Error-Simulatable Sanitization for TFHE and Applications

Nigel P. Smart[1,2] and Michael Walter[2]

[1] KU Leuven, Computer Security and Industrial Cryptography, Leuven, Belgium
[2] Zama, Paris, France

Abstract. We show that the randomized TFHE bootstrapping technique of Bourse and Izabechéne provides a form of sanitization which is error-simulatable. This means that the randomized bootstrap can be used not only for sanitization of ciphertexts (i.e. to hide the function that has been computed), but that it can also be used in server-assisted threshold decryption. Thus we extend the server-assisted threshold decryption method of Passelégue and Stehle (ASIACRYPT '24) to FHE schemes which have small ciphertext modulus (such as TFHE). In addition the error-simulatable sanitization enables us to obtain FuncCPA security for TFHE essentially for free.

Keywords: Multi-Party Computation · Fully Homomorphic Encryption

Contents

E-mail: nigel.smart@kuleuven.be,nigel@zama.ai (Nigel P. Smart), michael.walter@zama.ai (Michael Walter)

1 Introduction

The problem of threshold decryption for FHE schemes, henceforth called threshold-FHE, is as old as FHE itself. The problem is for a set of $\mathfrak{n}$ parties to have a secret sharing of the underlying FHE secret key, so that they can, between them, decrypt a given FHE ciphertext correctly, in the case where at most $\mathfrak{t}$ of the parties are corrupt. Indeed, Gentry's original thesis [Gen09] mentioned threshold-FHE as a way of utilizing FHE to perform a very low round complexity semi-honest MPC protocol. Since Gentry's thesis a number of FHE schemes have been proposed including BGV [BGV12], CKKS [CKKS17], BFV [Bra12, FV12] and TFHE [CGGI16, CGGI20]. In this work we primarily focus on the TFHE scheme.

At about the time of Gentry's thesis on FHE in 2009 [Gen09], the first threshold key generation and decryption for LWE based ciphertexts was also given by Bendlin and Damgård [BD10]. The methodology of Bendlin and Damgård used replicated secret sharing[1] to split the secret key, a method which requires the numbers of shares per party to grow as $O(\binom{\mathfrak{n}}{\mathfrak{t}})$. The simpler case of full-threshold, i.e. $\mathfrak{t} = \mathfrak{n} - 1$, decryption for LWE ciphertexts was combined with SHE and formed the basis of the SPDZ MPC protocol [DPSZ12]. The same techniques were then used in the context of FHE by Asharov et al [AJL+12] in the full threshold setting. All of [AJL+12, BD10, DPSZ12] consider a threshold decryption process which was only semi-honestly secure; i.e. the adversary could send in invalid shares resulting in an invalid threshold decryption.

Such semi-honest security is not a problem in the applications considered in [AJL+12, DPSZ12], as the higher level protocols can deal with active adversaries (when needed) by adopting other defensive measures. However, for some applications one requires robust, a.k.a. guaranteed output delivery (GOD), protocols for which any adversarially introduced errors can be overcome by the honest parties executing the threshold decryption process. In [CLO+13] a robust threshold decryption protocol was presented for the threshold setting of $\mathfrak{t} < \mathfrak{n}/3$ and the BGV scheme. The work of [CLO+13] was extended to the TFHE scheme in [DDK+23]. This later paper also optimized the threshold decryption process for BGV, BFV and TFHE, as well as discussing the case of asynchronous networks, and other extensions.

To understand the technical problem with threshold-FHE it is worth considering the "format" of a simple FHE scheme. To explain we utilize the format of BFV/TFHE [FV12, CGGI16, CGGI20] ciphertexts, but a similar discussion can be provided for other FHE schemes such as BGV. Consider encrypting an element $m \in \mathbb{Z}/(p)$, using a standard Learning-With-Errors (LWE) ciphertext of the form $(\underline{a}, b)$ with ciphertext modulus q , where $\underline{a} \in (\mathbb{Z}/(q))^\ell$ and $b \in \mathbb{Z}/(q)$, using the equation

$$b = \underline{a} \cdot \underline{s} + e + \Delta \cdot m \pmod{q}$$

where $\Delta = \lfloor q/p \rfloor$, e is some "noise" term and $\underline{s} \in (\mathbb{Z}/(q))^\ell$ is the secret key. Usually, in the FHE setting, $\underline{s}$ is chosen to be a vector with small norm, for example $\underline{s} \in \{0,1\}^\ell$.

To enable threshold-FHE we first secret share the secret key $\underline{s}$ among $\mathfrak{n}$ parties, a process which we shall denote by $\langle \underline{s} \rangle$ to signal a sharing modulo q with respect to a threshold $\mathfrak{t} < \mathfrak{n}$ linear secret sharing scheme. On input of the ciphertext $(\underline{a}, b)$ we can then locally produce a secret sharing of the value $e + \Delta \cdot m$ by computing

$$\langle p \rangle = b - \underline{a} \cdot \langle \underline{s} \rangle = \langle \Delta \cdot m + e \rangle.$$

We shall call the value $p = \Delta \cdot m + e$ the "pre-decryption" value. By opening the value of $\langle p \rangle$, all parties can then perform rounding to obtain m. However, this reveals the value of

[1] Note, a generic thresholdizer for arbitrary protocols was given by Boneh et al. in [BGG+18] using threshold-FHE. The construction of Boneh et al. utilizes a special form of secret sharing called $\{0,1\}$-LSSS, which is closely related to replicated sharing; and thus does not scale to more than a few players.

e, which combined with the ciphertext and the message, will reveal information about the secret key $\underline{s}$.

The prior literature provides four major ways around this problem. The first is to add a large amount of additional noise into the secret sharing before opening (a process called noise flooding), the second is to apply techniques based on Renyi divergence, the third method is to extract the message m from the shared value $\langle p \rangle$ using a generic MPC protocol. A relatively new fourth method is to utilize so-called server assisted threshold decryption. All of these are troublesome in practice. The first method requires an increase in the underlying ciphertext modulus q (in the case of TFHE) or a decrease in the number of available levels (in the case of BGV/BFV), the second results in the loss of simulation security, the third method requires relatively high round protocols, resulting in poor performance over a Wide-Area-Network (WAN), whilst the fourth requires a super-polynomial ciphertext modulus to exist in the first place. We now elaborate on these four methods in more detail.

Threshold Decryption via Noise Flooding:

In this method the decrypting parties somehow generate an additional secret shared noise term $\langle E \rangle$, and the pre-decryption value which is opened is now

$$\langle p \rangle = b - \underline{a} \cdot \langle \underline{s} \rangle + \langle E \rangle = \langle \Delta \cdot m + e + E \rangle.$$

We require that E should introduce enough randomness to mask the e value after the shared value $\langle p \rangle$ is opened. If E is too small then too much information about e is revealed, if E is too big then the final rounding will not reveal the correct value of m. Diagrammatically we can consider this process as approximated by the diagram in Figure 1.

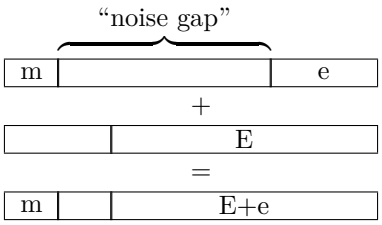

Figure 1: Representation of the noise addition for threshold decryption

To mask, statistically, all information in e we would (naively) require E to be chosen uniformly from a range which is 2^{dist} larger than e. Thus if we can bound the ciphertext noise by $|e| < B$, then we would require E to be chosen uniformly in the range $[-2^{\mathsf{dist}} \cdot B, \ldots, 2^{\mathsf{dist}} \cdot B]$. This process is dubbed "noise flooding" in the literature. However, this would mean we require $\Delta > 2^{\mathsf{dist}} \cdot B$, which in turn seems to imply that the ciphertext modulus q needs to be "large". The work effort of the adversary to distinguish the distributions in the underlying security proof is then 2^{dist}. Prudently this would lead us to select $\mathsf{dist} = 80$.

In [DDK+23] it is shown that by adding an E term on, which is itself the addition of at least two uniform distributions in the range $[-2^{\mathsf{stat}} \cdot B, \ldots, 2^{\mathsf{stat}} \cdot B]$, the resulting work effort of the adversary to distinguish the distributions becomes $2^{2 \cdot \mathsf{stat}}$. We can, hence, obtain enough security by selecting $\mathsf{stat} \approx 40$, and so reduce the need for very much larger q values. The main benefit is that for "small" values of $\binom{n}{t}$ (say less than $10,000$) one obtains a simple one round threshold decryption protocol which works over asynchronous networks. When $\binom{n}{t}$ is "large" the method in [DDK+23] uses a one-round online phase, but requires an offline phase to pre-prepare secret shared random bits (something which can be done via generic MPC protocols).

When using BFV/BGV in an SHE leveled mode the problem of needing a larger q does not occur. In such schemes each level essentially adds an extra 20-40 bits (depending on the implementation) into the noise gap. Thus by simply decreasing the number of usable levels by a small constant (say, one or two) one can obtain a noise gap which is enough to apply the flooding technique. Thus in such schemes this methodology can be applied, without any need for prior processing of a ciphertext.

When using BFV/BGV in a mode which enables the use of bootstrapping we also do not need to increase the size of q. Bootstrapping enables us to reduce the size of the noise e in the ciphertext $(\underline{a}, b)$ to be as small as possible. Thus if bootstrapping is performed, and the FHE scheme is such that the noise gap between e and m in Figure 1 is large enough, then the noise flooding methodology will work "out-of-the-box".

Thus the only place where noise flooding is in practice a problem is when the FHE parameters are such that the noise gap is tiny, even after a bootstrapping operation is performed. This is exactly the situation in TFHE, where one (usually) selects a relatively small q value (for example $q = 2^{64}$). This small q value, and associated small LWE dimension ℓ, requires the size of the noise even after bootstrapping to be around 2^{30} in order to ensure security. This means the noise gap is too small, but only by tens of bits. In such a situation we can apply, following the method of [DDK+23], a switch of the parameters (q, ℓ) to slightly larger ones (q', ℓ') with $q' = 2^{128}$, and then perform a large (and hence expensive) bootstrapping operation to squash the noise down; this technique the authors of [DDK+23] dub SwitchSquash.

Threshold Decryption via The Renyi Divergence:

Even with the optimizations of [DDK+23] we still require a super-polynomial gap between the bound on the noise term e and the ciphertext modulus, q or q'. Such super-polynomial blow-ups in other areas of cryptography based on LWE have recently been avoided by utilizing the Renyi divergence [BLR+18]. This, as an approach to threshold-FHE, was recently examined by [BS23] and [CSS+22]. The problem with using the Renyi divergence in the context of distributed decryption, is that the general technique of Renyi divergence is hard to apply to security problems which are inherently about distinguishing one distribution from another. In [BS23] and [CSS+22] a way around this was found by designing special security games for threshold-FHE usage, which enabled the use of the Renyi divergence. The problem is that these games need to cope with the homomorphic nature of the underlying encryption scheme, and thus cannot be adaptive. In applications of threshold FHE we really require a threshold-FHE protocol which is indistinguishable, to an adversary, from a simulation interacting with an ideal functionality. The security games presented in [CSS+22] and [BS23] do not allow such a usage. However in [PS24] it is shown that the methods based on Renyi divergence, such as [BS23], have additional problems with their security. Indeed currently they can only obtain selective security and not full blown adaptive simulation based security.

Threshold Decryption via Generic MPC:

Another approach is to apply generic MPC to the problem of threshold decryption of FHE ciphertexts. This method is most suited for TFHE, as (as we remarked earlier) the noise-gap issue is not really an issue for BGV and BFV. In this method we take the sharing of the pre-decryption directly, i.e. we compute

$$\langle p \rangle = b - \underline{a} \cdot \langle \underline{s} \rangle.$$

One then needs to extract the message m from the pre-decryption. The message is encoded in p, for TFHE, in the following manner:

$$p = \Delta \cdot m + e \pmod{q}.$$

As for TFHE one usually has that both q and $\Delta = q/p$ is a power of two, one can extract the message m via secret sharing based bit-decomposition techniques. Such generic MPC bit-decomposition methods are well known, see for example [CdH10, DFK+06, NO07, sec09]. Thus using generic MPC, to perform the threshold decryption, we do not need to require any noise gap. Thus there is no need for any switch to larger parameters, or a large bootstrapping to produce a large noise gap.

The disadvantage of this method is that we really need to apply the full power of our MPC engine. The use of such protocols means that threshold decryption requires a relatively large number of "online phase" rounds (if $q = 2^{64}$ as in TFHE, then this is 16 rounds). In addition, one requires an "offline phase" to generate sets of secret shared random bits. Thus this method whilst not requiring the large bootstrap, may actually be slower, if the parties are separated by a network which requires a large ping-time.

Server Assisted Threshold Decryption:

A method to achieve simulation security for threshold LWE without using noise flooding or generic MPC, was described in [MS25]. However, the technique of [MS25] could only be applied to traditional LWE public key encryption schemes, i.e. for schemes which do not allow any form of homomorphic operations. In recent work, [PS24], this method was extended to the case of threshold FHE. The technique works in two phases; in the first phase (executed by a trusted server) the ciphertext is flooded with a large amount of noise, and then the ciphertext is reduced in size via Gaussian rounding. In the second phase the method of [MS25] is essentially applied, which requires only a small amount of noise flooding. The authors dub there method "double flood and round", although, since the second application of flooding is via a small amount of noise, one could better dub it "flood, round and bath".

The need for a trusted server to compute the initial flooding is due to the need for the randomness used in the first (large) flooding to not be exposed to the threshold decryption parties. This could produce a major system problem, but it also imposes a constraint on the parameters; namely the ciphertext modulus to apply the server flooding needs to be super-polynomially large. This means the method, as given in [PS24], only applies to FHE schemes such as BGV, BFV and CKKS.

Another issue with the presentation in [PS24], is that the scheme is only suitable for full threshold access structures. This means that the threshold decryption can only be passively secure, and one looses robustness.

1.1 Our Contribution

Our core contribution is a new technical result on the ciphertext sanitization strategy given in [BI22, BdPMW16]. Ciphertext sanitization for FHE schemes was introduced by Ducas and Stehle in [DS16]. Unlike in threshold decryption, where we are worried about the noise term e in the pre-decryption value p revealing information about the secret key, in sanitization we are worried about e revealing information about the function which has been homomorphically evaluated. Thus the two situations seem similar, except that in sanitization we are usually trying to protect against an adversary which already knows the secret key.

The papers [BI22, BdPMW16] introduce a sanitization algorithm that is suitable for small parameters, but, as opposed to the generic algorithm given in [DS16], requires only one bootstrapping. We present a way of mapping the sanitization method of [BI22, BdPMW16], in particular the first paper, over to our situation. In particular, we define a sanitization algorithm $\mathfrak{ct}' \leftarrow \mathsf{Sanitize}(\mathfrak{ct})$ which take as input a TFHE ciphertext (in flattened GLWE or LWE format) and applies a sanitization algorithm to map $\mathfrak{ct} = (\underline{a}, b)$ into a ciphertext $\mathfrak{ct}' = (\underline{a}', b')$ in flattened GLWE format. The output ciphertext $\mathfrak{ct}'$ encrypts the same

message m as $\mathfrak{ct}$. We can then produce a sanitized pre-decryption value $p' = b' - \underline{a}' \cdot \underline{s}$. The combination of the two operations we will denote by $p' \leftarrow \mathsf{PD\text{-}Sanitize}(\mathfrak{ct})$.

Associated to the algorithm $\mathsf{PD\text{-}Sanitize}(\mathfrak{ct})$ (and hence $\mathsf{Sanitize}(\mathfrak{ct})$) is a simulator algorithm $\mathsf{PD\text{-}Simulate}(m)$ which inputs a message m and outputs a simulated pre-decryption value p_s. The key aspect (for use in threshold decryption for TFHE) is that the output of algorithm $\mathsf{PD\text{-}Sanitize}(\mathfrak{ct})$ is indistinguishable from that of $\mathsf{PD\text{-}Simulate}(m)$. Thus this simulation can be directly plugged into the simulation security proof of [DDK+23] without needing to increase the ciphertext modulus q.

Unpealing a further layer of detail, our sanitization method $\mathsf{Sanitize}(\mathfrak{ct})$ is inspired by the standard sanitization method of [BI22], in that it "simply" involves executing a randomized version of the TFHE bootstrapping algorithm. In particular, the standard (deterministic) GSW decomposition operation $G^{-1}(\cdot)$ is replaced by a randomized version $G_r^{-1}(\cdot)$. This is then followed by the addition of a random GLWE encryption of zero. This results in us being able to show that the noise term e contains no information about the secret key $\underline{s}$ (see Figure 4 and Theorem 1 later). Thus we also achieve sanitization in the sense of [DS16], just as is done in [BI22][2].

1.1.1 Application to Server-Assisted Threshold Decryption:

In this work we mitigate two of the problems with the "double flood and round"/"flood, round and bath" method of [PS24], in that we first extend it from BGV/BFV/CKKS style FHE schemes to schemes such as TFHE and we also extend it from full-threshold to arbitrary threshold access structures.

