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

Marina Checri, Renaud Sirdey, Aymen Boudguiga, Jean-Paul Bultel and Antoine Choffrut

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

Abstract. In their 2021 seminal paper, Li and Micciancio [9] 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 statu 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 particularily exposed to CPA^D attackers and would be CPA^D-insecure without smudging noise addition after partial decryption. Finally we successfully implement our attack against several mainstream FHE libraries and discuss a number of natural countermeasures and discuss 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.

Preliminary remark: 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.

1 Introduction

Since its inception more than 10 years ago, Fully Homomorphic Encryption has been the subject of a lot of research towards more efficiency and better

* This work was supported by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute.

practicality. From a security perspective, however, FHE still raises a number of questions and challenges. In particular, all the FHE usable in practice, BFV, BGV, CKKS and TFHE, achieve only CPA-security. Although it is well-known that malleability is contradictory with CCA2 security, building efficient FHE construction 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 [9] were the first to study the security of FHE against a slight, seamlessly benign extension of CPA security where the adversary is granted access 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 or even equivalent to CPA security. Unfortunately, Li and Micciancio demonstrated that these intuitions are not true for approximate FHE scheme 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.

This paper is organised as follows: we first recall the CPA^D security game in Sect. 2. In Sect. 3 We then detail 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 praticality on some of the mainstream libraries implementing them. Lastly, Sect. 5 discuss the implications of our attack on existing threshold variants of these cryptosystems. We conclude the paper by a discussion on countermeasures and their implications in terms of FHE efficiency.

2 Background on CPA^D security

In this section, we recall the CPA^D security game [9].

Given an encryption scheme

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

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 $(\texttt{test messages}, m_0, m_1)$, $m_0, m_1 \in \mathcal{P}$ 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 $(\texttt{eval}, f, l_1, \ldots, l_K)$ $(l_i \leq |S|, \forall i)$, 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
$$c' = \mathsf{Eval}(f, S[l_1].c, \ldots, S[l_K].c),$$
and do
$$S := [S; (m'_0, m'_1, c')].$$
- Decryption request: When $\mathcal{A}$ queries $(\texttt{ciphertext}, l)$ $(l \leq |S|)$ 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 and decryption requests): When $\mathcal{A}$ 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 (either produced by an encryption oracle via an encryption request or derived by the evaluation oracle via an evaluation request i.e. derived by correctly applying homomorphic operators to well-formed ciphertexts). As such, the above game thus 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, equivalently in this case, $S[l].m_1$). For exact FHE, this has no impact as, $\mathsf{Dec}_{\mathsf{dk}}(S[l].c) = S[l].m_0 = S[l].m_1$ in this case

(and, as $\mathcal{A}$ learns nothing it does not already knows, CPA^D security coincide with CPA security for exact FHE). For approximate or non-exact FHE, however, 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 FHEs, 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 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 [12]. Withouth loss of generality, we describe only the symetric 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 $\mathbb{Z}_2$ and ciphertexts are elements of $\mathbb{Z}_q^n \times \mathbb{Z}_q$. The algorithms of the scheme are as follows:

- `KeyGen`: pick a (symmetric) secret key $s \in \mathbb{Z}_q^n$ (different probability distributions are possible).
- `Encrypt`: 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 $b = \langle a, s\rangle + \frac{q}{2}m + e$.
- `Decrypt`: given a ciphertext $\mathrm{ct} = (a; b)$, 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', ciphertext $(a + a'; b + b')$ is an encryption of $m + m'$, provided the total error 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 its 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 $(\texttt{eval}, \mathsf{sum}, \mathsf{idx}(c_0), \mathsf{idx}(c_0))$, $(\texttt{ciphertext}, \mathsf{idx}(c_1))$, $(\texttt{eval}, \mathsf{sum}, \mathsf{idx}(c_1), \mathsf{idx}(c_1))$, $(\texttt{ciphertext}, \mathsf{idx}(c_2)), \ldots, (\texttt{eval}, \mathsf{sum}, \mathsf{idx}(c_{k-1}), \mathsf{idx}(c_{k-1}))$, $(\texttt{ciphertext}, \mathsf{idx}(c_k))$, $\mathcal{A}$ can obtain ciphertexts $c_1 = c_0 + c_0$, ..., $c_k = c_{k-1} + c_{k-1}$ where

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

and decrypt them, stopping either when c_{k-1} decrypts to 0 and c_k decrypts to 1, in which case $\mathcal{A}$ can conclude that

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

or when c_k still decrypts to 0 when $2^k > \frac{q}{4}$, in which case she can conclude that $|e| = e = 0$ and gets one linear equation in s (the probability of this happening depending on the noise distribution).

