

It follows that,

$$\left\lfloor \frac{T}{\left\lfloor \frac{T}{|e|} \right\rfloor} \right\rfloor = |e|$$

if and only if $\frac{r}{k} < 1$ i.e. $r < k$. Since $r < |e|$ this is true when $|e| < k$ and since $k = \left\lfloor \frac{T}{|e|} \right\rfloor$, this is guaranteed to happen when $|e| < \left\lfloor \frac{T}{|e|} \right\rfloor$ i.e. when $|e| < \sqrt{T}$ (indeed, we then have $\frac{T}{|e|} = \left\lfloor \frac{T}{|e|} \right\rfloor + z = |e| + z' + z$ for some positive $z \geq 0, z' > 0$, so that $T > |e|^2$).

As this is a conservative bound on $|e|$, the LHS of Eq. (11) can be used to identify the cases slightly above that bound for which the noise can also be exactly determined.

B BFV module dimensioning for 2^λ CPA^D security

Bound (9) is straightforward to derive from the Chebyshev inequality. Indeed, we want $P(|e| \geq \frac{q}{2t}) \leq 2^{-\lambda}$ where e has mean 0 and standard deviation B. Chebyshev bound tells us that $P(X \geq k\sigma) \leq \frac{1}{k^2}$, so $P(|e| \geq \frac{q}{2tB}B) \leq \frac{4t^2B^2}{q^2}$. Hence we want that $\frac{4t^2B^2}{q^2} = 2^{-\lambda}$ leading to $q = 2tB2^{\frac{\lambda}{2}}$.

For simplicity sake, in order to order to coarsely estimate the noise standard deviation budget required by a given algorithm, we focus only on the multiplicative depth. In BFV, the multiplication operator increases the noise deviation approximately by a factor of $2tn^2\|\mathsf{sk}\|$ [15,16], so, replacing $\|\mathsf{sk}\|$ by $E[\|\mathsf{sk}\|] = \sqrt{\frac{2}{3}n}$, we get a noise deviation around

$$B_k \approx \sigma \left(\sqrt{\frac{8}{3}} t n^{\frac{5}{2}} \right)^k,$$

where σ is the initial noise deviation used for encryption, after chaining k homomorphic multiplications. Hence, following bound (9), to achieve a probability of incorrect decryption below $2^{-\lambda}$ we can choose

$$q \geq 2t2^{\frac{\lambda}{2}} \sigma \left(\sqrt{\frac{8}{3}} t n^{\frac{5}{2}} \right)^k. \tag{12}$$

As an example, if we consider a multiplicative depth of $k = 4$, $t = 65537$, $n = 16384$, $\sigma = 3.2$ and $\lambda = 128$, the above formula tells us that $\log_2(q)$ should be around 289.

C Examples of TFHE bootstrapping parameters

In this section we compare two illustrative parameter sets for TFHE (with binary plaintext) which respectively achieve 2^{80} and 2^{128} bootstrapping error probability with $\lambda = 128$. We remind that in practice, we discretize the Torus either on 32 or 64 bits. That is, TLWE or TRLWE samples will correspond to Regev LWE or RLWE ciphertexts with $q = 2^{32}$ or $q = 2^{64}$. In Table 5, we refer to our chosen discretization of the Torus with the parameter $1/q$.

In Table 5, n denotes the TLWE sample dimension and σ_{TLWE} is the standard deviation of the Gaussian noise. Meanwhile, N and σ_{TRLWE} are respectively the cyclotomic polynomial degree and the standard deviation used for TRLWE noise sampling. In practice, σ_{TLWE} and σ_{TRLWE} are scaled by q. Also, b_{KS} and l_{KS} are the decomposition basis and precision required for key switching during a bootstrapping. Finally, b_{BS} and l_{BS} are the decomposition basis and precision required for gadget matrix decomposition and external products computation within blind rotation during a bootstrapping. Note that the most time consuming building block of TFHE bootstrapping is the blind rotation, and its runtime is quasi-linear in N and linear in n and l_{BS} [13,14].

For generating the parameters in Table 5, we assume that we will be running several consecutive bootstrapping in practice as in Clet et al., [14]. That is, the bootstrapping probability of error will depend partially on the variance of the output of the first bootstrapping.

ϵ	λ	n	$1/q$	σ_{TLWE}	N	σ_{TRLWE}	l_{KS}	b_{KS}	l_{BS}	b_{BS}	Boot. time (ms)
2^{-80}	128	700	2^{-32}	$1.9 \cdot 10^{-5}$	1024	$5.6 \cdot 10^{-8}$	1	1024	2	256	41.86
2^{-128}	128	700	2^{-32}	$1.9 \cdot 10^{-5}$	1024	$5.6 \cdot 10^{-8}$	1	1024	3	64	60.44

Table 5. Example of TFHE parameters achieving ϵ probability of bootstrapping error.