The extension to TFHE is done by providing a different methodology on the server end. Instead of flooding and applying Gaussian rounding, the server will apply a method of ciphertext sanitization which is particularly tailored to the TFHE scheme; namely the sanitzation method described above. The final threshold decryption is performed in the same way as in [PS24], by appealing to the yaLWE problem introduced in [MS25, PS24], via a small amount of additional noise being added (i.e. what we have dubbed above a "bath"). Thus our method becomes one of "sanitize and bath".

The major benefit of this new method, in comparison to the TFHE threshold decryption method from [DDK+23], is that we do not need to increase the ciphertext modulus from $q = 2^{64}$ to 2^{128} due to the sanitization algorithm replacing the SwitchSquash algorithm. A major disadvantage is that the protocol requires the assistance of an honest server to apply the sanitization step.

We also show how the "bath" part of the algorithm can be applied not only for full threshold access structures (which we only achieve passive security) as is done in [PS24], but also for threshold access structures. This is done by applying the techniques given in [DDK+23]. This threshold "bath" techniques can also be applied to the BGV/BFV/CKKS situation as presented in [PS24], and when $\mathfrak{t} < \mathfrak{n}/3$ it enables a robust threshold decryption protocol. However, this extension to arbitrary thresholds requires a slight modification to the yaLWE problem. This modification is a new assumption, although closely related to the original yaLWE problem it is not (unlike the yaLWE problem) implied by the LWE assumption.

We then go on to show how our methodology can be applied to the small parameters one finds in TFHE, without the need to increase the ciphertext modulus.

1.1.2 Application to funcCPA:

The notion of funcCPA security was introduced in [AGHV22] to capture the security of *client-aided outsourcing protocols*. These are protocols, where a client outsources some computation but assists the server during the evaluation. Typically, this is used to circumvent the costly bootstrapping operation of FHE schemes, where the client decrypts

[2]There is a small bug in the definition of sanitization from [DS16], which we discuss in Appendix A.

the ciphertext to bootstrap, possibly applies a function to the plaintext, encrypts the result and returns this ciphertext to the server, which may now continue the computation. To model this kind of protocol in a security game, the funcCPA notion considers a game that extends the classical IND-CPA security game with a functional re-encryption oracle. The adversary may query this oracle with arbitrary ciphertext and function and the challenger provides the corresponding re-encryption. The work of [AGHV22] shows that CPA security is not sufficient to argue funcCPA security, which means that such protocols instantiated with IND-CPA secure schemes may be insecure. Constructing funcCPA secure schemes generically from IND-CPA secure schemes is challenging, see for example [DHW23].

In [AGHV22] a generic construction of a funcCPA secure scheme from a *homomorphic* scheme was given relying on ciphertext sanitization. Given a sanitization algorithm, we can turn an IND-CPA secure FHE scheme into a funcCPA secure scheme: simply run the sanitization after encryption and after evaluation.[3] The idea is that now $\mathsf{Enc}(f(\mathsf{Dec}(\mathfrak{ct})))$ is indistinguishable from $\mathsf{Eval}(f, \mathfrak{ct})$ for any ciphertext $\mathfrak{ct}$ and thus the re-encryption oracle can be easily simulated using Eval.

The main issue in the context of client-aided outsourcing protocols is efficiency if sanitization is based on bootstrapping (possibly multiple times as in [DS16]). Following the construction from [AGHV22], the client needs to run the sanitization once for every re-encryption query. But this defeats the purpose of client-aided outsourcing protocols, where the idea usually is to improve efficiency by not running the bootstrapping at all. Performing bootstrapping-based sanitization during encryption essentially just puts the burden of computation back on the client.

Efficiently simulatable sanitization offers a solution to this problem. Note that during encryption, the underlying message is known, so the encryption algorithm can simply run the simulation and does not need to perform the entire sanitization. In the context of our work, where we show that sanitized TFHE ciphertexts are statistically close to fresh LWE encryptions, the encryption algorithm does not need to be changed at all (beyond adjusting the noise parameter).

Interestingly, in the context of client-aided outsourcing protocols, we need not worry about the concrete efficiency of the sanitization algorithm, which is in contrast to other applications of sanitization, including the one of threshold FHE described above. This is because it is only used for the proof. In fact, we do not even need to publish the bootstrapping key. The mere existence of the evaluation algorithm suffices to guarantee funcCPA security. This means we can use parameters that minimize the noise in the output ciphertext at the "expense" of a larger running time, for example a minimal decomposition base of two, high precision FFT computations, etc.

2 Preliminaries

2.1 Basic Mathematical Recap

2.1.1 Notation:

Let $R = \mathbb{Z}[X]/(X^N + 1)$ for $N = 2^k$ for some $k \in \mathbb{N}$ and $R_q = R/qR$. Thoughout this work, we assume $q = 2^{k'}$ for some $k' \in \mathbb{N}$. Elements in $\mathbb{Z}$, $\mathbb{Z}_q$ or $\mathbb{R}$ will be denoted by a, vectors by $\underline{a}$ and matrices by A. Elements in R or R_q will be denoted by $\mathbf{a}$, vectors and matrices over R and R_q by $\underline{\mathbf{a}}$ and $\mathbf{A}$, respectively. The inner product of two vectors $\underline{a}, \underline{b}$ (or $\underline{\mathbf{a}}, \underline{\mathbf{b}}$) will be denoted by $\underline{a} \cdot \underline{b}$ (or $\underline{\mathbf{a}} \cdot \underline{\mathbf{b}}$, resp.). As vector spaces R and R_q are isomorphic to $\mathbb{Z}^N$ and $\mathbb{Z}_q^N$, respectively, via coefficient embedding and we will use them interchangeably in such contexts. Accordingly, norms of elements in R are defined using this isomorphism and the norm of a vector $\underline{\mathbf{v}} \in R^k$ as $\|\underline{\mathbf{v}}\|_2 = \sqrt{\sum_i \|\mathbf{v}_i\|_2}$. Multiplication

[3] Technically, one also needs to run sanitization on the input ciphertexts before evaluation.

in R can be viewed as a matrix-vector product over $\mathbb{Z}$, where one of the factors is expanded to its *multiplication matrix* and the other factor is viewed as vector over $\mathbb{Z}$ via coefficient embedding, see e.g. [BI22] for a more detailed description. This easily extends to inner products of vectors over polynomials: simply construct a wide block matrix with each block being the multiplication matrix of the corresponding element of one of the vectors.

2.1.2 Distributions:

For two distributions $\mathcal{X}, \mathcal{Y}$ it is common to use the statistical distance, defined as $\mathrm{SD}(\mathcal{X}, \mathcal{Y}) = \sup_A |\Pr[\mathcal{X} \in A] - \Pr[\mathcal{Y} \in A]|$, to measure distinguishability. If $\epsilon = \mathrm{SD}(\mathcal{X}, \mathcal{Y})$, then a distinguisher requires at least about $1/\epsilon$ samples to distinguish the two distribution with high advantage. In this work, it will be convenient to rely on the max-log distance [MW17], since it more conveniently captures the relative error and allows to obtain tighter parameters. For two distributions $\mathcal{X}, \mathcal{Y}$ with the same support, the max-log distance is defined as $\mathrm{ML}(\mathcal{X}, \mathcal{Y}) = \sup_a |\ln \Pr[\mathcal{X} = a] - \ln \Pr[\mathcal{Y} = a]|$. The max-log distance is closely related to KL divergence (of order infinity) and the relative error, but, in contrast to these measures, it is a metric (i.e. symmetric and satisfies triangle inequality). Like the statistical distance, it satisfies the data processing inequality, i.e. $\mathrm{ML}(f(\mathcal{X}, \mathcal{Y})) \leq \mathrm{ML}(\mathcal{X}, \mathcal{Y})$ for any f, $\mathcal{X}$ and $\mathcal{Y}$. If $\epsilon = \mathrm{ML}(\mathcal{X}, \mathcal{Y}) \leq 1/3$, then a distinguisher requires at least about $1/\epsilon^2$ many samples to distinguish the two distributions with high advantage. This is due to the fact that $\mathrm{SD}(\mathcal{X}^n, \mathcal{Y}^n) \leq \sqrt{n} \cdot \mathrm{ML}(\mathcal{X}, \mathcal{Y})$ (as long as $\mathrm{ML}(\mathcal{X}, \mathcal{Y}) \leq 1/3$) [MW17]. In this work, we use the notation $\mathcal{X} \overset{\epsilon}{\approx} \mathcal{Y}$ to mean that $\mathrm{ML}(\mathcal{X}, \mathcal{Y}) \leq \epsilon$. In contrast, we will use $\mathcal{X} \approx \mathcal{Y}$ to refer to statistical indistinguishability in a more general and informal sense, by which we mean that there exists a "small enough" ϵ such that $\mathrm{SD}(\mathcal{X}, \mathcal{Y}) \leq \epsilon$ or $\mathrm{ML}(\mathcal{X}, \mathcal{Y}) \leq \epsilon$.

2.1.3 Linear Algebra:

The singular values $\sigma_1(A) \geq \sigma_2(A) \geq \cdots \geq \sigma_d(A)$ of a matrix $A \in \mathbb{R}^{m \times n}$ are the square roots of the first $d = \min(m, n)$ eigenvalues of its Gram matrix $A^t \cdot A$ in non-increasing order. The spectral norm $\|A\|_2$ of a matrix is its largest singular value $\sigma_1(A)$. A matrix Σ is positive semidefinite if and only if it can be written as $\Sigma = S \cdot S^t$ for some matrix S; we write $S = \sqrt{\Sigma}$, and say that S is a *square root* of Σ. Note that such a square root is not unique, because, e.g., $-S = \sqrt{\Sigma}$ as well. We often just write $\sqrt{\Sigma}$ to refer to some arbitrary but *fixed* square root of Σ. For any matrix A with full column rank we define A^+ as the pseudo-inverse of A, i.e. $A^+ = (A^t \cdot A)^{-1} \cdot A^t$ (which simplifies to A^{-1} if A is invertible).

2.1.4 Norm of Matrix Representation:

We will use the following lemma which bounds the matrix norms of a vector of the matrix representations of a vector of elements in R in terms of norms of the elements in R.

Lemma 1. *Let $\underline{\mathbf{e}} \in R^k$ and let $E \in \mathbb{Z}^{N \times k \cdot N}$ be the multiplication matrix of $\underline{\mathbf{e}}$. Then*

$$\|E\|_2 \leq \sqrt{N} \cdot \|\underline{\mathbf{e}}\|_2 \quad .$$

Proof. Recall that $\|A\|_2 = \max_{\|\underline{x}\|_2 = 1} \|A \cdot \underline{x}\|_2$. Observe that

$$\|A \cdot \underline{x}\|_2^2 = \sum \underline{a}_i \cdot \underline{x}^2 \leq \sum \|\underline{a}_i\|_2^2 \cdot \|\underline{x}\|_2^2$$

by Cauchy-Schwarz. Since we have $\|\underline{x}\|_2 = 1$ and, in the special case of $A = E$, $\|\underline{a}_i\|_2^2 = \|\underline{\mathbf{e}}\|_2$, the result follows. □

2.2 Lattices and Gaussians

An n-dimensional lattice Λ is a discrete subgroup of $\mathbb{R}^n$. Every lattice is generated by a (non-unique) basis B, i.e. $\Lambda = B\mathbb{Z}^k$, where B has full column rank. A coset of a lattice Λ is a set of the form $A = \Lambda + \underline{c} = \{\underline{v} + \underline{a} \mid \underline{v} \in \Lambda\}$ for some $\underline{c} \in \mathbb{R}^n$. The dual $\Lambda^\vee$ of a lattice Λ is the lattice $\Lambda^\vee = \{\underline{x} \in \mathsf{span}(\Lambda) \mid x \cdot \Lambda \subset \mathbb{Z}\}$. A lattice subspace S of Λ is the linear span of some set of lattice points, i.e. $S = \mathsf{span}(\Lambda \cap S)$. For a lattice Λ of rank k and $i < k$ we denote the i-th minimum $\lambda_i(\Lambda)$ as the smallest r such that a zero centered ball with radius r contains i linearly independent lattice vectors.

Let $\mathcal{D}$ be the Gaussian probability measure on $\mathbb{R}^k$ (for any $k \geq 1$) having density function defined by $\rho(\underline{x}) = e^{-\pi \cdot \|\underline{x}\|^2}$, the Gaussian function with total measure $\int_{\underline{x} \in \mathbb{R}^k} \rho(\underline{x})\, d\underline{x} = 1$. For any (possibly non-full-rank) matrix $S \in \mathbb{R}^{n \times k}$, we define the (possibly non-spherical) Gaussian distribution[4]

$$\mathcal{D}_S := S \cdot \mathcal{D}$$

as the image of $\mathcal{D}$ under S; this distribution has covariance $\Sigma/(2\pi)$ where $\Sigma = SS^t$ is positive semidefinite. Notice that $\mathcal{D}_S$ depends only on Σ, and not on any specific choice of the square root S. So, we often write $\mathcal{D}_{\sqrt{\Sigma}}$ instead of $\mathcal{D}_S$. When $\Sigma = s^2 I$ is a scalar matrix, we often write $\mathcal{D}_s$ (observe that $\mathcal{D} = \mathcal{D}_1$).

For any lattice coset $A = \Lambda + \underline{c}$ and matrix S, we define the distribution $\mathcal{D}_{\Lambda + \underline{c}, S}$ as the (origin-centered) discrete Gaussian distribution given by $\Pr[\underline{x} \leftarrow \mathcal{D}_A] := \rho_S(\underline{x}) / \sum_{\underline{y} \in A} \rho_S(\underline{y})$.

We now recall the notion of the *smoothing parameter* [MR04] and its generalization to non-spherical Gaussians [Pei10].

Definition 1. For a lattice Λ and $\epsilon \geq 0$, we say $\eta_\epsilon(\Lambda) \leq 1$ if $\rho(\Lambda^\vee) \leq 1 + \epsilon$. For any matrix S of full column rank, we write $\eta_\epsilon(\Lambda) \leq S$ if $\Lambda \subset \mathsf{span}(S)$ and $\eta_\epsilon(S^+\Lambda) \leq 1$. When $S = sI$ is a scalar matrix, we may simply write $\eta_\epsilon(\Lambda) \leq s$.

Lemma 2 ([MR04])**.** *Let Λ be any rank m lattice and ϵ be any positive real. Then*

$$\eta_\epsilon(\Lambda) \leq \lambda_m(\Lambda) \cdot \sqrt{\frac{\ln\left(2 \cdot m \cdot (1 + 1/\epsilon)\right)}{\pi}} \quad .$$

2.2.1 Convolution over R:

We will base our analysis on the following basic result from [GMPW20].

Lemma 3. *For any $\epsilon \in [0, 1)$ defining $\bar{\epsilon} = 2 \cdot \epsilon/(1 - \epsilon)$, matrix S of full column rank, lattice coset $A = \Lambda + \underline{a} \subset \mathsf{span}(S)$, and matrix T such that $\ker(T)$ is a Λ-subspace and $\eta_\epsilon(\Lambda \cap \ker(T)) \leq S$, we have*

$$T \cdot \mathcal{D}_{A,S} \overset{\bar{\epsilon}}{\approx} \mathcal{D}_{T \cdot A, T \cdot S} \quad .$$

We use the above lemma to provide a simpler proof of the following lemma, which is for Lemma 9 from [BI22].

Lemma 4. *Let $\epsilon > 0$ and $\bar{\epsilon} = 2\epsilon/(1 - \epsilon)$, $\Lambda \in R^\nu$ be a rank-νN lattice, $S_1 \in \mathbb{R}^{k \times \nu N}$ and $S_2 \in \mathbb{R}^{k' \times N}$ be matrices with full column rank. For any $\underline{\mathbf{e}} \in R_q^\nu$ and any $\underline{\mathbf{c}} \in R_q^\nu$, let E be the multiplication matrix corresponding to $\underline{\mathbf{e}}$. Then if*

$$\sigma_{(\nu+1) \cdot N} \begin{pmatrix} S_1 & 0 \\ 0 & S_2 \end{pmatrix} \geq \left(1 + \sqrt{N} \cdot \|\underline{\mathbf{e}}\|_2\right) \cdot \lambda_{\nu N}(\Lambda) \cdot \sqrt{\frac{\ln(2 \cdot \nu \cdot N \cdot (1 + 1/\epsilon))}{\pi}}$$

[4] There are two slightly different definitions of Gaussian distribution, one with a factor $-\pi$ in the exponent and one with the factor $-1/2$. In the latter, the noise parameter is typically denoted by σ and is equal to the standard deviation of the distribution, which is a little more intuitive. However, we choose to use the former definition for consistency with [GMPW20], which we base our analysis on. It is straight-forward to translate between the parameters of these two definitions using a factor $\sqrt{2 \cdot \pi}$.

we have

$$\underline{e}^t \mathcal{D}_{\Lambda+\underline{\mathbf{c}},S_1} + \mathcal{D}_{R,S_2} \overset{\bar{\epsilon}}{\approx} \mathcal{D}_{R,\Gamma}$$

where

$$\Gamma = \sqrt{E \cdot S_1^t \cdot S_1 \cdot E^t + S_2^t \cdot S_2} \ .$$

Proof. Apply Lemma 3 with $A = \Lambda \otimes R + (\underline{\mathbf{c}}, \mathbf{0})$ (viewed as a coset of a lattice in $\mathbb{Z}^{(\nu+1)\cdot N}$), $T = [E \mid I]$, $S = \begin{pmatrix} S_1 & 0 \\ 0 & S_2 \end{pmatrix}$. It is easy to verify that $T \cdot (\Lambda \otimes R) = R$ and $T \cdot (\underline{\mathbf{c}}, \mathbf{0}) \in R$, i.e. $T \cdot A = R$. Note also that $T \cdot S$ is a square root of Γ^2, i.e. $(T \cdot S)^t \cdot (T \cdot S) = E \cdot S_1^t \cdot S_1 \cdot E^t + S_2^t \cdot S_2$. We now verify the conditions of Lemma 3.