The parameter α_*. Now, unless c_0 is identified as noise free, given $1 \leq \alpha < \frac{q}{4}$ 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 adversary can then 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

$$\frac{q}{4(\alpha_* + 1)} \leq |e| < \frac{q}{4\alpha_*}. \tag{3}$$

Determination of $|e|$. Then $|e|$ is uniquely determined when $\left\lceil \frac{q}{4(\alpha_*+1)} \right\rceil = \left\lfloor \frac{q}{4\alpha_*} \right\rfloor$ and this occurs (Sect. A) when

$$|e| < \frac{\sqrt{q}}{2}\,. \tag{4}$$

So the bottom-line, at this point, 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. Then ciphertexts with noise below $\sqrt{q/4} = 2^{15}$ will have their noise completely determined. A

"back of the envelope" calculation tells us that this will be the case for around 20% of the ciphertexts. So, as an order of magnitude, with the above parameter set, the attack will need to work over around $1.5 \times n/0.2$ fresh encryptions of 0 to achieve a full key recovery. For BFV, BGV, the tendency generally is to use quite large moduli e.g., 2^{200}, and small variance e.g., $\sigma = 3.2$. With this kind of parameters condition (4), is satisfied with overwhelming probability.

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 c_0 is then determined 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 Sect. If this latter ciphertext decrypts to 1, she can conclude that e and e' have the same sign (as $\alpha_* e$ 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 section $\mathcal{A}$ 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. 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 from the CPA^D game as in Sect. 3.3 (note that noise-free ciphertexts 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, ..., c_N with known noise magnitudes $|e_1|$, ..., $|e_N|$ (determined from the CPA^D game 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 our candidate secret key (with the correct one always leading 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 $(\texttt{test messages}, 0, 1)$.

3.5 First experimental results

Table 3.5 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, P_0 the proportion of ciphertexts for which $|e| = 0$, the other columns respectively provide the number of encryption, evaluation and decryption requests done to achieve secret key recovery. Following these results, we are now ready to attempt implementing the attack against real-world libraries. We do so in Sect. 3.

n	q	σ	λ	P	P_0	# enc	# eval	# dec
636	2^{32}	2^{17}	97	0.235	ε	5358	150437	150437
1024	2^{32}	2^{17}	175	0.226	ε	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 $P = 1$) since the condition that $|e| < \sqrt{q}/2$, see (3), is satisfied with overwhelming probability.

3.6 Adaptation to RLWE

Let us consider an elementary RLWE variant of Regev working over $\mathbb{Z}_q[X]/(X^N + 1)$ [10], where N is a power of 2. Encryption: given $m_0, ..., m_{N-1} \in \mathbb{Z}_2^N$ return $(a; b)$ where $b = a.s + \frac{q}{2}p_m + e$ where $p_m(X) = \sum_{i=0}^{N-1} m_i X^i$. Decryption: given $(a; b)$, return $\left\lceil \frac{2}{q}(b - a.s) \right\rfloor \mod 2$ where the rounding is performed coefficient-wise. Essentially, this RLWE variant consists in grouping N LWE pairs in a single ciphertext as the i-th $(i = 0, ..., N-1)$ coefficient of polynomial b is

$$\sum_{j=0}^{i} a_{i-j} s_j - \sum_{j=i+1}^{N-1} a_{i+N-j} s_j + \frac{q}{2} m_i + e_i. \tag{5}$$

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

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 p_0 via a request of the form $(\texttt{test messages}, p_0, p_0)$. $\mathcal{A}$ then gets $c_0 = (a; a.s + e)$, the polynomial e being of course unknown to her. Similarly to the LWE case, by means of CPA^D game requests of the form $(\texttt{eval}, \mathsf{sum}, \mathsf{idx}(c_0), \mathsf{idx}(c_0))$, ..., $(\texttt{eval}, \mathsf{sum}, \mathsf{idx}(c_{k-1}), \mathsf{idx}(c_{k-1}))$, $\mathcal{A}$ can obtain ciphertexts $c_1 = c_0 + c_0$, ..., $c_k = c_{k-1} + c_{k-1}$ where

$$c_k = (2^k a; 2^k a.s + 2^k e).$$

stopping 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 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 when $2^k > \frac{q}{4}$, in which case she can conclude that $|e_i| = e_i = 0$ and straight away obtain one linear equation in s following (5):