Clearly, $\Lambda \otimes \mathbb{Z}^N \subset \mathbb{Z}^{(\nu+1)\cdot N} \subset \mathsf{span}(S)$ since S_1 and S_2 have full column rank. Furthermore, we have $\ker(T) = \{(\underline{v}, \underline{w}) \in \mathbb{R}^{\nu \cdot N} \times \mathbb{R}^N \mid E \cdot \underline{v} = -\underline{w}\}$, which has rank $\nu \cdot N$. Now take any set of linearly independent lattice vectors in Λ of norm bounded by $\lambda_{\nu \cdot N}$ and arrange them in the matrix B. Extend them to $B' = \begin{pmatrix} B \\ -E \cdot B \end{pmatrix}$ which is a set of $\nu \cdot N$ linearly independent vectors that are in $\Lambda \otimes \mathbb{Z}^N$ and that generate $\ker(T)$. Hence, $\ker(T)$ is a $(\Lambda \otimes \mathbb{Z}^N)$-subspace. It remains to show that $\eta_\epsilon(\Lambda') \leq S$, where $\Lambda' = (\Lambda \otimes \mathbb{Z}^N) \cap \ker(T)$. By definition, this is equivalent to $\eta_\epsilon(S^+ \cdot \Lambda') \leq 1$. Note that the vectors $S^+ \cdot \begin{pmatrix} I \\ -E \end{pmatrix} B \in \Lambda'$ are all bounded by $c \cdot \lambda_{\nu \cdot N}(\Lambda)$, where $c = \left\| S^+ \cdot \begin{pmatrix} I \\ -E \end{pmatrix} \right\|_2$. So, $\lambda_{\nu \cdot N}(\Lambda') \leq c \cdot \lambda_{\nu \cdot N}(\Lambda)$ and by Lemma 2

$$\eta_\epsilon(S^+\Lambda') \leq c \cdot \lambda_{\nu \cdot N}(\Lambda) \cdot \sqrt{\frac{\ln\left(2 \cdot \nu \cdot \cdot N \cdot (1 + 1/\epsilon)\right)}{\pi}} \ .$$

We conclude by observing that $c \leq \|S^+\|_2 \cdot \left\| \begin{pmatrix} I \\ -E \end{pmatrix} \right\|_2$ and $\|S^+\|_2 \leq 1/\sigma_{(\nu+1)\cdot N}(S)$ and $\left\| \begin{pmatrix} I \\ -E \end{pmatrix} \right\|_2 = 1 + \|E\|_2 \leq 1 + \sqrt{N} \cdot \|\underline{e}\|_2$ by Lemma 1. □

Note that by the data processing inequality and triangle inequality, we may replace the input distributions $\mathcal{D}_{\Lambda+\underline{\mathbf{c}},S_1}$ and $\mathcal{D}_{R,S_2}$ in Lemma 4 by distributions, that approximate them. The approximation error in max-log distance will at most add to the resulting approximation error.

Lemma 4 requires that the minimal singular values of S_1 and S_2 are suitably bounded from below in order to be applicable. In the the proof of Theorem 1 and Theorem 2 in Section 4 we will apply Lemma 4 to matrices with a certain structure. In the following lemma, we show that their minimal singular value can be easily bounded from below.

Lemma 5. *Let $r > 0$ and $E_j \in \mathbb{R}^{N \times m}$ be arbitrary matrices for $j \in \{1, \ldots, n\}$ and $E_{\ell+1} = I_N$. Then, for any $i \in \{0, 1, \ldots \ell\}$ there exists a matrix $\Gamma = r \cdot \sqrt{\sum_{j=i}^{\ell+1} E_j \cdot E_j^t}$ that has full column rank and*

$$\sigma_N(\Gamma) \geq r \ .$$

Proof. Clearly, $r^2 \sum_i^\ell E_i \cdot E_i^t$ is positive semi-definite and thus all its eigenvalues are greater than or equal to zero. Then, from the definition of eigenvalues, it follows that the matrix $r^2 \cdot (I_N + \sum_i^\ell E_i \cdot E_i^t)$ has eigenvalues at least greater than or equal to $r^2 > 0$. This means that the matrix is positive definite and has a root with full column rank given by the Cholesky decomposition. Furthermore, this also shows that $\sigma_N(\Gamma) \geq r$. □

2.3 Hardness Assumptions

We first introduce the standard (G)LWE hardness assumption.

Definition 2. Let $k, q \in \mathbb{N}$ and $\mathcal{X}$ be a "small" distribution over the degree N extension R of $\mathbb{Z}$. Then, for a fixed $\underline{\mathbf{s}} \in R_q^k$ the *GLWE distribution* $\mathsf{GLWE}^{\underline{\mathbf{s}}}_{N,q,k,\mathcal{X}}$ is defined as $(\underline{\mathbf{a}}, \mathbf{b} = \underline{\mathbf{a}} \cdot \underline{\mathbf{s}} + \mathbf{e})$ where $\underline{\mathbf{a}}$ is chosen uniformly at random from R_q^k and $\mathbf{e}$ is chosen from $\mathcal{X}$.

Let $\mathcal{S}$ be some distribution over R_q^k. The *GLWE problem* $\mathsf{GLWE}^{\mathcal{S}}_{N,q,k,\mathcal{X}}$ is to distinguish the distribution $\mathsf{GLWE}^{\underline{\mathbf{s}}}_{N,q,k,\mathcal{X}}$ from the uniform distribution over R_q^{k+1}, where $\underline{\mathbf{s}} \leftarrow \mathcal{S}$.

The *GLWE assumption* is that the *GLWE problem* is considered to be a hard computational problem. Note that, the traditional $\mathsf{LWE}^{\mathcal{S}}_{q,k,\mathcal{X}}$ problem is a special case of the $\mathsf{GLWE}^{\mathcal{S}}_{N,q,k,\mathcal{X}}$ where $N = 1$. With suitable choice for the secret distribution $\mathcal{S}$, error distribution $\mathcal{X}$ (e.g. discrete or rounded Gaussian or even bounded uniform distributions with sufficiently large variance) and ring dimension k the corresponding $\mathsf{GLWE}^{\mathcal{S}}_{N,q,k,\mathcal{X}}$ problem is considered to be hard.

In the context of threshold LWE and threshold FHE, [MS25, PS24] introduce variants of "yet another LWE" problem (yaLWE). We will state the version from [PS24] first and then discuss variants of it.

Definition 3 (The yaLWE Problem [PS24])**.** Let $\ell, q \in \mathbb{N}$, $\sigma, \eta \geq 0$ and $\mathcal{S}$ denote a distribution over $\mathbb{Z}_q^\ell$. The $\mathsf{yaLWE}^{\mathcal{S}}_{q,\ell,\sigma,\eta}$ problem is to distinguish the two distributions

$$(\underline{a},\ b = \underline{a} \cdot \underline{s} + e, b' = \underline{a} \cdot \underline{s} + d, \|\underline{s}\|) \quad \text{and} \quad (\underline{a}, u, u + h, \|\underline{s}\|),$$

where $\underline{a} \leftarrow \mathbb{Z}_q^\ell$, $\underline{s} \leftarrow \mathcal{S}$, $e \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}$, $d \leftarrow \mathcal{D}_{\mathbb{Z},\eta}$, $u \leftarrow \mathbb{Z}_q$, and $h \leftarrow \mathcal{D}_{\mathbb{Z},\sqrt{\sigma^2+\eta^2}}$.

The yaLWE assumption is that the yaLWE problem is indeed hard. If the noise and secret distributions are discrete Gaussians, it was argued in [PS24] (see also [MS25]) that the yaLWE assumption is implied by the LWE assumption (with approriate parameters). We assume that the problem is also hard when choosing the uniform binary distribution as secret distribution $\mathcal{S}$. On the other hand, [MS25] also showed that yaLWE does not hold if *both* the variables e and d are drawn from uniform distributions (over polynomially sized sets). To see this, note that it is trivial to obtain the value $e - d$ from a yaLWE sample, which leaks the values e and d with noticeable probability. So if considering other noise distributions, the yaLWE can only plausibly hold if e and d retain some entropy even conditioned on any possible value for $e - d$.

We now introduce a variant of the yaLWE assumption, the $(\mathsf{ya})^2\mathsf{LWE}$ assumption (for "yet another yaLWE assumption") that we believe is plausibly secure, for when e is drawn from a discrete Gaussian and d is drawn from a sum of uniform distributions.

Definition 4 (The $(\mathsf{ya})^2\mathsf{LWE}$ Problem)**.** Let $\ell, q \in \mathbb{N}$, $\sigma, w, B \geq 0$ and $\mathcal{S}$ denote a distribution over $\mathbb{Z}_q^\ell$. We define $\mathcal{U}_{w,B} = \sum_{i=1}^{w} \mathcal{U}([-B, \ldots, B])$. The $(\mathsf{ya})^2\mathsf{LWE}^{\mathcal{S}}_{q,\ell,\sigma,w,B}$ problem is to distinguish the two distributions

$$(\underline{a},\ b = \underline{a} \cdot \underline{s} + e, b' = \underline{a} \cdot \underline{s} + d, \|\underline{s}\|) \quad \text{and} \quad (\underline{a}, u, u + h, \|\underline{s}\|),$$

where $\underline{a} \leftarrow \mathbb{Z}_q^\ell$, $\underline{s} \leftarrow \mathcal{S}$, $e \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}$, $d \leftarrow \mathcal{U}_{w,B}$, $u \leftarrow \mathbb{Z}_q$, and $h \leftarrow \mathcal{D}_{\mathbb{Z},\sigma} + \mathcal{U}_{w,B}$.

We conjecture that the $(\mathsf{ya})^2\mathsf{LWE}$ assumption is secure as long as w and B are chosen such that $\mathsf{LWE}^{\mathcal{S}}_{q,\ell,\mathcal{U}_{w,B}}$ is hard. The latter can be reasonably argued to be as hard as $\mathsf{LWE}^{\mathcal{S}}_{q,\ell,\mathcal{D}_{\mathbb{Z},r}}$ with $r = \sqrt{2\pi \cdot w \cdot B \cdot (B+1)/3}$, i.e. LWE with a discrete Gaussian with matching standard deviation.

Note that even for $w = 1$ (and suitably large B) the attack from [MS25] outlined above fails, so it is reasonable to assume that such an instantiation is secure. Furthermore, by

the CLT, $\mathcal{U}_{w,B}$ approaches the (discrete) Gaussian as w grows. So for "large enough" w, the $(\mathsf{ya})^2\mathsf{LWE}$ problem matches the yaLWE problem from [PS24].

In summary, even though we will instantiate yaLWE and $(\mathsf{ya})^2\mathsf{LWE}$ with $\mathcal{S}$ being the binary uniform distribution, we believe there is sufficient evidence that both problems are computationally as hard as LWE (with appropriate parameters).

2.4 TFHE Recap

We recap here on the TFHE scheme, in as much detail as neeeded to understand this paper. In the following we try to give a succinct intuitive description of TFHE that we hope is detailed enough to follow the rest of the work without cluttering it with too much formal notation.

For a more full and rigorous description, we refer to [CGGI20] and follow up work, or the survey [Joy22]. The TFHE scheme is an FHE scheme based on the (G)LWE problem. The basic ciphertexts in TFHE are simple LWE ciphertexts, but the bootstrapping and sanitization methods use a range of other ciphertexts based on GLWE; namely (G)LWE ciphertexts, GLev ciphertexts and GGSW ciphertexts.

(G)LWE ciphertexts: Let $p, q, k \in \mathbb{N}$, $\mathcal{D}_{R,r}$ be GLWE (or simply LWE in case $N = 1$) parameters with $p < q$. For a message $\mathbf{m} \in R_p$, we define its (G)LWE encryption to be $(\underline{\mathbf{a}}, \mathbf{b} = \underline{\mathbf{a}} \cdot \underline{\mathbf{s}} + \mathbf{e} + \Delta \cdot \mathbf{m})$, where $\underline{\mathbf{a}} \in R_q^k$ is uniformly random, $\underline{\mathbf{s}} \in R_q^k$ is chosen from the uniform binary distribution and $\mathbf{e}$ from $\mathcal{D}_{R,r}$, and $\Delta = \lfloor q/p \rceil$. By the hardness of (G)LWE this is a semantically secure ciphertext. It can be decrypted using $\underline{\mathbf{s}}$ if $\mathbf{m}$ represents a suitable encoding of a message that is robust with respect to the error distribution. In the context of LWE ciphertexts we typically denote the dimension by ℓ instead of k. Note that (G)LWE ciphertexts are additively homomorphic and may be multiplied with "small" elements in R_q, where smallness is determined such that the resulting ciphertext can still be correctly decrypted given the error distribution and the encoding. It is trivial to turn this description into a symmetric encryption scheme, but there are also efficient ways to construct a public key scheme [Reg05, Joy24]. We note that in either case the structure of the ciphertexts are the same as in the symmetric scheme.

GLev Ciphertexts: GLev ciphertexts (where the "Lev" stands for *levelled*) are a way to extend (G)LW ciphertexts in order to allow for multiplication with arbitrary constants. It is based on the standard approach of decomposition: let $\underline{g} = (1, 2^\beta, \ldots, 2^{\nu \cdot \beta})^t$ be the "standard" gadget vector[5], where $\nu \cdot \beta = \log q$. Decomposition, takes as input a value $a \in \mathbb{Z}_q$ and computes a vector $\underline{v} \in \mathbb{Z}^\nu$ such that $\underline{v} \cdot \underline{g} = a$. This is typically done via a radix decomposition and is easily generalized to vectors by defining $G = I \otimes \underline{g}^t$ and performing decomposition componentwise. Similarly, this can be extended to vectors over R by performing the decomposition coefficientwise and collecting the results in polynomials again, see e.g. [BI22]. With this decomposition at hand, we define the GLev encryption of $\mathbf{m} \in R_q$ with parameters β and ν to be the set of element-wise (G)LWE encryptions of $\mathbf{m} \cdot \underline{g}$. Note that, such a ciphertext can be multiplied with an arbitrary element $\mathbf{a} \in R_q$ by first computing $G^{-1}(\mathbf{a})$ and taking the inner product with the GLev ciphertext. Since all components of $G^{-1}(\mathbf{a})$ are small the result is an (G)LWE encryption of $\mathbf{a} \cdot \mathbf{m}$ by the homomorphic properties of the (G)LWE ciphertexts (and assuming suitable parameters). It is this operation $G^{-1}(\mathbf{a})$ which when generalized to a randomized variant $G_r^{-1}(\mathbf{a})$ that will form the basis of our sanitization methodology.

[5] For simplicity, we focus in this work on power of 2 modulus q and decomposition base 2^β. Furthermore, the sanitization in this work does not extend to approximate decomposition, so we only consider exact decomposition.

GGSW Ciphertexts: While GLev ciphertexts allow one to multiply encrypted values with arbitrary constants, we would also like to be able to efficiently multiply encrypted values with each other. This can be achieved using GGSW ciphertexts (named after [GSW13]). The idea is to encrypt $\mathbf{m}$ as a GLev ciphertext and for each element $\mathbf{s}_i$ of the secret key $\underline{\mathbf{s}} \in R_q^k$, additionally encrypt $\mathbf{m} \cdot \mathbf{s}_i$ as a GLev ciphertext. This set of $k+1$ GLev ciphertexts forms the GGSW ciphertext. By the properties of GLev ciphertexts, this allows one to perform the multiplication while homomorphically decrypting a ciphertext $(\underline{\mathbf{a}}, \mathbf{b} = \underline{\mathbf{a}} \cdot \underline{\mathbf{s}} + \Delta \cdot \mathbf{m}' + \mathbf{e})$ by homomorphically computing $\mathbf{b} \cdot \mathbf{m}$ and $\mathbf{a}_i \cdot \mathbf{s}_i \cdot \mathbf{m}$ and using the additive homomorphism of GLWE ciphertexts. Note that, $\mathbf{m}$ should not be too large as this would blow up the error. In TFHE, the message $\mathbf{m}$ is usually a key bit and thus binary, so clearly small. In summary, a GGSW ciphertext C allows homomorphic multiplication with a GLWE ciphertext $\mathfrak{ct}$, which results in a GLWE ciphertext $\mathfrak{ct}'$ encrypting the product of the two plaintexts (as long as the plaintext in the GGSW ciphertext is sufficiently small). This operation is typically called the *external product* and is denoted by $\mathfrak{ct}' \leftarrow \mathfrak{ct} \boxdot C$.

We are now in a position to define the main homomorphic operation of the TFHE scheme, namely the programmable bootstrapping (PBS) operation. The input to the PBS operation are the following sets of data.

- An LWE ciphertext $\mathfrak{ct} = (\underline{a}, b = \underline{a} \cdot \underline{s} + e + \Delta \cdot m) \in \mathbb{Z}_q^{k \cdot N + 1}$ to bootstrap, where the corresponding secret key is $\underline{\tilde{s}} \in \{0,1\}^{k \cdot N}$ (resulting from a flattening of the secret bootstrapping key $\underline{\tilde{\mathbf{s}}} \in R^k$),
- An element $\mathbf{t} \in R_q$ that encodes a function[6] $f : \mathbb{Z}_p \mapsto \mathbb{Z}_p$ into the bootstrap,
- A bootstrapping key bsk, which is a collection of GGSW ciphertexts encrypting the individual bits $s_i \in \{0,1\}$ of the secret key $\underline{s} \in \mathbb{Z}_q^\ell$ under the bootstrapping secret key $\underline{\tilde{\mathbf{s}}} \in R_q^k$ with binary coefficients, and
- A key switching key ksk, which is a collection of GLev ciphertexts encrypting the coefficients of the bootstrapping key under the secret key $\underline{s}$.

The PBS operation outputs a ciphertext $\mathfrak{ct}' = (\underline{a}', \underline{b}' = \underline{a}' \cdot \underline{\tilde{s}} + e' + \Delta \cdot f(m))$, where e' only depends on bsk and ksk, not on e. For suitable parameters, we have that $|e'| < |e|$. Combining this with the additive homomorphism of LWE ciphertexts we obtain a Fully Homomorphic Encryption scheme. The PBS operation itself consists of four steps: Key Switch, Mod Switch, Blind Rotation and Flattening, executed (for our purposes) in this order. We describe each of these in detail.

$\mathsf{KeySwitch}(\mathfrak{ct}, \mathsf{ksk})$**:** The key switch is a classic LWE type operation that follows from the observation that GLev ciphertexts can be used to homomorphically decrypt a GLWE ciphertext. Let $\mathfrak{ct} = (\underline{a}, b) \in \mathbb{Z}_q^{k \cdot N + 1}$ be a GLWE ciphertext with corresponding secret key $\underline{\tilde{s}} \in \mathbb{Z}_q^{k \cdot N}$. We would like to obtain a ciphertext $(\underline{a}', b') \in \mathbb{Z}_q^{\ell + 1}$ encrypting the same message as $\mathfrak{ct}$ but under the secret key $\underline{s} \in \mathbb{Z}_q^\ell$. We can do so by constructing a key switching key ksk that consists of GLev encryptions of $\tilde{s}_i$ under $\underline{s}$. Then, using the fact that we can multiply these ciphertexts with arbitrary constants using decomposition, we can homomorphically compute a ciphertext encrypting $b - \underline{a} \cdot \underline{\tilde{s}}$ under $\underline{s}$, which yields the desired ciphertext.

$\mathsf{ModSwitch}(\mathfrak{ct})$**:** We embed the input ciphertext $\mathfrak{ct} = (\underline{a}, b)$ into the group $\langle X \rangle \subset R_q$, which is of size $2 \cdot N$. By this we mean that we map $\mathfrak{ct}$ to $(X^{a_1}, X^{a_2}, \ldots, X^{a_\ell}, X^b)$ and, looking ahead, this will allow us to homomorphically compute $X^{b - \sum_i a_i \cdot s_i}$. So in order

[6] There is a requirement for the function to be negacyclic, but we omit details since it is irrelevant for our work.

to match up the moduli, we first perform a modulus switch. In particular, this outputs $\mathfrak{ct}' = (\underline{a}', b') \in \mathbb{Z}_{2\cdot N}^{\ell+1}$, where

$$a_i' = \left\lfloor \frac{a_i \cdot 2 \cdot N}{q} \right\rceil$$

and similar for b'.

$\mathsf{BlindRotate}(\mathfrak{ct}, \mathsf{bsk}, \mathbf{t})$**:** The blind rotation is the core of the PBS. We begin its description by introducing a homomorphic ciphertext multiplexer (CMUX) operation: given two GLWE ciphertext $\mathfrak{ct}_0, \mathfrak{ct}_1 \in (R_q^{k+1})^2$ and a GGSW encryption C_μ of a bit $\mu \in \{0,1\}$, all under the same key $\underline{\tilde{\mathbf{s}}} \in R_q^k$, we can compute the GLWE ciphertext

$$\mathfrak{ct} = (\mathfrak{ct}_1 - \mathfrak{ct}_0) \boxdot C_\mu + \mathfrak{ct}_0$$

where $\boxdot$ corresponds to the external product described above. By the additive homomorphism and the properties of the external product, $\mathfrak{ct}$ will encrypt the same plaintext as $\mathfrak{ct}_\mu$.

We are now ready to describe the blind rotation. Let $(\underline{a}, b) \in \mathbb{Z}_{2\cdot N}^{\ell+1}$ be the ciphertext after the mod switch. The blind rotation begins by constructing a trivial GLWE ciphertext $(\underline{\mathbf{0}}, X^{-b} \cdot \mathbf{t})$, where $\underline{\mathbf{0}} \in R_q^k$ is the all zero vector of size k. Then, it iterates over the elements a_i of $\underline{a}$, where the output GLWE ciphertext $\mathfrak{ct}$ from the previous iteration is multiplied element-wise by X^{a_i}. Note that, since X^{a_i} has low norm, this corresponds to multiplying the plaintext with X^{a_i}. The two ciphertexts $\mathfrak{ct}$ and $X^{a_i} \cdot \mathfrak{ct}$ are input to a homomorphic CMUX, with the control bit being the corresponding part of bsk, which is a GGSW ciphertext encrypting s_i. Accordingly, the result is a ciphertext encrypting the same plaintext as $X^{a_i s_i} \cdot \mathfrak{ct}$. After executing the full loop, the result is a GLWE ciphertext encrypting $X^{-b+\sum_i a_i \cdot s_i} \cdot \mathbf{t} = X^{-b+\underline{a}\cdot\underline{s}} \cdot \mathbf{t} = X^{-m-e} \cdot \mathbf{t}$. Note that, in R_q, this corresponds to a negacyclic rotation of $\mathbf{t}$ by $m + e$ positions. The operation is given in Figure 2, it is this operation which we will randomize in our sanitization procedure in the corresponding Figure 4.

$\mathsf{BlindRotate}(\mathfrak{ct}, \mathsf{bsk}, \mathbf{t})$

On input of an LWE ciphertext $\mathfrak{ct} = (\underline{a}, b) \in \mathbb{Z}_{2\cdot N}^{\ell+1}$ encrypting message m, a bootstrapping key $\mathsf{bsk} = (\mathsf{bsk}_i)_{i=1}^{\ell}$, and a test polynomial $\mathbf{t} \in R_q$ this outputs a GLWE ciphertext $\mathfrak{ct}'$ encrypting the plaintext $X^{-m-e} \cdot t$.

1. $\underline{\mathrm{z}} \leftarrow (\mathbf{0}, \ldots, \mathbf{0}, \mathbf{t} \cdot X^{-b}) \in R_q^{k+1}$.
2. For $i = 1$ to ℓ do:
   (a) $\underline{\mathrm{z}} \leftarrow \underline{\mathrm{z}} + \mathsf{bsk}_i \boxdot ((X^{a_i} - 1) \cdot \underline{\mathrm{z}})$.
3. Return $\underline{\mathrm{z}}$.

Figure 2: The Blind Rotation Algorithm.

By redundantly embedding the function f into the test polynomial $\mathbf{t}$, we can ensure that the error e is rounded away and the resulting ciphertext contains an encryption of $\Delta \cdot f(m)$ in its constant coefficient. In our application for sanitization we will use the identity function for f, and so as the redundant encoding, we use

$$\mathsf{enc}_{\mathsf{id}} = \sum_i \left(\sum_k \Delta \cdot i \cdot X^{i \cdot K + k} \right),$$

where the index k is over the error range (which has size at most K) and i over the plaintext space.

$\mathsf{Flatten}(\mathfrak{ct})$**:** The goal of flattening (in the literature also often referred to as sample extraction) is to convert a GLWE ciphertext into an LWE ciphertext encrypting the constant coefficient of the GLWE ciphertext, and where the key is a vector of bits corresponding to the concatenation of coefficient vectors in the GLWE secret key. We describe the special case of $k = 1$, since the generalization is straight-forward. So, given $(\mathbf{a}, \mathbf{b}) \in R_q^2$ we seek to construct $(\underline{a}', b') \in \mathbb{Z}_q^{N+1}$ such that $(\mathbf{b} - \mathbf{a} \cdot \tilde{\mathbf{s}})_0 = b' - \underline{a}' \cdot \underline{\tilde{s}}$. We note that

$$\mathbf{a} \cdot \tilde{\mathbf{s}} = \sum_i \mathbf{a} \cdot \tilde{s}_i \cdot X^i = \sum_i \left(X^i \cdot \mathbf{a}\right) \cdot \tilde{s}_i \ .$$

Since addition in R_q is elementwise, we may set $a'_i = (X^i \cdot \mathbf{a})_0$ and $b' = b_0$ in order to achieve our goal. Note that the error in the resulting ciphertext is the constant coefficient of the error in the GLWE ciphertext.

In the "traditional" booststrap operation in TFHE, as introduced in [CGGI16, CGGI20], the operations are executed in a different order, where the key switch is performed at the end of the PBS and the linear operations in between the bootstraps are computed on LWE ciphertexts of dimension ℓ. In [BBB+23, CJP21] the alternative order as outlined above is introduced, and intermediate operations are applied to LWE ciphertexts of dimension $k \cdot N$. This was done by [BBB+23, CJP21] to produce better parameters and a more efficient scheme. Looking ahead, we focus on the order of operations given in [BBB+23, CJP21], not for efficiency reasons, but because we focus on sanitizing the blind rotation. If this were followed by a keyswitch operation, it would induce a key dependency on the noise term again. So it is easier to argue about the output distribution of the entire bootstrap when considering the order of operations given in [BBB+23, CJP21].

In summary TFHE can be defined by four algorithms:

- $(\{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}\}, \underline{\tilde{s}}) \leftarrow \mathsf{KeyGen}(1^\kappa)$: A randomized key generation algorithm which produces a public key pk, a bootstrapping key bsk, a key switching key ksk and a secret key $\underline{\tilde{s}}$.

- $\mathfrak{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$: A randomized algorithm taking a message $m \in R_p$ and a public key pk and outputing a flattened GLWE ciphertext (i.e. an LWE ciphertext of dimension $k \cdot N$) which encrypts the message m in the form $\mathfrak{ct} = (\underline{a}, b = \underline{a} \cdot \underline{\tilde{s}} + e + \Delta \cdot m)$.

- $m \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct})$: Which takes a flattened GLWE ciphertext $\mathfrak{ct} = (\underline{a}, b)$ and a secret key, and outputs the plaintext m by rounding the value of $b - \underline{a} \cdot \underline{\tilde{s}}$ on division by Δ.

- $\mathfrak{ct}' \leftarrow \mathsf{PBS}_{\mathsf{ksk},\mathsf{bsk}}(\mathfrak{ct}, f)$ which takes a ciphertext $\mathfrak{ct}$ encrypting a value m, a function $f : R_p \longrightarrow R_p$, a keyswitching key ksk and a bootstrapping key bsk, and outputs a ciphertext $\mathfrak{ct}'$ encrypting $f(m)$.

By choosing the various parameters carefully one can obtain a scheme which is semantically secure (with any desired level of security) and which is correct (again to any desired level of failure probability).

3 Sanitization

In this section we present our main method for sanitization, we leave the formal proofs and analysis to the next section; leaving this section for the algorithmic description. The key idea, as explained before, and as used in [BI22], is a randomized decomposition method.

3.1 Randomized Decomposition

We can view the decomposition via the map $G^{-1}(\underline{a})$ as a lattice algorithm: let $\Lambda^{\perp}_{\underline{a}}(G) = \{\underline{v} \in \mathbb{Z}^{\ell \cdot \nu} \mid \underline{v} \cdot G = \underline{a} \bmod q\}$, then decomposition computes a short vector in the lattice coset $\Lambda^{\perp}_{\underline{a}}(G)$. The reason we need this vector to be short is that the resulting error in the context of an external product will be the inner product of $\underline{v}$ and the error in the GGSW ciphertext. The radix decomposition computes the shortest vector in $\Lambda^{\perp}_{\underline{a}}(G)$, but it is by no means the only possible solution. As shown in [MP12], we can sample from the discrete Gaussian over $\Lambda^{\perp}_{\underline{a}}(G)$, which will still yield a relatively short vector and gives better control of the resulting distribution. For completeness, we give the algorithm from [MP12] in Figure 3, which is a specialization of the sampling algorithm from [Kle00] for general lattices to the case of $\Lambda^{\perp}(\underline{g})$.

SampleG$_{r,\beta}(a)$

On input an element $a \in \mathbb{Z}_q$ this outputs an element distributed according to $\mathcal{D}_{\Lambda^{\perp}_{a}(\underline{g}^t),r}$, where $\underline{g} = (1, 2^{\beta}, 2^{2\beta}, \ldots, 2^{\nu\beta})$, $\nu \cdot \beta \geq \log q$ and $r \geq \eta_{\epsilon}(\Lambda^{\perp}(\underline{g}^t))$.

1. $x \leftarrow a$
2. For $i = 1$ to ν do:
   (a) $x_i \leftarrow \mathcal{D}_{2^{\beta}\mathbb{Z}+x,r}$
   (b) $x \leftarrow (x - x_i)/2^{\beta}$
3. Return $(x_1, x_2, \ldots, x_{\nu})$.

Figure 3: Sampling algorithm for $\mathcal{D}_{\Lambda^{\perp}_{a}(\underline{g}^t),r}$

Note that the only difference between radix decomposition with base 2^{β} and SampleG is in Step 2a: where radix decomposition would compute $x_i \leftarrow x \bmod 2^{\beta}$ (which can be viewed as choosing the smallest vector in the lattice coset $2^{\beta}\mathbb{Z} + x$), the sampling algorithm instead chooses a representative using a discrete Gaussian over $2^{\beta}\mathbb{Z}$. In the language of [CGGI20], SampleG is a decomposition algorithm of quality $\tau \cdot r$ and precision 0, where τ is a suitable tailcut parameter.

The algorithm reduces the problem of sampling from $\mathcal{D}_{\Lambda^{\perp}_{a}(\underline{g}^t),r}$ to sampling from $\mathcal{D}_{2^{\beta}\mathbb{Z}+x,r}$, i.e. sampling 2^{β} cosets of the integers. For this problem, there are a myriad of algorithms to choose from [BCG+14, DDLL13, Fol15, Kar16, MW17, PDG14, Wal19]. A key performance bottleneck of our method will be the generation of such samples in a suitably efficient manner.

We now describe the generalization of decomposition from $\mathbb{Z}_q$ to R, since it is not mentioned in [MP12] nor made explicit in [BI22]. Let $\underline{\mathbf{g}} = (1, 2^{\beta}, \ldots, 2^{\nu\beta})^t \in R^{\nu}$ as before, but considered as a vector of (constant) polynomials. We want to show that $\Lambda^{\perp}(\underline{\mathbf{g}}^t) = \{\underline{\mathbf{v}} \in R^{\nu} \mid \underline{\mathbf{v}} \cdot \underline{\mathbf{g}}^t = 0 \bmod q\}$ is a lattice and that we can generalize SampleG to this lattice. The set $\Lambda^{\perp}(\underline{\mathbf{g}}^t)$ is indeed a lattice if we identify R^{ν} with $\mathbb{Z}^{\nu \cdot N}$, since it is closed under addition and thus a discrete subgroup of $\mathbb{R}^{\nu \cdot N}$. In fact, we can write $\Lambda^{\perp}(\hat{G}) = \{\underline{v} \in \mathbb{Z}^{\nu \cdot N} \mid \underline{v} \cdot \hat{G} = 0 \bmod q\}$ with $\hat{G} = \underline{g} \otimes I_N$, which is the same lattice. This is because the multiplication matrix of $\mathbf{g}_i$, where $\mathbf{g}_i \in \mathbb{Z} \subset R$ is just a constant, is the matrix $g_i I_N$. Let B be a basis for $\Lambda^{\perp}(\underline{g}^t \in \mathbb{Z}^{\nu})$. Then $B \otimes I_N$ is a basis for $\Lambda^{\perp}(\hat{G})$. For such a structured matrix, the sampling algorithm from [Kle00] reduces to N independent copies of the same algorithm on each coordinate with base B (as claimed in [BI22]). Hence, SampleG can be used to sample from $\mathcal{D}_{\Lambda^{\perp}(\underline{\mathbf{g}}^t) \in R^{\nu} + \mathbf{c}, r}$, provided that $r > \eta_{\epsilon}(\Lambda^{\perp}(\hat{G}))$.

In [MP12] it was shown that all vectors in B have length less than or equal to $\sqrt{1 + 2^{\beta}}$ and due to the structure of the basis $B \otimes I_N$ it follows that $\lambda_{\nu \cdot N}(\Lambda^{\perp}(\underline{\mathbf{g}}^t \in R^{\nu})) =$

$\lambda_{\nu \cdot N}(\Lambda^{\perp}(\hat{G})) \leq \sqrt{1+2^{\beta}}$.