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

The rest of the attack works similarly to the LWE case, performing RLWE CPA^D game requests until she gets N ciphertexts $c^{(1)}$, ..., $c^{(N)}$ with known noise magnitudes $|e_i^{(1)}|$, ..., $|e_i^{(N)}|$ and same noise signs *in the i-th coefficient of their noise polynomials*. She can then postprocess 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 equations for a given such ciphertext. Indeed, given a RLWE encryption of the null polynomial $(a; a.s + e)$, the above noise absolute value determination procedure can be repeated independently for each coefficient index i of the RHS polynomial of the RLWE pair, yielding, when condition (3) holds for this index, $|e_i|$. To determine, whether

the noises e_i and e_j for two coefficients $i < j$, 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 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. We assume the reader is already familiar with these cryptosystems.

4.1 BFV

With BFV[3,8], to encrypt a polynomial message m with the public key $\mathbf{pk} = (p_0; p_1) = \left([-(a \cdot sk + e)]_q ; a\right) \in {R_q}^2$, one must sample $r \xleftarrow{\$} R_2$, $e_0, e_1 \xleftarrow{\$} \chi$ and return $\mathbf{c} = \left([\Delta \cdot m + p_0 \cdot r + e_0]_q ; [p_1 \cdot r + e_1]_q\right)$, where $\Delta = \lfloor \frac{q}{t} \rfloor$. An encryption of 0 is then $\mathbf{c} = (c_0; c_1) = \left([p_0 \cdot r + e_0]_q ; [p_1 \cdot r + e_1]_q\right)$. Thus, modulo q we have

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

If we only select one coefficient of the polynomial ciphertext, we obtain an LWE instance and can proceed with the attack 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. We choose to use the BFV cryptosystem with parameters giving a security of 128 or 256 bits. To carry out the attack, we need to access the polynomial's coefficients before encryption and after decryption, which is easily done with this library, which overloads the `[ ]` operator in C++. The practical attack follows exactly the theoretical attack: we try to estimate the noise of the ciphertext, which can be exactly recovered if the condition $|e| < \frac{\sqrt{q}}{2}$ from (3) is verified. This condition is almost always satisfied with the default BFV parameters, which uses a q that is very large compared to the error (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 $\sigma = 3.2$).

[2] https://github.com/microsoft/SEAL

When carrying out the attack, we (almost) systematically find the noise of the ciphertext with the dichotomy. To check that we correctly recover the absolute value of the noise, we display the true noise of this ciphertext - a normally prohibited action - by modifying the SEAL library. Note that this modification is not used for the attack itself and only allows us to check that the attack finds the correct absolute noise value.

We ran the attack with the 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 in about 1 m 20 s. The number of calls to the encryption oracle was on average 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.

OpenFHE In order to compare the libraries, we also attempt the attack on BFV as implemented in OpenFHE[3] with the default security given for 128- and 256-bit. As described previously, we need to recover the coefficients before encryption and after decryption. This operation in OpenFHE is less trivial than in SEAL. As described in the library, OpenFHE uses three main data classes for polynomial representations, namely:

- `Poly` - a single-CRT representation (`BigInteger` types as coefficients, supporting a large modulus q);
- `NativePoly` - a single-CRT representation (`NativeInteger` types limiting the size of the coefficients, modulus q to 64 bits);
- `DCRTPoly` - a double-CRT representation.

The encoding used for polynomials is `COEF_PACKED_ENCODING`, which allows us to recover the `DCRTPoly` underlying the freshly created plaintext (before encryption) and then interpolate it to get a `Poly`, then get one of the coefficients of the polynomial. After encryption and decryption, the decrypted plaintext no longer has a `DCRTPoly` but a `NativePoly`: we only then have to extract the coefficient of interest.

To run the attack on BFV with the OpenFHE library, we choose parameters generated by default by the library, ensuring the 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 used 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. 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.