The generalization of decomposition to vectors over R now follows in the same way as generalizing $\mathsf{SampleG}$ to vectors over $\mathbb{Z}$. One simply observes that $G = I_k \otimes \underline{\mathbf{g}}^t \in R^{k \cdot \nu \times k}$ and apply the randomized R-decomposition algorithm from above on each coordinate. The following fact follows directly from the structure of the basis for $\Lambda^{\perp}(G)$ and is useful to bound its smoothing parameter in our proofs.

Fact 1.

$$\lambda_{k \cdot \nu \cdot N}(\Lambda^{\perp}(G \in R^{k \cdot \nu \times k})) \leq \sqrt{1+2^{\beta}} \ .$$

3.2 Sanitization Algorithms

Denote by $G_r^{-1}(\cdot)$ the randomized decomposition from Section 3.1. We define $\boxdot_r$ to be the external product using $G_r^{-1}(\cdot)$ instead of radix decomposition. This allows us to define a variant of blind rotation whose output is simulatable; the algorithm being given in Figure 4 and denoted by $\mathsf{Sanitized\text{-}BR}(\mathfrak{ct}, \mathsf{bsk}, \mathbf{t})$, which is essentially that given in [BI22]. This algorithm will form the core of our threshold decryption method, as well as our solution to providing funcCPA security for TFHE. The algorithm is the randomized version of the traditional Blind Rotation method given in Figure 2.

For the sake of modularity, we present the algorithms here in a form, where they require access to an oracle that generates fresh GLWE samples for a secret corresponding to the GLWE secret key in the bootstrapping key bsk. We discuss in Section 3.3 how to instantiate the oracle by augmenting the bsk.

$\mathsf{Sanitized\text{-}BR}^{\mathcal{O}_{\underline{\tilde{\mathbf{s}}},r}()}(\mathfrak{ct}, \mathsf{bsk}, \mathbf{t})$

On input of an LWE ciphertext $\mathfrak{ct} = (\underline{a}, b) \in \mathbb{Z}_{2 \cdot N}^{\ell+1}$ encrypting a message m, a bootstrapping key $\mathsf{bsk} = (\mathsf{bsk}_i)_{i=1}^{\ell}$, and a test polynomial $\mathbf{t} \in R_q$ this outputs a GLWE ciphertext $\mathfrak{ct}'$ encrypting $X^{-m-e} \cdot t$. Note that $\mathcal{O}_{\underline{\tilde{\mathbf{s}}},r}()$ is an oracle that generates fresh GLWE samples for secret $\underline{\tilde{\mathbf{s}}}$ and Gaussian parameter $r \cdot \sqrt{E \cdot E^t + I}$ for some E of bounded size.

1. $\underline{\mathbf{z}} \leftarrow (\mathbf{0}, \ldots, \mathbf{0}, \mathbf{t} \cdot X^{-b}) \in R_q^{k+1}$.
2. For $i = 1$ to ℓ do:
   (a) $\underline{\mathbf{z}} \leftarrow \underline{\mathbf{z}} + \mathsf{bsk}_i \boxdot_r ((X^{a_i} - 1) \cdot \underline{\mathbf{z}})$.
3. $(\underline{\mathbf{a}}', \mathbf{b}') \leftarrow \mathcal{O}_{\underline{\tilde{\mathbf{s}}},r}()$.
4. Return $\underline{\mathbf{z}} + (\underline{\mathbf{a}}', \mathbf{b}')$.

Figure 4: The Sanitized Blind Rotation Algorithm.

The work of [BI22] shows that the version of the blind rotation given by $\mathsf{Sanitized\text{-}BR}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{pk}, \mathbf{t})$ yields a distribution that only depends on the input plaintext, not the ciphertext. In Section 4 we will show something stronger: namely, that we can precisely define the output distribution of $\mathsf{Sanitized\text{-}BR}(\mathfrak{ct}, \mathsf{bsk}, \mathbf{t})$ and easily simulate it without having to run the algorithm itself as long as we know the underlying message. Thus we obtain the simulatable sanitization algorithm in Figure 5.

3.3 Instantiating the GLWE Oracle

The goal of this section is to show how to delegate the generation of GLWE samples with secret $\underline{\tilde{\mathbf{s}}}$ without revealing $\underline{\tilde{\mathbf{s}}}$. This is similar to the notion of *rerandomizers* from

$\mathsf{Sanitize}^{\mathcal{O}_{\underline{\tilde{\mathbf{s}}},r}()}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{ksk}, \mathbf{t}, \mathsf{pk})$

1. If $\mathfrak{ct}$ is a GLWE ciphertext of dimension $k \cdot N$ apply $\mathfrak{ct} \leftarrow \mathsf{KeySwitch}(\mathfrak{ct}, \mathsf{ksk})$; otherwise $\mathfrak{ct}$ is an LWE ciphertext of dimension ℓ.
2. $\mathfrak{ct} \leftarrow \mathsf{ModSwitch}(\mathfrak{ct})$.
3. $\mathfrak{ct} \leftarrow \mathsf{Sanitized\text{-}BR}^{\mathcal{O}_{\underline{\tilde{\mathbf{s}}},r}()}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{pk}, \mathbf{t})$.
4. $\mathfrak{ct} \leftarrow \mathsf{Flatten}(\mathfrak{ct})$.
5. Return $\mathfrak{ct}$.

Figure 5: Santization Algorihtm

[BI22, DS16], but with stronger requirements. While [DS16] only requires the rerandomizer to inject some randomness to slightly bridge the gap between two ciphertexts encrypting the same message (which is then repeated multiple times to obtain statistical closeness), [BI22] requires that the mask of the rerandomizer is (statistically or computationally close to) uniformly random (even in the presence of the secret key $\underline{\tilde{\mathbf{s}}}$). In our case, we also require the mask to be uniformly random, but we also need the noise distribution to be statistically close to a Gaussian.

In the following, we will show two methods to instantiate the GLWE oracle, both based on the work of [BI22]. The first one will produce samples that are statistically close to fresh GLWE samples, the other will be computationally indistinguishable. In combination with $\mathsf{Sanitize}$ they yield a statistical and a computational sanitization algorithm, respectively. The two algorithms work in a similar fashion: publishing a set of GLWE samples allows to generate fresh samples by taking random linear combinations.

3.3.1 Statistical Instantiation:

A statistical version of the GLWE oracle is already implicit in [BI22].

Lemma 6. *Let $\epsilon, \epsilon' > 0$, $\ell' = \ell \cdot (\ell + 1)$ be such that*

$$\ell' \geq \log\left(\frac{((N \cdot \log q) - 1) \cdot (1 + \epsilon)}{4 \cdot \epsilon'^2 \cdot (1 - \epsilon)} + \ell\right) .$$

For any $\underline{\tilde{\mathbf{s}}} \in R^{\ell}$ and $\underline{\mathbf{e}} \in R^{\ell'}$, let $(\mathbf{A}, \underline{\mathbf{b}} = \mathbf{A} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}) \in R_q^{\ell' \times (\ell+1)}$, where $\mathbf{A} \in R^{\ell' \times \ell}$ is uniformly random. Let r be such that

$$r \geq (1 + \sqrt{N} \cdot \|\underline{\mathbf{e}}\|_2) \cdot \sqrt{\frac{\ln\left(2 \cdot \ell' \cdot N \cdot (1 + 1/\epsilon)\right)}{\pi}} .$$

Then

$$\mathrm{SD}\left((\underline{\mathbf{r}} \cdot \mathbf{A}, \underline{\mathbf{r}} \cdot \underline{\mathbf{b}} + \mathbf{e}''), (\underline{\mathbf{a}}, \underline{\mathbf{a}} \cdot \underline{\tilde{\mathbf{s}}} + \mathbf{e})\right) \leq \epsilon' + \epsilon$$

where $\underline{\mathbf{r}} \leftarrow \mathcal{D}_{R^{\ell'},r}$, $\mathbf{e}'' \leftarrow \mathcal{D}_{R,r}$, $\underline{\mathbf{a}}$ is uniformly random and $\mathbf{e} \leftarrow \mathcal{D}_{R,\Gamma}$ with

$$\Gamma = r \cdot \sqrt{E \cdot E^t + I}$$

and E being the multiplication matrix of $\underline{\mathbf{e}}$.

3.3.2 Computational Instantiation:

We now extend the approach from [BI22]. We show that their rerandomizer not only yields pseudorandom masks, but also that the resulting noise distribution is statistically close to

a (non-spherical) discrete Gaussian. We simplify the technique a little at the expense of slightly worse parameters by assuming the same distribution for the randomizing terms below. This is for ease of exposition and can easily be generalized to distributions with different noise parameters for $\underline{\mathbf{r}}$, $\underline{\mathbf{e}}'$ and $\mathbf{e}$.

Lemma 7. *For any $\underline{\tilde{\mathbf{s}}} \in R^{\ell}$ and $\underline{\mathbf{e}} \in R^{\ell'}$, let $(\mathbf{A}, \underline{\mathbf{b}} = \mathbf{A} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}) \in R_q^{\ell' \times (\ell+1)}$, where $\mathbf{A} \in R^{\ell' \times \ell}$ is uniformly random. Let r be such that*

$$r \geq (1 + \sqrt{N} \cdot \|[\underline{\tilde{\mathbf{s}}} \mid \underline{\mathbf{e}}]\|_2) \cdot \sqrt{\frac{\ln\left(2 \cdot (\ell' + \ell) \cdot N \cdot (1 + 1/\epsilon)\right)}{\pi}}$$

and define

$$r' = \frac{\sqrt{2} \cdot r}{\sqrt{1 + N \cdot \|[\underline{\tilde{\mathbf{s}}} \mid \underline{\mathbf{e}}]\|_2^2}} \geq \frac{\sqrt{2} \cdot r}{1 + \sqrt{N} \cdot \|[\underline{\tilde{\mathbf{s}}} \mid \underline{\mathbf{e}}]\|_2} \geq \sqrt{2} \cdot \eta_{\epsilon}\left(\mathbb{Z}^{(\ell'+\ell)\cdot N}\right) \quad .$$

Then, assuming the hardness of $\mathsf{GLWE}_{N,q,\ell',r'}$,

$$(\underline{\mathbf{r}} \cdot \mathbf{A} + \underline{\mathbf{e}}', \underline{\mathbf{r}} \cdot \underline{\mathbf{b}} + \mathbf{e}'') \approx_c (\underline{\mathbf{a}}, \underline{\mathbf{a}} \cdot \underline{\tilde{\mathbf{s}}} + \mathbf{e})$$

where $\underline{\mathbf{r}} \leftarrow \mathcal{D}_{R^{\ell'},r}$, $\underline{\mathbf{e}}' \leftarrow \mathcal{D}_{R^{\ell},r}$, $\mathbf{e}'' \leftarrow \mathcal{D}_{R,r}$, $\underline{\mathbf{a}}$ is uniformly random and $\mathbf{e} \leftarrow \mathcal{D}_{R,\Gamma}$ with

$$\Gamma = r \cdot \sqrt{E \cdot E^t + I}$$

and E being the multiplication matrix of $[-\underline{\tilde{\mathbf{s}}} \mid \underline{\mathbf{e}}]$.

The proof relies on a lemma from [BI22] reproduced as Lemma 8 below.

Proof. First note that the lower bound on r implies that r' meets the condition of Lemma 8. So it remains to show that $\mathbf{e}_u = \underline{\mathbf{r}} \cdot \underline{\mathbf{e}} - \underline{\mathbf{e}}' \cdot \underline{\tilde{\mathbf{s}}} + \mathbf{e}''$ is statistically close to $\mathcal{D}_{R,\Gamma}$. This follows directly from Lemma 4 with $\Lambda = R^{\ell'+\ell}$. □

Lemma 8 (adapted from [BI22], Lemma 14)**.** *With the notation of Lemma 7, if*

$$r' \geq \sqrt{2} \cdot \eta_{\epsilon}\left(\mathbb{Z}^{(\ell'+\ell)\cdot N}\right)$$

then, assuming the hardness of $\mathsf{GLWE}_{N,q,\ell',r'}$,

$$(\underline{\mathbf{r}} \cdot \mathbf{A} + \underline{\mathbf{e}}', \underline{\mathbf{r}} \cdot \underline{\mathbf{b}} + \mathbf{e}'') \approx_c (\underline{\mathbf{a}}, \underline{\mathbf{a}} \cdot \underline{\tilde{\mathbf{s}}} + \mathbf{e}_u)$$

where $\mathbf{e}_u = \underline{\mathbf{r}} \cdot \underline{\mathbf{e}} - \underline{\mathbf{e}}' \cdot \underline{\tilde{\mathbf{s}}} + \mathbf{e}''$.

4 Analysis of $\mathsf{Sanitize}^{\mathcal{O}_{\underline{\tilde{\mathbf{s}}},r}()}(\mathsf{ct}, \mathsf{bsk}, \mathsf{ksk}, \mathsf{t}, \mathsf{pk})$

Recall that the bootstrapping key bsk consists of GGSW encryptions of each bit s_i of the secret key. These ciphertext are encryption under another secret key, which we will denote by $\underline{\tilde{\mathbf{s}}}$ throughout this section. We begin with the analysis of $\mathsf{Sanitized\text{-}BR}$.

Theorem 1. *Let B_{bsk} be such that $\|\underline{\mathbf{e}}_i\|_2 \leq B_{\mathsf{bsk}}$ for all i, where $\underline{\mathbf{e}}_i$ corresponds to the vector of noise polynomials in the i-th element of the bootstrapping key bsk. Under the condition that*

$$r \geq \sqrt{1 + 2^{\beta}} \cdot (1 + \sqrt{N} \cdot B_{\mathsf{bsk}}) \cdot \sqrt{\frac{\ln\left(2 \cdot (k+1) \cdot \nu \cdot N \cdot (1 + 1/\epsilon)\right)}{\pi}}$$

and r satisfies the condition of Lemma 6 or Lemma 7 (for statistical or statistical sanitization, respectively) we have

$$\mathsf{Sanitized\text{-}BR}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{pk}, \mathbf{t}; \cdot) \approx (\underline{\mathbf{u}}, \underline{\mathbf{u}} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}' + \mathbf{t} \cdot X^{-b+\sum_i a_i \cdot s_i})$$

where $\underline{\mathbf{u}} \in R_q^k$ is uniformly random and $\underline{\mathbf{e}}' \leftarrow \mathcal{D}_{R,\Gamma}$. Here,

$$\Gamma = r \cdot \sqrt{\sum_i^{\ell+1} E_i \cdot E_i^t}$$

with r being the noise parameter of the randomized decomposition algorithm, E_i the multiplication matrix of $\underline{\mathbf{e}}_i$ and $E_{\ell+1} = \sqrt{E \cdot E^t + I}$ is the noise matrix from the LWE oracle.

The proof is inspired by [BI22].

Proof. Let $\underline{\mathrm{z}}_t$ be the state of the accumulator after t iterations. A short derivation shows:

$$\underline{\mathrm{z}}_t = X^{a_t \cdot s_t} \cdot \underline{\mathrm{z}}_{t-1} + (\mathsf{bsk}_i - s_i \cdot G) \boxdot_r ((X^{a_t} - 1)\underline{\mathrm{z}}_{t-1}) \ .$$

For convenience we define $\underline{\mathrm{z}}_{\ell+1} = \underline{\mathrm{z}}_n + (\underline{\mathbf{u}},\ \underline{\mathbf{u}} \cdot \underline{\tilde{\mathbf{s}}} + y)$ (where $y \leftarrow \mathcal{D}_{\mathbb{Z}, r \cdot E_{\ell+1}}$), i.e. the output of $\mathsf{Sanitized\text{-}BR}(\mathfrak{ct}, \mathsf{bsk}, \mathbf{t})$. Following the structure of the proof of Lemma 17 in [BI22], we show that for all t

$$\underline{\mathrm{z}}_{\ell+1} \overset{\bar{\epsilon}_t}{\approx} X^{\sum_{j>t} a_j \cdot s_j} \cdot \underline{\mathrm{z}}_t + (\underline{\mathbf{u}},\ \underline{\mathbf{u}} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}})$$

where for $\bar{\epsilon}_t = (\ell + 1 - t) \cdot 2 \cdot \epsilon/(1-\epsilon)$, $\underline{\mathbf{u}} \in R_q^k$ is uniformly random and $\underline{\mathbf{e}} \leftarrow \mathcal{D}_{R,\Gamma_t}$ with

$$\Gamma_t = r \cdot \sqrt{\sum_{i \geq t} E_i \cdot E_i^t} \ .$$

The proof is by induction from $\ell + 1$ to zero. Clearly, the statement is true for $\ell + 1$. So assume it holds for some t. We have

$$\begin{aligned}
\underline{\mathrm{z}}_{\ell+1} &\overset{\bar{\epsilon}_t}{\approx} X^{\sum_{j>t} a_j \cdot s_j} \cdot \underline{\mathrm{z}}_t + (\underline{\mathbf{u}},\ \underline{\mathbf{u}} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}) \\
&= X^{\sum_{j \geq t} a_j \cdot s_j} \cdot \underline{\mathrm{z}}_{t-1} + X^{\sum_{j>t} a_j \cdot s_j} \cdot \left((\mathsf{bsk}_t - s_t \cdot G) \boxdot_r ((X^{a_t} - 1) \cdot \underline{\mathrm{z}}_{t-1})\right) \\
&\qquad + (\underline{\mathbf{u}},\ \underline{\mathbf{u}} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}) \\
&= X^{\sum_{j \geq t} a_j \cdot s_j} \cdot \underline{\mathrm{z}}_{t-1} + (\mathsf{bsk}_t - s_t \cdot G) \boxdot_r (X^{\sum_{j>t} a_j \cdot s_j}(X^{a_t} - 1) \cdot \underline{\mathrm{z}}_{t-1}) \\
&\qquad + (\underline{\mathbf{u}},\ \underline{\mathbf{u}} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}) \ .
\end{aligned}$$

The last step follows since for any $\underline{\mathbf{v}}$ and i we have $X^i \cdot G_r^{-1}(\underline{\mathbf{v}}) = G_r^{-1}(X^i \cdot \underline{\mathbf{v}})$ (i.e. we may first apply a negacyclic shift and then decompose, or first decompose and then apply the negacyclic shift to all components). Define $\underline{\mathbf{v}} = (X^{\sum_{j>t} a_j \cdot s_j}(X^{a_t} - 1) \cdot \underline{\mathrm{z}}_{t-1})$. We need to show that

$$(\mathbf{A}_i,\ \mathbf{A}_i \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}_i) \boxdot_r \underline{\mathbf{v}} + (\underline{\mathbf{u}},\ \underline{\mathbf{u}} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}) \overset{\bar{\epsilon}'}{\approx} (\underline{\mathbf{u}},\ \underline{\mathbf{u}} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}})$$

(where the distribution of $\underline{e}$ has standard deviation Γ_t on the left hand side and Γ_{t-1} on the right hand side) with $\bar{\epsilon}' = 2 \cdot \epsilon/(1-\epsilon)$. Note that $\bar{\epsilon}_{t-1} = \bar{\epsilon}_t + \bar{\epsilon}'$, so the result will follow by triangle inequality. We have

$$\begin{aligned}
&(\mathbf{A}_i,\ \mathbf{A}_i \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}_i) \boxdot_r \underline{\mathbf{v}} + (\underline{\mathbf{u}},\ \underline{\mathbf{u}} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}) \\
=&(\mathbf{A}_i \boxdot_r \underline{\mathbf{v}} + \underline{\mathbf{u}},\ (\mathbf{A}_i \boxdot_r \underline{\mathbf{v}} + \underline{\mathbf{u}}) \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}_i \boxdot_r \underline{\mathbf{v}} + \underline{\mathbf{e}}) \\
=&(\underline{\mathbf{u}},\ \underline{\mathbf{u}} \cdot \underline{\tilde{\mathbf{s}}} + \underline{\mathbf{e}}_i \boxdot_r \underline{\mathbf{v}} + \underline{\mathbf{e}})
\end{aligned}$$

where the last step follows from the uniformity of $\underline{\mathbf{u}}$. We conclude by applying Lemma 4 to $\underline{\mathbf{e}}_i \boxdot_r \underline{\mathbf{v}} + \underline{\mathbf{e}}$: Note that the marginal distribution of this term is $\underline{\mathbf{e}}^t \mathcal{D}_{\Lambda^\perp(G) + G^{-1}(\underline{\mathbf{v}}),\ r \cdot I} + \mathcal{D}_{R,\Gamma_t}$. Lemma 5 and Fact 1 ensure that Γ_t meets the conditions of Lemma 4 and thus we conclude that $\underline{\mathbf{e}}_i \boxdot_r \underline{\mathbf{v}} + \underline{\mathbf{e}} \overset{\bar{\epsilon}'}{\approx} \mathcal{D}_{R,\Gamma_{t-1}}$. □

Finally, to obtain an LWE ciphertext from the result of $\mathsf{Sanitized\text{-}BR}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{pk}, \mathbf{t})$, the algorithm $\mathsf{Sanitize}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{ksk}, \mathbf{t}, \mathsf{pk})$ flattens the GLWE ciphertext.

Theorem 2. *Let $\underline{\mathbf{e}} \leftarrow \mathcal{D}_{R,\Gamma}$ with r and Γ as in Theorem 1, and $\mathcal{X}$ be the marginal distribution of e_1. Then*

$$\mathcal{X} \overset{\bar{\epsilon}}{\approx} \mathcal{D}_{\mathbb{Z},r_{\mathsf{br}}}$$

where $\bar{\epsilon} = 2 \cdot \epsilon/(1-\epsilon)$, $r_{\mathsf{br}} = r \cdot \sqrt{\sum \|\underline{\mathbf{e}}'\|_2^2}$ and $\underline{\mathbf{e}}'$ runs over the all noise polynomials in the bootstrapping key, the possible noise vectors from the GLWE oracle, and the secret key polynomials $\tilde{\underline{\mathbf{s}}}$ in case of the computational instantiation from Lemma 7.

Proof. Apply Lemma 3 with $T = (1, 0, \ldots, 0)$. Note that $\ker(T) = 0 \otimes \mathbb{Z}^{N-1}$, which is clearly a lattice subspace of $\mathbb{Z}^N$. By Lemma 5 we have that $S = \Gamma$ has full column rank and $\sigma_N(S) \geq r$. Since

$$\eta_\epsilon\left(\ker(T) \cap \mathbb{Z}^N\right) = \eta_\epsilon\left(\mathbb{Z}^{N-1}\right) = \eta_\epsilon(\mathbb{Z}) \leq \sqrt{\frac{\ln\left(2 \cdot (N-1) \cdot (1 + 1/\epsilon)\right)}{\pi}} \leq r,$$

the result follows from

$$r_{\mathsf{br}} = r \cdot \sqrt{\sum_i T \cdot E_i \cdot E_i^t \cdot T} = r \cdot \sqrt{\sum \|\underline{\mathbf{e}}'\|_2^2} \ .$$

□

Note that Theorem 2 implies that we can easily simulate the output distribution of $\mathsf{Sanitize}^{\mathcal{O}_{\tilde{\underline{\mathbf{s}}},r}()}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{ksk}, \mathbf{t}, \mathsf{pk}; \cdot)$ knowing only the message and r_{br}.

Corollary 1. *If $\mathbf{t}$ is such that the constant coefficient of $\mathbf{t} \cdot X^{-(m+e)}$ is $\Delta \cdot m$ for all m in the message space and r satisfies the conditions in Theorem 1 and Lemma 7, then the output of $\mathsf{Sanitize}^{\mathcal{O}_{\tilde{\underline{\mathbf{s}}},r}()}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{ksk}, \mathbf{t}, \mathsf{pk}; \cdot)$ is indistinguishable from $(\underline{u}, b = \underline{u} \cdot \tilde{\underline{s}} + e + \Delta \cdot m)$ under the GLWE assumption, where $\underline{u} \in \mathbb{Z}_q^N$ is uniformly random, $e \leftarrow \mathcal{D}_{\mathbb{Z},r_{\mathsf{br}}}$, and $\tilde{\underline{s}}$ corresponds to the flattening of the GLWE key.*

A similar statement can be made using Lemma 6, achieving statistical indistinguishability.

5 Threshold FHE

The application of our simulatable sanitization method to server assisted threshold TFHE is now relatively immediate. The protocol and proof follow almost identically to that in [PS24], except that the large flooding operation in the protocol is now replaced by our $\mathsf{Sanitize}$ procedure. Our protocols are therefore in a system in which we have $\mathfrak{n}+1$ parties, consisting of a server $\mathcal{S}$ and the threshold parties $\mathcal{P}_1, \ldots, \mathcal{P}_\mathfrak{n}$. The assumption is that the server $\mathcal{S}$ is always honest, however up to $\mathfrak{t}$ of the threshold parties may be (potentially malicously) dishonest.

5.1 Threshold Decryption Ideal Functionalities

Following [DDK+23] we define the threshold decryption via two ideal functionalities, as opposed to the game-based definitions in [PS24]. Unlike [DDK+23], as we are in a server aided situation, the ideal functionalities implicitly assume that the protocol runs between the parties $\{\mathcal{S}, \mathcal{P}_1, \ldots, \mathcal{P}_\mathfrak{n}\}$, for which party $\mathcal{S}$ is always an honest party. The first $\mathcal{F}_{\mathsf{KeyGen}}$, in Figure 6, acts as a set-up assumption for our protocol, needed for the UC proof we

provide. It generates a key pair (where the public key consists of the public encryption key pk, the bootstrapping key bsk and the keyswitching key ksk), and secret shares the secret $\underline{s}$ key for the flattened GLWE ciphertexts among the players using the secret sharing scheme. It also generates, and returns to the players, the value r_{br}, since this value will be needed in our simulator for threshold decryption[7]. As explained in [DDK+23] one can realize this functionality (between the parties $\mathcal{P}_1, \ldots, \mathcal{P}_{\mathfrak{n}}$) using a generic MPC protocol relatively simply; for example using full threshold secret sharing one can implement a non-robust version of this functionality using the methodology given in [RST+22].

$\mathcal{F}_{\mathsf{KeyGen}}$

Init:

1. On input of Init from all parties.
2. Execute $(\{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}\}, \underline{s}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ for the underlying TFHE encryption scheme.
3. Compute $r_{\mathsf{br}} \leftarrow r \cdot \sqrt{\sum \|\underline{\mathbf{e}}_i\|_2^2}$.
4. Generate a secret sharing $\langle \underline{s} \rangle$ of the secret key.
5. Send $\{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}, r_{\mathsf{br}}\}$ to all players (including the adversary), and send $\langle \underline{s} \rangle_i$ to player $\mathcal{P}_i$ (including adversarially controlled players).

Figure 6: The ideal functionality for distributed key generation

The key functionality we want to implement is $\mathcal{F}_{\mathsf{KeyGenDec}}$ given in Figure 7. Note, that this functionality always returns the correct result. If the adversary is passive then by definition it will return the correct result. If the adversary is actively malicious (which we will consider in Section 5.3) then we need the protocol to ensure that the correct result is returned.

$\mathcal{F}_{\mathsf{KeyGenDec}}$

Init:

1. On input of Init from all parties.
2. Execute $(\{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}\}, \underline{s}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ for the underlying TFHE encryption scheme.
3. Send $\{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}\}$ to all players, including the adversary and store the value $\underline{s}$.

ThreshDec:

1. On input of $\mathsf{ThreshDec}(\mathfrak{ct}, \mathcal{U})$ from all parties, where $\mathfrak{ct}$ is a valid (i.e. decryptable) ciphertext and $\mathcal{U}$ is a given party who should receive the plaintext message encrypted by $\mathfrak{ct}$.
2. Compute $m \leftarrow \mathsf{Dec}(\mathfrak{ct}, \underline{s})$.
3. If $\mathcal{U}$ is adversarially controlled then send $(\mathfrak{ct}, m)$ to the adversary.
4. Otherwise send m to player $\mathcal{U}$ and $\mathfrak{ct}$ to the adversary.

Figure 7: The ideal functionality for distributed key generation and decryption

[7] The actual protocol does not need to compute or publish this value. However, publishing this value does not affect security.

5.2 Full Threshold Version (Non-Robust)

We assume a threshold secret sharing scheme $\langle\cdot\rangle$ which allows us to share values in $\mathbb{Z}_q$ amongst $\mathfrak{n}$ parties. In this section we assume that at most $\mathfrak{t} = \mathfrak{n} - 1$ of these parties can be **passively** corrupted. Such a secret sharing scheme can be obtained using a trivial additive secret sharing scheme; i.e. a secret $\underline{s} \in \mathbb{Z}_q^{k \cdot N}$ is secret shared $\langle \underline{s} \rangle$ by giving each player $\underline{s}_i \in \mathbb{Z}_q^{k \cdot N}$ where

$$\underline{s} = \underline{s}_1 + \ldots + \underline{s}_{\mathfrak{n}}.$$

We note that in this situation by adding zero-knowledge proofs to our protocol, much like as in the protocol in [ABGS23], one can obtain (in our threshold decryption application) active-with-identifiable-abort security; i.e. the receiving party will know which dishonest parties tried to add on a non-zero value of Γ to the final decryption result. This modified protocol is, however, only efficient when a large number of threshold decryptions are required, and so we will not consider it further here.

ThreshDec$(\mathfrak{ct}, \langle \underline{s} \rangle, \{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}\}, \mathcal{U})$

Init():

1. The parties $\mathcal{P}_1, \ldots, \mathcal{P}_{\mathfrak{n}}$ obtain, via a generic MPC protocol, the sharing $\langle \underline{s} \rangle$ of the value $\underline{s}$.

ThreshDec$(\mathfrak{ct}, \langle \underline{s} \rangle, \mathcal{U})$:

1. Party $\mathcal{S}$ executes $(\underline{a}', b' = \underline{a} \cdot \underline{s} + \Delta \cdot m + e') \leftarrow \mathsf{Sanitize}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{ksk}, \mathsf{enc}_{\mathsf{id}}, \mathsf{pk})$.
2. Party $\mathcal{S}$ sends $(\underline{a}', b')$ to all parties $\mathcal{P}_i$ and $\mathcal{U}$.
3. Party $\mathcal{P}_i$ samples $d_i \leftarrow \mathcal{D}_{\mathbb{Z},\eta}$.
4. Party $\mathcal{P}_i$ computes $p_i \leftarrow \underline{a}' \cdot \underline{s}_i + d_i \pmod q$.
5. Party $\mathcal{P}_i$ sends p_i to party $\mathcal{U}$.
6. Party $\mathcal{U}$ extracts m from $p' = b' - \sum_{i=1}^{\mathfrak{n}} p_i = \Delta \cdot m + e' - \sum_{i=1}^{\mathfrak{n}} d_i$.
7. Party $\mathcal{U}$ returns m.

Figure 8: The Full Threshold Threshold Decryption Protocol

The threshold decryption protocol is then immediate, and given in Figure 8. With the correctness and security being given by the following theorem.

Theorem 3. *In the $\{\mathcal{F}_{\mathsf{KeyGen}}\}$-hybrid model the protocol in Figure 8 implements $\mathcal{F}_{\mathsf{KeyGenDec}}$ with computational security against any static* **passive** *adversary corrupting I parties, with $|I| \leq \mathfrak{n} - 1$ assuming:*

- *TFHE parameters that ensure correctness of the FHE operations.*
- *$\left|e' + \sum_{i=1}^{\mathfrak{n}} d_i\right| < \Delta/2$ (to ensure correctness of the output of the "bath"-ing operation)*[8]*.*
- *The noise parameter r of* Sanitize *satisfies the conditions of Theorem 1 and Lemma 7.*
- *The hardness of $\mathsf{yaLWE}^{\mathcal{S}}_{q,k\cdot N,\sigma,\eta}$, where $\sigma = r_{\mathsf{br}}$ from Theorem 2.*

Proof. Correctness follows from the correctness of the TFHE homomorphic operations, the bound $|e'| + \sum_i d_i < \Delta/2$ and the correctness of the sanitization procedure.

[8] This inequality could be rephrased in terms of σ, η, $\mathfrak{n}$ and a suitably chosen constant derived from the erfc function.

Without loss of generality we can assume there is only one honest party, which we will assume is player $\mathcal{P}_{\mathfrak{n}}$. Security of the protocol follows by showing that the output of the simulator in Figure 9 is computationally indistinguishable, from the output of an adversary controlling the parties $\mathcal{P}_1, \ldots, \mathcal{P}_{\mathfrak{n}-1}$, in a real execution of the protocol.

Simulator Threshold Decryption

On input of

1. A ciphertext $\mathfrak{ct} = (\underline{a}, b)$ and the public keys $\{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}\}$.
2. The underlying message m encrypted by $\mathfrak{ct}$.
3. A set of adversarial parties I with $|I| \leq \mathfrak{n} - 1$.
4. The share values $\underline{s}_i$ for $i \in I$.
5. The value σ, which is a publicly known value from the key generation method.

this algorithm outputs the simulated shares $\{\langle p \rangle_j\}_{j \notin I}$.

Sim-DistDecrypt:

1. $\underline{a}' \leftarrow \mathbb{Z}_q^{k \cdot N}$.
2. $b' \leftarrow \mathbb{Z}_q$.
3. For $i = 1, \ldots, \mathfrak{n} - 1$ compute
   (a) $d_i \leftarrow \mathcal{D}_{\mathbb{Z},\eta}$.
   (b) $p_i \leftarrow \underline{a}' \cdot \underline{s}_i + d_i \pmod q$.
4. $h \leftarrow \mathcal{D}_{\mathbb{Z},\sqrt{\sigma^2+\eta^2}}$
5. $p_{\mathfrak{n}} = b' + h - \sum_{i=1}^{\mathfrak{n}-1} \underline{a}' \cdot \underline{s}_i - \Delta \cdot m$.
6. The simulator outputs $\{\underline{a}', b', p_1, \ldots, p_{\mathfrak{n}}\}$.

Figure 9: Simulator for $\mathsf{ThreshDec}(\mathfrak{ct}, \langle \underline{s} \rangle, \{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}\}, \mathcal{U})$

Note, the values $\{p_1, \ldots, p_{\mathfrak{n}-1}\}$ produced by the simulator are the true decryption share values produced by the adversary. Our proof of security follows the proof of Theorem 5.1 of [PS24], via a sequence of hybrids. We let Hyb_0 denote the situation that the UC distinguisher sees when the adversary is interacting with the real world.