[3] https://github.com/openfheorg/openfhe-development

4.2 BGV

For BGV[4], 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 $e_1, e_2 \xleftarrow{\$} \chi$ then create a level-L BGV ciphertext $\mathbf{c} = (u \cdot pk_{L,0} + m + p \cdot e_0; u \cdot pk_{L,1} + p \cdot e_1)$, where the public key at level L is $\mathbf{pk_L} = (pk_{L,0}; pk_{L,1}) = (a_L \cdot sk_L + p \cdot e_L; -a_L)$. A level-$\ell$ encryption of 0 is $\mathbf{c} = (c_0; c_1) = (u \cdot pk_{\ell,0} + m + p \cdot e_0; u \cdot pk_{\ell,1} + p \cdot e_1)$. Then,

$$\begin{aligned} -c_0 &= -(a_\ell \cdot sk_\ell + p \cdot e_\ell) \cdot u + p \cdot e_0 \\ &= -(a_\ell \cdot u) \cdot sk_\ell + p \cdot e_\ell \cdot u + p \cdot e_0 + p \cdot e_1 \cdot sk_\ell - p \cdot e_1 \cdot sk_\ell \\ &= c_1 \cdot sk_\ell - (p \cdot e_\ell \cdot u + p \cdot e_0 + p \cdot e_1 \cdot sk_\ell) \\ -b' &= a' \cdot sk_\ell + e'. \end{aligned}$$

As for the BFV case, we can focus on only one coefficient of the polynomial ciphertext to obtain an LWE instance and go back to the Regev case.

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 (almost) systematically found the absolute noise's value and were thus able to recover the linear equations required for the attack.

To attack the OpenFHE implementation of BGV, 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 approximately 3475514 while that of the encryption oracle is 32779. We present these results in Table 2.

HELIB HElib[4] 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.

Our attack requires to generate ciphertexts which actually fail to correctly decrypt before they are flagged by Ctxt::isCorrect(). But the parameters in HElib are so conservative that the probability of this happening is negligible. The

[4] https://github.com/homenc/HElib

implication is that one cannot determine the (additive) depth k on ciphertexts, as in (2), let alone the value of α_* defined in (3). As a result, HElib prevents us from extracting even the magnitude of the noise e in the initial ciphertext c_0.

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 levels in ciphertexts and choosing instance parameters rendering the probability of incorrect decryption negligible.

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

4.3 TFHE

The TFHE encryption scheme was proposed in 2016 [6] and updated in [7]. 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 [2].

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

Let $\mathcal{M} \subset \mathbb{T}$ be the discrete message space[5]. To encrypt a message $m \in \mathcal{M} \subset \mathbb{T}$, we add $(\boldsymbol{0}; m)$ to a fresh TLWE sample $(\boldsymbol{a}; b)$, to obtain the ciphertext $\boldsymbol{c} = (\boldsymbol{a}; b + m)$. In the following, we refer to an encryption of m with the secret key $\boldsymbol{s}$ as a TLWE ciphertext noted $[\![m]\!] = \boldsymbol{c} \in \mathrm{TLWE}_{\boldsymbol{s}}(m)$.

To decrypt a sample $\boldsymbol{c} \in \mathrm{TLWE}_{\boldsymbol{s}}(m)$, we compute its *phase* $\phi(\boldsymbol{c}) = b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle = m + e$. Then, we round to 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 $\boldsymbol{c_1} \in \mathrm{TLWE}_{\boldsymbol{s}}(m_1)$ to $\boldsymbol{c_2} \in \mathrm{TLWE}_{\boldsymbol{s}}(m_2)$, we get $\boldsymbol{c} \in \mathrm{TLWE}_{\boldsymbol{s}}(m_1 + m_2)$.

TFHELib TFHELib[6] is an open source library providing the original TFHE implementation. It supports two discretization for $\mathbb{T}$ either on 32 or 64 bits[7]. That is, in practice, a TLWE sample corresponds to a Regev encryption with

[5] In practice, we discretize the Torus with respect to our plaintext modulus. For example, if we want to encrypt $m \in \mathbb{Z}_4 = \{0, 1, 2, 3\}$, we encode it in $\mathbb{T}$ as one of the following value $\{0, 0.25, 0.5, 0.75\}$.

[6] https://tfhe.github.io/tfhe/

[7] 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.

$q = 2^{32}$ (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 implement this attack within TFHELib and we succeed, in less than 1 second, in finding the number of linear equations required for key recovery (as described in Table 2).