In Hyb_1 we alter the way in which server $\mathcal{S}$ produces the values $(\underline{a}', b')$; instead of calling $\mathsf{Sanitize}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{ksk}, \mathsf{enc}_{\mathsf{id}}, \mathsf{pk})$ the ciphertext $(\underline{a}', b')$ is generated as follows:

- $\underline{a}'$ is sampled at random from $\mathbb{Z}_q^{k \cdot N}$.
- b' is computed from $b' \leftarrow \underline{a}' \cdot \underline{s} + p$ where $p = \Delta \cdot m + \mathcal{D}_{\mathbb{Z},\sigma}$.

Note, in creating this hybrid we can assume the secret key is known, as we are trying to show that the distinguisher cannot tell the difference if this change is made. The indistinguishability of Hyb_0 and Hyb_1 follows directly from Corollary 1.

In Hyb_1 we have that the following equation holds:

$$\Delta \cdot m + e' = b' - \sum_{i=1}^{\mathfrak{n}} \underline{a}' \cdot \underline{s}_i,$$

i.e.

$$\underline{a}' \cdot \underline{s}_{\mathfrak{n}} = b' - \sum_{i=1}^{\mathfrak{n}-1} \underline{a}' \cdot \underline{s}_i - \Delta \cdot m - e',$$

i.e.

$$p_{\mathfrak{n}} - d_{\mathfrak{n}} = b' - \sum_{i=1}^{\mathfrak{n}-1} \underline{a}' \cdot \underline{s}_i - \Delta \cdot m - e',$$

i.e.

$$p_{\mathfrak{n}} = b' + d_{\mathfrak{n}} - e' - \sum_{i=1}^{\mathfrak{n}-1} \underline{a}' \cdot \underline{s}_i - \Delta \cdot m.$$

Thus, in Hyb_2 we now make a further change in how the values are sampled

- b' is sampled at random from $\mathbb{Z}_q$.
- $p_{\mathfrak{n}}$ is sampled via

$$p_{\mathfrak{n}} = b' + h - \sum_{i=1}^{\mathfrak{n}-1} \underline{a}' \cdot \underline{s}_i - \Delta \cdot m$$

This last substitution replaces the $d_{\mathfrak{n}} - e'$ term with h where $h \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_h}$ with

$$\sigma_h = \sqrt{\sigma^2 + \eta^2}.$$

It is clear that Hyb_2 now corresponds to the view of the disginguisher when the adversary is interacting in the ideal world with the simulator.

Thus we only need to show that Hyb_1 and Hyb_2 are indistinguishable. This follows from the $\mathsf{yaLWE}^{\mathcal{S}}_{q,k\cdot N,\sigma,\eta}$ hardness assumption of Definition 3. To see this, recall that in Hyb_1 the view of the environment is[9] $(\underline{a}', b', p_{\mathfrak{n}})$, where $\underline{a}'$ is uniformly random, $b' = \sum_{i=1}^{\mathfrak{n}} \underline{a}' \cdot \underline{s}_i + \Delta \cdot m + e'$ and $p_{\mathfrak{n}} = \underline{a}' \cdot \underline{s}_{\mathfrak{n}} + d_{\mathfrak{n}}$. Since the adversary knows $\Delta \cdot m$ and $\underline{s}_i$ for all $i < \mathfrak{n}$, we may remove the term $\alpha = \sum_{i=1}^{\mathfrak{n}-1} \underline{a}' \cdot \underline{s}_i + \Delta \cdot m$ from the second component and are left with $(\underline{a}', \underline{a}' \cdot \underline{s}_{\mathfrak{n}} + e', \underline{a}' \cdot \underline{s}_{\mathfrak{n}} + d_{\mathfrak{n}})$. By the yaLWE assumption (cf. Definition 3), this is indistinguishable from $(\underline{a}', u, u + h)$, where u is uniformly random and $h \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_h}$. Setting $b' \leftarrow u + \alpha$, we have that $(\underline{a}', b', u + h = b' - \alpha + h)$ corresponds to the view of the environment in Hyb_2, which shows that the two hybrids are indistinguishable under the assumption made in Definition 3. □

5.3 Extension to Thresholds $\mathfrak{t} < \mathfrak{n}/2$ and $\mathfrak{t} < \mathfrak{n}/3$

A natural question is whether one can apply the above methods when up to $\mathfrak{t} < \mathfrak{n}$ can be corrupted. The main difference between the full threshold case, i.e. $\mathfrak{t} = \mathfrak{n} - 1$, and the general case is how the "bathing" terms d_i are added on. In the full threshold version above the values d_i are selected by each player. They are small enough not to affect the correctness, but they are large enough to ensure secrecy of each players secret share $\underline{s}_i$, and hence of the entire secret $\underline{s}$. However, using such player dependent masking terms d_i would result in the secret sharing scheme being invalid in the case of other forms of secret sharing, since the share recovery equation for other secret sharing schemes is not the sum of all other players partial decryptions.

Thus, following [DDK+23], we produce a single, player independent, mask value d using either a Pseudo-Random Secret Sharing (PRSS) scheme (when $\binom{\mathfrak{n}}{\mathfrak{t}}$ is "small"), or using an offline MPC phase (when $\binom{\mathfrak{n}}{\mathfrak{t}}$ is "large"). We descibe the PRSS based method here, and refer the reader to [DDK+23] for the offline MPC based approach.

[9] We ignore p_i for $i < \mathfrak{n}$ since they are generated identically in both hybrids and independently of b' and $p_{\mathfrak{n}}$.

5.3.1 Shamir Sharing over Rings:

Secret sharing, for an arbitrary threshold $\mathfrak{t} < \mathfrak{n}$, for secrets in the ring $\mathbb{Z}_q$, as explained in [DDK+23], can be accomplished by (potentially) taking a Galois ring extension and applying Shamir sharing over this extension in a relatively standard manner; see [ACD+19, Feh98, JSvL22] for more details. The important point to note is, the interpolation points for the Shamir Secret Sharing scheme $\{\gamma_1, \ldots, \gamma_{\mathfrak{n}}\}$ form an exceptional set in the underlying Galois ring extension.

The underlying secret key $\underline{s} \in \mathbb{Z}_q^{k \cdot N}$ for our flattened GLWE ciphertexts of dimension $k \cdot N$ we can then assume to be secret shared using the Shamir secret sharing scheme with threshold $\mathfrak{t}$. We shall denote this sharing of the vector $\underline{s}$ by $\langle \underline{s} \rangle$.

If $\mathfrak{t} < \mathfrak{n}/3$ then, using a method going back to [BCG93], the $\mathfrak{n}$ parties can send their shares to a given party $\mathcal{U}$, who can then robustly reconstruct an underlying value x which has been secret shared via $\langle x \rangle$; i.e. the party $\mathcal{U}$ can recover x irrespective of the malicious behaviour of the dishonest parties. The robust reconstruction protocol, which we shall denote by $\mathsf{RobustOpen}(\langle x \rangle, \mathcal{U})$ is a one round protocol and it works in both the synchronous and asynchronous network settings.

We note, but will not consider further in this paper, that if $\mathfrak{t} < \mathfrak{n}/2$ then the $\mathsf{RobustOpen}$ procedure can be replaced (trivially) by a procedure which provides active-with-abort security; i.e. the receiving honest party will abort if a malicious party sends an incorrect value. With this change the following robust methodology of threshold decryption for $\mathfrak{t} < \mathfrak{n}/3$ becomes an active-with-abort threshold decryption for $\mathfrak{t} < \mathfrak{n}/2$.

5.3.2 Pseudo-Random Secret Sharing:

Pseudo-Random Secret Sharing (PRSS) was introduced in [CDI05]. It enables parties to non-interactively generate a sharing of a random value. In [DDK+23], following an idea first introduced in [CLO+13], this is extended to generating a sharing of a "small" value as follows.

The algorithms for our PRSS are defined in Figure 10. The algorithm $\mathsf{PRSS.Init}()$ iterates over all sets A of size $\mathfrak{n} - \mathfrak{t}$. Thus the complexity of $\mathsf{PRSS.Init}()$, i.e. the number of sets A we need to deal with, depends on $\binom{\mathfrak{n}}{\mathfrak{t}}$, which can become very large for large $\mathfrak{n}$ and $\mathfrak{t}$. The PRSS makes use of a PRF ψ of the form

$$\psi : \begin{cases} \{0,1\}^{\mathsf{sec}} \times S & \longrightarrow & \mathbb{Z} \\ (\kappa, \mathsf{cnt}) & \longmapsto & \psi(\kappa, \mathsf{cnt}) \end{cases}$$

where $\{0,1\}^{\mathsf{sec}}$ is the keyspace and S is a set of counters. The output of the function ψ is assumed to be uniform in the range $[-B, \ldots, B]$. The shared value which is output by the $\mathsf{PRSS.Next}()$ invocation is then the sharing of the value

$$E \leftarrow \sum_A \psi(r_A, \mathsf{cnt}_{\mathsf{PRSS}}),$$

and it is sampled from the distribution formed by the sum of $\binom{\mathfrak{n}}{\mathfrak{t}}$ uniformly random values in $[-B, \ldots, B]$. However, from the adversarial point of view the adversary knows upto $\binom{\mathfrak{n}}{\mathfrak{t}} - 1$ of these values, and so there is potentially only one uniformly random value unknown to the adversary.

5.3.3 Robust Threshold Decryption when $\mathfrak{t} < \mathfrak{n}/3$:

We refer to Figure 11 as to how our threshold decryption protocol is modified. It is parametrized by a parameter w, which controls the generation of the bathing value $\langle d \rangle$. In particular d can be written as $d_a + d_h$ where d_a is known to the adversary, and d_h is unknown to the adversary. The value d_h is the sum of (at least) w unknown (to the

PRSS

PRSS.Init(): For every set $A \subseteq \{1, \ldots, \mathfrak{n}\}$ of size $\mathfrak{n} - \mathfrak{t}$:

1. $S \leftarrow \{\mathcal{P}_i\}_{i \in A}$.
2. Players $\mathcal{P}_i$ with $i \in A$ agree on a shared secret random value r_A; see the full version of [DDK+23] for how this can be done.
3. Define $f_A(X) \in \mathbb{Z}_q[X] = \mathbb{Z}/(2^k)[X]$ to be the polynomial of degree $\mathfrak{t}$ such that $f_A(0) = 1$ and $f_A(\gamma_i) = 0$ for all $i \notin A$. Each party $\mathcal{P}_i$ only needs store $f_A(\gamma_i)$ though.
4. $\mathsf{cnt}_{\mathsf{PRSS}} \leftarrow 0$.

PRSS.Next():

1. Party $\mathcal{P}_i$ computes, where the sum is over every set A containing i,
$$\langle E \rangle_i \leftarrow \sum_{A : i \in A} \psi(r_A, \mathsf{cnt}_{\mathsf{PRSS}}) \cdot f_A(\gamma_i).$$
2. $\mathsf{cnt}_{\mathsf{PRSS}} \leftarrow \mathsf{cnt}_{\mathsf{PRSS}} + 1$.
3. Return $\langle E \rangle$.

Figure 10: Pseudo-Random Secret Sharing PRSS

ThreshDec($\mathfrak{ct}, \langle \underline{s} \rangle, \{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}\}, \mathcal{U}$)

Init():

1. The parties $\mathcal{P}_1, \ldots, \mathcal{P}_{\mathfrak{n}}$ obtain, via a generic MPC protocol, the sharing $\langle \underline{s} \rangle$ of the value $\underline{s}$.
2. The parties execute PRSS.Init().

ThreshDec($\mathfrak{ct}, \langle \underline{s} \rangle, \mathcal{U}$):

1. Party $\mathcal{S}$ executes $(\underline{a}', b' = \underline{a} \cdot \underline{s} + \Delta \cdot m + e') \leftarrow \mathsf{Sanitize}(\mathfrak{ct}, \mathsf{bsk}, \mathsf{ksk}, \mathsf{enc}_{\mathsf{id}}, \mathsf{pk})$.
2. Party $\mathcal{S}$ sends $(\underline{a}', b')$ to all parties $\mathcal{P}_i$ and $\mathcal{U}$.
3. Parties $\mathcal{P}_1, \ldots, \mathcal{P}_{\mathfrak{n}}$ execute $\langle d \rangle \leftarrow \sum_{i=1}^{w} \mathsf{PRSS.Next}()$.
4. Parties $\mathcal{P}_1, \ldots, \mathcal{P}_{\mathfrak{n}}$ compute $\langle p \rangle \leftarrow b' - \underline{a}' \cdot \langle \underline{s} \rangle + \langle d \rangle \pmod q$.
5. $p \leftarrow \mathsf{RobustOpen}(\langle p \rangle, \mathcal{U})$.
6. Party $\mathcal{U}$ extracts m from $p = \Delta \cdot m + e' + d$.
7. Party $\mathcal{U}$ returns m.

Figure 11: The Threshold Decryption Protocol

adversary) uniform random variables in the range $[-B, \ldots, B]$, whereas d_a is the sum of values known to the adversary. In total d consists of the sum of $w \cdot \binom{\mathfrak{n}}{\mathfrak{t}}$ uniform random variables in the range $[-B, \ldots, B]$.

Note that, unlike [DDK+23], we do not apply the PRSS masking operation twice in Figure 11, but we apply it w times. This is because we reduce security to our modified yaLWE computational assumption, as opposed to the statistical distance argument applies in [DDK+23]. The use of w applications means that the distribution we add on is the sum

of at least w, unknown to the adversary, uniform random variables in $[-B, \ldots, B]$.

In particular, since we would reasonably expect security for $(\mathsf{ya})^2\mathsf{LWE}$ when $\sqrt{2 \cdot \pi \cdot w \cdot B \cdot (B+1)/3} \approx \eta$, for the η parameter from the yaLWE problem, and this η is relatively small, the value of B can itself be very small. This should be contrasted with [DDK+23], where, due to the use of the statistical noise flooding argument, the value of B needs to be very large indeed. Also note that for fixed η we can increase w and hence decrease B so as to make the $(\mathsf{ya})^2\mathsf{LWE}$ problem become more like the yaLWE problem.

We can then prove the following theorem.

Theorem 4. *In the $\{\mathcal{F}_{\mathsf{KeyGen}}\}$-hybrid model the protocol in Figure 8 implements $\mathcal{F}_{\mathsf{KeyGenDec}}$ with computational security against any static* **active** *adversary corrupting I parties, with $|I| \leq \mathfrak{t} < \mathfrak{n}/3$ assuming:*

- *TFHE parameters that ensure correctness of the FHE operations.*
- *$|e'| + w \cdot \binom{\mathfrak{n}}{\mathfrak{t}} \cdot B < \Delta/2$ (to ensure correctness of the output of the "bath"-ing operation)[10].*
- *The noise parameter r of $\mathsf{Sanitize}$ satisfies the conditions of Theorem 1 and Lemma 7.*
- *The hardness of modified $(\mathsf{ya})^2\mathsf{LWE}^{\mathcal{S}}_{q,k\cdot N,\sigma,w,B}$, where $\sigma = r_{\mathsf{br}}$ from Theorem 2.*

Proof. Correctness follows, even in the presence of $\mathfrak{t} < \mathfrak{n}/3$ fully malicious parties, on noticing that, if sanitization does not create an invalid ciphertext, then, the correct value will be returned due to the robust nature of the $\mathsf{RobustOpen}$ procedure and the bound $|e'| + w \cdot \binom{\mathfrak{n}}{\mathfrak{t}} \cdot B < \Delta/2$.

Security of the protocol follows by showing that the output of simulator in Figure 12 is computationally indistinguishable, from the output of an adversary controlling I parties, with $|I| \leq \mathfrak{t} < \mathfrak{n}/3$, in a real execution of the protocol.

Note, the values $\{\langle p_r \rangle_j\}_{j \in I}$ produced by the simulator are the true decryption share values of the real pre-decryption value p_r produced by the adversary which the adversary should broadcast (even if he does not) if they acted honestly. Note that the simulated value p_s encodes the original message correctly. This means that the Lagrange interpolation in the simulation will recover the shares for the honest players $\{\langle p_s \rangle_j\}_{j \notin I}$ as required. By the properties of Shamir secret sharing the values $\{\langle p_r \rangle_j\}_{j \in I}$ are also valid shares of p_s, i.e. we can equate $\{\langle p_r \rangle_j\}_{j \in I}$ with $\{\langle p_s \rangle_j\}_{j \in I}$.

In a real execution of the protocol the shares output by the honest players are consistent and are enough to allow the honest parties to decrypt correctly, since $\mathfrak{t} < \mathfrak{n}/3$. The simulation has exactly the same properties.

The proof of security then is very much the same as in Theorem 3, except we replace the appeal to the hardness of yaLWE with an appeal to the hardness of $(\mathsf{ya})^2\mathsf{LWE}$. □

[10] This inequality could be rephrased in terms of σ, w, B, $\mathfrak{n}$, $\mathfrak{t}$ and a suitably chosen constant derived from the erfc function.

Simulator Threshold Decryption

On input of

1. A ciphertext $\mathfrak{ct} = (\underline{a}, b)$ and the public keys $\{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}\}$.
2. The underlying message m encrypted by $\mathfrak{ct}$.
3. A set of adversarial parties I with $|I| \leq \mathfrak{t}$.
4. The share values $\langle \underline{s} \rangle_i$ for $i \in I$.
5. The value σ, which is a publicly known value from the key generation method.

this algorithm outputs the simulated shares $\{\langle p \rangle_j\}_{j \notin I}$.