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						
SEAL	BFV	95	4096	109	3.2	1	6250/232858	≈ 7979	4096	$\approx 2'50''$	≈ 927163
	BFV	227	4096	58	3.2	1	1481/860557	≈ 7393	4096	$\approx 1'20''$	≈ 664138
OpenFHE	BFV	128	8192	120	3.19	1	69/48929	≈ 16235	8192	$\approx 19'30''$	≈ 3357143
	BFV	256	16384	120	3.19	1	173/130535	≈ 32700	16384	$\approx 75'30''$	≈ 6630736
	BGV	128	8192	69	3.19	1	59/32811	≈ 16405	8192	$\approx 18'30''$	≈ 1690360
	BGV	256	16384	71	3.19	1	80/65559	≈ 32779	16384	$\approx 68'50$	≈ 3475514
HElib	BGV	187	8096	108	3.2	-	-	-	-	-	-
TFHELib	TFHE	97	630	32	2^{17}	952/4206	0	4206	630	$0.201s$	156396
	TFHE	128	1024	32	81604.378	1518/4273	0	4273	1024	$0.267s$	392293
	TFHE	128	1024	32	279.172	1536/1548	7/1548	1548	1024	$0.130s$	98436

Table 2. Experimental results - Summary

5 Remarks on threshold homomorphic encryption

One of the existing multi-user approaches is the Threshold (also called Multi-Party) Homomorphic Encryption. It allows users to encrypt their data using a joint public key, constructed from their individual public keys. The decryption phase is collaborative, so no user holds the associated global private key. To keep the secrecy of their individual private keys, no user should be able to extract information on another user's or the joint private key from the public 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 (encrypted under the global secret key that he or she does not know) and, at the end of the collaborative decryption protocol, 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, CPA^D security is a must-have in the threshold setting.

5.1 The case of Lattigo

Now, we illustrate this with the N-out-of-N threshold scheme from [11] 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 [11], which are standard in the context of RLWE-based schemes.

Let CRS be the uniform distribution in R_q, according to a common random string, i.e., elements sampled from this distribution are uniformly distributed and the same for all N 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 (see [5] for more information about the CRS model).

This scheme is a tuple MBFV = (`EncKeyGen`, `RelinKeyGen`, `KeySwitch`, `PubKeySwitch`) that extends the BFV scheme:

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

Each party P_i constructs its individual public key $p_{0,i} \coloneqq -p_i s_i + e_i$ from its individual private key s_i. The joint public key $\texttt{EncKeyGen}(\mathrm{sk}_1, \ldots, \mathrm{sk}_N)$ is

then $(p_0; p_1)$ with $p_0 = \sum_{P_j \in \mathcal{P}} p_{0,j}$. There is no need to detail how the joint re-linearization key is constructed in our context.

To perform a collaborative key switch $\texttt{KeySwitch}(\mathrm{ct}, \mathrm{sk}'_1, \ldots, \mathrm{sk}'_N, \mathrm{sk}_1, \ldots, \mathrm{sk}_N)$ on a ciphertext

$$\mathrm{ct} = (c_0; c_1) = (m + up_0 + e_0; up_1 + e_1),$$

each party P_i (owner of the secret keys s_i and s'_i) picks a smudging noise e_i and discloses

$$h_i := (s_i - s'_i)c_1 + e_i$$

to the other parties. The smudging noise is sampled from a discrete Gaussian distribution with (*very large*) variance $\sigma^2_{\mathrm{smg}} = 2^\lambda \sigma^2_{\mathrm{ct}}$, where σ^2_{ct} is the ciphertext's noise variance (which is monitored during FHE evaluation) and λ is the desired security level. The output re-encrypted ciphertext $\mathrm{ct}' := (c_0 + \sum_{P_j \in \mathcal{P}} h_j; c_1)$ can then be computed by any party. Finally, there is no need to detail the public key switch protocol `PubKeySwitch` here.

Now, observe that the key switching protocol `KeySwitch` with private inputs $(s_i; 0)$ (that is, $s'_i = 0$ for all party i) is actually a collaborative decryption. To perform this collaborative decryption, each party P_i picks a smudging noise e_i and discloses $h_i := s_i c_1 + e_i$ to the other parties. The decryption is then performed by any 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 $s := \sum_{P_i \in \mathcal{P}} s_i$, which is not known by any unique party. Hence, any party is actually a potential CPA^D attacker. 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 private key s that she does not know.

The smudging technique, as introduced by Asharov et al. [1], aims at making the ciphertext-noise unexploitable as a side-channel by flooding it with some freshly sampled noise terms in a distribution of larger-variance, as mentionned in [1]. Characterizing the side-channel leakage in this setting is presented as an open problem in [11]. Now, the smudging technique is an efficient countermeasure against our attack, without which the threshold scheme would be vulnerable to our attack (as long as the base FHE scheme is not CPA^D secure). Hence, smugding noise addition should then be used in a systematic way and *not be made optional* by any library implementing the threshold homomorphic encryption scheme specified in [11].

5.2 Other threshold schemes

There is a variant of the scheme in [11] with a fixed parameter $T < N$, in which any T parties among the N ones are enough to perform a collaborative decryption. In this variant, the individual keys are constructed by means of Shamir's secret sharing [13], 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 Concluding remarks on mitigation

In this section, we propose several countermeasures.

Monitor&block. Inspired by HElib's strategy (Sect. 4.2), one possible countermeasure is as follows. Given a noise budget B, the cryptosystem parameters shoud be chosen such that a B-noisy ciphertext has probability $\mathrm{neg}(\lambda)$ to generate a decryption error. Then ciphertext noise bounds must be monitored by the homomorphic operators implementation and the decryption function has to provide no output for ciphertexts with noise beyond the noise budget B. Note that these mechanisms thus become part of the cryptosystem specification and, as such, a (passive) CPA^D attacker cannot work around them. This has implications on the choice of modulus as $\frac{q}{t} \geq 2^\lambda B$ (hence on the choice of n and hence on the efficiency of the homomorphic operations).

Bootstrap. Another natural strategy is to bootstrap after each homomorphic operation. Since bootstrapping resets the noise to a preset value, decryption errors cannot occur anymore and the scheme becomes exact (with probability $1-\mathrm{neg}(\lambda)$). In this case, CPA security and CPA^D security are known to be equivalent. Using systematic bootstrapping as a countermeasure also has efficiency implications as an efficient bootstrapping procedure does not always exist (e.g. for BFV and BGV) and, when it does (e.g. for TFHE), the current trend in FHE research is to try to avoid doing it too often. It is also interesting to note that bootstrapping may thus have (positive) security implications when it is usually considered a (slight) weakness due to the circular security assumption.

Monitor&smudge. For threshold schemes, we have shown that they are inherently subject to CPA^D adversaries. The (very large variance) smudging noise, which also depends on the ciphertext noise bound, added on each partial decryption acts as an effective countermeasure against our attack and allows to build secure threshold schemes from CPA secure and (possibly) CPA^D insecure schemes. As a consequence, *noise smudging must not be optional* in FHE threshold scheme implementations. As a side remark, this diminishes the interest of using CKKS in the threshold setting. 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) or, when smudging is used, the resulting large noise jeopardizes the precision of the postdecryption results.

References

1. 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)
2. Badawi, A.A., Polyakov, Y.: Demystifying bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/149 (2023), https://eprint.iacr.org/2023/149, https://eprint.iacr.org/2023/149
3. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Advances in Cryptology – CRYPTO 2012. pp. 868–886. Springer Berlin Heidelberg (2012)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
5. Canetti, R., Fischlin, M.: Universally composable commitments. Annual International Cryptology Conference pp. 19–40 (2001)
6. 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.) Advances in Cryptology – ASIACRYPT 2016. pp. 3–33. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
7. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33** (04 2019). https://doi.org/10.1007/s00145-019-09319-x
8. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. (2012)
9. Li, B., Micciancio, D.: On the security of homomorphic encryption on approximate numbers. In: EUROCRYPT. pp. 648–677 (2021)
10. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. p. 1–23 (2010)
11. 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)
12. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC. pp. 84–93 (2005)
13. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)

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 (3) can then be rewritten as

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

We are now looking under which condition on $|e|$ we have

$$\left\lceil \frac{T}{\left\lfloor \frac{T}{|e|} + 1 \right\rfloor} \right\rceil = \left\lfloor \frac{T}{\left\lfloor \frac{T}{|e|} \right\rfloor} \right\rfloor. \tag{6}$$

We first deal with the RHS. 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 $|e| < \sqrt{T}$.

As this is a conservative bound on $|e|$, the LHS of Eq. (3) can be used to identify the cases slightly above that bound for which the noise can also be exactly determined.