Sim-DistDecrypt:

1. $\underline{a}' \leftarrow \mathbb{Z}_q^{k \cdot N}$.
2. $b' \leftarrow \mathbb{Z}_q$.
3. The simulator computes d_a (which the adversary knows, and the simulator can also compute) and samples d_h according to the correct distribution (i.e. d_h is the sum of w uniformly random variables in $[-B, \ldots, B]$), and sets $d \leftarrow d_a + d_h$. As a by-product the simulator knows $\langle d \rangle_i$ for all $i \in I$.
4. The simulator computes, for $i \in I$, the share values $\langle p_r \rangle_i = b' - \underline{a}' \cdot \langle \underline{s} \rangle_i + \langle d \rangle_i$.
5. The simulator computes $p_s = \Delta \cdot m + \mathcal{D}_{\mathbb{Z},\sigma} + d$.
6. The simulator generates the decryption shares $\{\langle p_s \rangle_j\}_{j \notin I}$ via Lagrange interpolation (and possibly generating random shares if $|I| < \mathfrak{t}$) from p_s and the values $\{\langle p_r \rangle_i\}_{i \in I} = \{\langle p_s \rangle_i\}_{i \in I}$.
7. The simulator outputs $\{\langle p_s \rangle_j\}_{j \notin I}$.

Figure 12: Simulator for $\mathsf{ThreshDec}(\mathfrak{ct}, \langle \underline{s} \rangle, \{\mathsf{pk}, \mathsf{bsk}, \mathsf{ksk}\}, \mathcal{U})$

5.4 Parameters

To demonstrate that our approach indeed yields a threshold FHE scheme with small, TFHE-like parameters, we now provide a workable set of such parameters in Table 1 for the full threshold variant. These are adjusted from a typical TFHE parameter set that satisfies 132 bits of IND-CPA security and has failure probability $< 2^{-64}$. Our adjusted parameters provide around 124 bits of security and below we analyze the failure probability. Note that we do not claim optimality. In contrast to typical TFHE instantiations, which use a binary secret bootstrapping key, we use a secret bootstrapping key that is sampled from a bounded uniform(-like) distribution.

In typical applications, where we might use TFHE with a precision of 4 bits and a padding bit, we will have $\Delta = 2^{59}$. Thus to ensure correctness, i.e. that a ciphertext decrypts correctly, we need to ensure that any error term is less than 2^{58}.

Table 1: Parameters for Full Threshold Version

q	N	k	B_s	B_e	β	ℓ	$\log_2 r$	$\log_2 \sigma$	$\log_2 \eta$
2^{64}	2048	1	8	2048	14	879	31.08	55.05	46

Full Threshold Threshold Decryption: The final error term in this case is given by $e' + \sum_{i=1}^{\mathfrak{n}} d_i$ (cf. Theorem 3), and this term follows the distribution $\mathcal{D}_{\mathbb{Z},\sigma} + \sum_{i=1}^{\mathfrak{n}} \mathcal{D}_{\mathbb{Z},\eta} \approx_s \mathcal{D}_{\mathbb{Z},\sqrt{\sigma^2+\mathfrak{n}\cdot\eta^2}}$. So the term $\left|e' + \sum_{i=1}^{\mathfrak{n}} d_i\right|$ is bounded by $\tau \cdot \sqrt{\sigma^2 + \mathfrak{n} \cdot \eta^2}$ with probability $1 - \mathsf{erfc}(\sqrt{\pi} \cdot \tau)$. For our parameters in Table 1 and, say, $\mathfrak{n} = 1024$, we have $2^{58}/\sqrt{\sigma^2 + \mathfrak{n} \cdot \eta^2} \approx 13.7$ resulting in a failure probability of $< 2^{-858}$.

General Threshold Threshold Decryption: We now analyse the parameters for threshold FHE scheme with general threshold. We aim to set w and B such that $\sqrt{2 \cdot \pi \cdot w \cdot B \cdot (B+1)/3} = \eta$ (where η is as in Table 1) for security. Using the most aggressive version of our $(\mathsf{ya})^2\mathsf{LWE}$ assumption, we set $w = 1$ which gives $B \approx \sqrt{\frac{3}{2\cdot\pi}} \cdot \eta$. In order for the ciphertext to decrypt correctly, we need again $|e'| + w \cdot \binom{\mathfrak{n}}{\mathfrak{t}} \cdot B < 2^{58}$, or, equivalently, $|e'| < 2^{58} - \binom{\mathfrak{n}}{\mathfrak{t}} \cdot w \cdot B$. For example, using the same parameters as in Table 1 (except for dropping the use of η) we find with $w = 1$ that $B \approx 2^{45.47}$ and so we require $|e'| < 2^{57.4}$ if $\binom{\mathfrak{n}}{\mathfrak{t}} \leq 2048$. Thus, since e' follows the distribution $\mathcal{D}_{\mathbb{Z},\sigma}$, we will have such a bound with probability $1 - \mathsf{erfc}(\sqrt{\pi} \cdot 2^{57.4}/\sigma)$. Thus we have a failure probability of 2^{-374}.

6 FuncCPA

Our second application is to obtain funcCPA security, which is defined by the following security game.

Definition 5 (funcCPA Security, [AGHV22])**.** A PKE $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is funcCPA secure if for all PPT adversaries $\mathcal{A}$ it holds that

$$\left| 2 \cdot \Pr\left[b = b' \;\middle|\; \begin{array}{c} (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ b \leftarrow U(\{0,1\}) \\ b' \leftarrow \mathcal{A}^{O^b_{Enc_{\mathsf{pk}}}, O^{re}_{(\mathsf{pk},\mathsf{sk})}}(\mathsf{pk}) \end{array} \right] - 1 \right| = \mathsf{negl}(\lambda)$$

where $O^b_{Enc_s}(m_0, m_1)$ is the oracle that returns $\mathsf{Enc}_{\mathsf{pk}}(m_b)$ (if $|m_0| = |m_1|$) and may only be queried once, and $O^{re}_{(\mathsf{pk},\mathsf{sk})}(f, \mathfrak{ct})$ is the oracle returns $\mathsf{Enc}_{\mathsf{pk}}(f(\mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct})))$.

The work of [AGHV22] shows that sanitization allows to transform an IND-CPA secure scheme into a funcCPA secure one. For this it builds on the following definition of sanitization given in [DS16].

Definition 6. An algorithm $\mathsf{Sanitize}$ is sanitizing for a scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, if for all $(\mathfrak{ct}_1, \mathfrak{ct}_2) \in \mathcal{C}^2$ we have

- $\mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct}_1) = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Sanitize}(\mathfrak{ct}_1))$ (*message preservation*); and
- $\mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct}_1) = \mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct}_2)$ implies $(\mathsf{Sanitize}(\mathfrak{ct}_1), \mathsf{pk}, \mathsf{sk}) \approx_s (\mathsf{Sanitize}(\mathfrak{ct}_2), \mathsf{pk}, \mathsf{sk})$ (*sanitization*).

with overwhelming probability over the choice of $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$.

An FHE scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ can then be transformed into a funcCPA secure one by defining $\mathcal{E}' = (\mathsf{Gen}, \mathsf{Enc}', \mathsf{Dec}, \mathsf{Eval}')$ with $\mathsf{Enc}'_{\mathsf{pk}}(m) = \mathsf{Sanitize}(\mathsf{Enc}_{\mathsf{pk}}(m))$ and

$$\begin{aligned} &\mathsf{Eval}'(f, \mathfrak{ct}_1, \mathfrak{ct}_2, \ldots, \mathfrak{ct}_\ell) \\ &\qquad = \mathsf{Sanitize}(\mathsf{Eval}(f, \mathsf{Sanitize}(\mathfrak{ct}_1), \mathsf{Sanitize}(\mathfrak{ct}_2), \ldots, \mathsf{Sanitize}(\mathfrak{ct}_\ell)) \ . \end{aligned}$$

Lemma 9 ([AGHV22])**.** *If $\mathcal{E}$ is a* IND-CPA *secure FHE scheme with sanitization algorithm* $\mathsf{Sanitize}$ *then $\mathcal{E}'$ is* funcCPA *secure.*

The key idea is that re-encryptions can be simulated using Eval' and due to the properties of sanitization, this does not change the success probability of the adversary significantly. Note that Definition 6 requires statistical indistinguishability and accordingly Lemma 9 only holds with respect to statistical sanitization. If the sanitization property only holds computationally, we would need to argue security using a hybrid argument. The key difficulty here, as already observered in [AGHV22, DHW23], is that this requires one to simulate the re-encryption oracle without knowing the secret key and it is unclear how to do that. Statistical sanitization allows us to circumvent this issue, since we may switch out a (product) distribution for another and argue security using the data processing inequality.

We now present another definition of sanitization used in [BI22] (slightly generalized to the public key setting).

Definition 7. An algorithm $\mathsf{Sanitize}$ is *simulatably sanitizing* for a scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, if there exists a simulation algorithm Sim such that for all $\mathfrak{ct} \in \mathcal{C}$ we have

- $\mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct}) = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Sanitize}(\mathfrak{ct}))$ (*message preservation*); and
- $(\mathsf{Sim}(\mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct})), \mathsf{pk}, \mathsf{sk}) \approx (\mathsf{Sanitize}(\mathfrak{ct}), \mathsf{pk}, \mathsf{sk})$ (*simulatable sanitization*).

with overwhelming probability over the choice of $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$.

Clearly, this latter definition is stronger than the former, as the simulation only depends on the plaintext and so we have $\mathsf{Sanitize}(\mathfrak{ct}_1) \approx \mathsf{Sim}(m) \approx \mathsf{Sanitize}(\mathfrak{ct}_2)$ for $\mathfrak{ct}_1$, $\mathfrak{ct}_2$ encrypting the same plaintext m. But this stronger property allows us to replace $\mathsf{Sanitize}$ in Enc' by the simulation Sim, since the encryption algorithm gets the message as input.

Theorem 5. *If $\mathcal{E}$ is a* IND-CPA *secure FHE scheme with simulatable sanitization algorithm* $\mathsf{Sanitize}$ *then $\mathcal{E}''$, where* $\mathsf{Eval}'' = \mathsf{Eval}'$ *and* $\mathsf{Enc}''_{\mathsf{pk}}(m) = \mathsf{Sim}(m)$*, is* funcCPA *secure.*

Proof. The sanitization property of simulatable sanitization ensures that $\mathcal{E}'$ and $\mathcal{E}''$ are indistinguishable, so Lemma 9 implies that $\mathcal{E}''$ is funcCPA secure. □

Note that if simulation is much more efficient than sanitization, Theorem 5 results in a much more efficient encryption algorithm.

Secret-Key Encryption: We now instantiate Theorem 5 using a secret-key version of TFHE, where we rely on the efficiently simulatable sanitization algorithm $\mathsf{Sanitize}$. Note that in this setting, we may provide the simulation algorithm the secret key, since it is also input to the encryption algorithm.

- $\mathsf{KeyGen}(1^\lambda)$: This algorithm generates secret keys $\tilde{\underline{\mathbf{s}}} \in R_q^k$ and $\underline{s} \in \mathbb{Z}^\ell$ with binary coefficients, and then computes a corresponding bootstrapping bsk and key switching key ksk as in standard TFHE. It also computes GLWE samples $(\mathbf{A}, \underline{\mathbf{b}})$ with secret $\tilde{\underline{\mathbf{s}}}$ as required for Lemma 6 and appends them to bsk. It then computes r such that it satisfies the conditions of Theorem 1 and Lemma 6 and r_{br} from r, as in Theorem 2. Finally the algorithm publishes the evaluation key $(\mathsf{bsk}, \mathsf{ksk}, r)$ and returns $\mathsf{sk} = (\tilde{\underline{s}}, \underline{s}, r_{\mathsf{br}})$.
- $\mathsf{Enc}_{\mathsf{sk}}(m)$: Consider $\tilde{\underline{s}}$ as a flattened LWE key. Choose uniform $\underline{a} \in \mathbb{Z}_q^{kN}$ and $e \leftarrow \mathcal{D}_{\mathbb{Z}, r_{\mathsf{br}}}$ and return $(\underline{a}, \underline{a} \cdot \tilde{\underline{s}} + e + \Delta \cdot m)$.
- $\mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct})$: Perform $\mathfrak{ct}' \leftarrow \mathsf{KeySwitch}(\mathfrak{ct}, \mathsf{ksk})$ and $\mathfrak{ct}'' \leftarrow \mathsf{ModSwitch}(\mathfrak{ct}', \mathsf{ksk})$ and decrypt $\mathfrak{ct}''$ using $\underline{s}$.

Table 2: Parameters for funcCPA secure encryption. r_{ms} is the noise parameters from the mod switch.

q	N	k	B_e	β	ℓ	$\log_2 r$	$\log_2 \sigma$	$\log_2 r_{\mathrm{ms}}$
2^{64}	4096	1	4	1	879	16.9	38.6	54.9

The algorithm Enc statistically simulates Sanitize by Corollary 1 and thus Theorem 5 shows that this version of TFHE is funcCPA secure. It might seem odd at first sight that Dec performs the KeySwitch and ModSwitch operations before decryption. This is due to a technicality that has been overlooked in [AGHV22, DS16]: we need to ensure that Sanitize satisfies message preservation for maliciously chosen ciphertexts. Since Sanitize applies these two (deterministic) operations that incur some noise, we need to perform these as well during decryption to ensure that a sanitized ciphertext decrypts to the same message as the original ciphertext. We discuss this issue in more detail in Appendix A.

Public-Key Encryption: We now outline how to obtain a funcCPA public-key version of TFHE. The difference is in the encryption algorithm: since it does not receive the secret key as input, it is not immediately clear how to compute the LWE sample required to mask the (encoding of the) message $\Delta \cdot m$. Luckily, the solution is already at hand: simply use the statistical instantiation of the GLWE oracle (Lemma 6) to obtain a GLWE sample and perform Flatten to obtain an LWE sample. Then add the remaining part of the noise and $\Delta \cdot m$. Since Flatten is linear, this is equivalent to the encryption in the secret-key version.

Final Remarks: For the proof it is sufficient that there exists an (asymptotically) efficient Eval algorithm such that evaluation and functional re-encryption are indistinguishable. In practice, this algorithm need not be run at all, since client-aided outsourcing protocols typically rely on the client to provide the functional re-encryption "oracle". This is independent of other potential evaluation algorithms, for example using more efficient (not-sanitizing) bootstrapping parameters or for a restricted circuit family (e.g. linear functions). This has the interesting consequence that the optimization target for parameter optimization differs in this application of sanitization from other typical applications, like circuit privacy or threshold FHE. For funcCPA-security, we need not worry too much about the concrete efficiency of the sanitization, but rather about the simulation. In the case of our simulation of Sanitize, this means we can set parameters to minimize the output noise σ of the sanitization rather than maximize concrete performance, such that the encryption incurs a minimal amount of noise.

Parameters We provide parameters for the funcCPA secure encryption scheme presented above in Table 2. As is common when setting parameters for FHE, we heuristically assume that the combined noise from sanitization, mod switch and key switch follows a Gaussian distribution. We conservatively assume that the key switch noise is bounded by the mod switch noise. The parameters in Table 2 provide > 128 bits of security. Assume that the protocol performs up to n additions of independent ciphertexts in between re-encryption queries. Then the final noise can be estimated to be a Gaussian with noise parameter $\sqrt{n \cdot \sigma + 2 \cdot r_{\mathrm{ms}}}$. For example, if $n \leq 2^{30}$ and $\Delta = 2^{59}$ (for a message space of 4 bits), the resulting ciphertext will decrypt correctly except with probability $< 2^{-153}$.

References

[ABGS23] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, and Tjerand Silde. Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023: 30th Conference on Computer and Communications Security*, pages 1467–1481, Copenhagen, Denmark, November 26–30, 2023. ACM Press. doi:10.1145/3576915.3616683.

[ACD+19] Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan. Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 471–501, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-36030-6_19.

[AGHV22] Adi Akavia, Craig Gentry, Shai Halevi, and Margarita Vald. Achievable CCA2 relaxation for homomorphic encryption. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part II*, volume 13748 of *Lecture Notes in Computer Science*, pages 70–99, Chicago, IL, USA, November 7–10, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-22365-5_3.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-29011-4_29.

[BBB+23] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization and larger precision for (T)FHE. *Journal of Cryptology*, 36(3):28, July 2023. doi:10.1007/s00145-023-09463-5.

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th Annual ACM Symposium on Theory of Computing*, pages 52–61, San Diego, CA, USA, May 16–18, 1993. ACM Press. doi:10.1145/167088.167109.

[BCG+14] Johannes Buchmann, Daniel Cabarcas, Florian Göpfert, Andreas Hülsing, and Patrick Weiden. Discrete ziggurat: A time-memory trade-off for sampling from a Gaussian distribution over the integers. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 402–417, Burnaby, BC, Canada, August 14–16, 2014. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-43414-7_20.

[BD10] Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 201–218, Zurich, Switzerland, February 9–11, 2010. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-11799-2_13.

[BdPMW16] Florian Bourse, Rafaël del Pino, Michele Minelli, and Hoeteck Wee. FHE circuit privacy almost for free. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 62–89, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-53008-5_3.

[BGG+18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 565–596, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-96884-1_19.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325, Cambridge, MA, USA, January 8–10, 2012. Association for Computing Machinery. doi:10.1145/2090236.2090262.

[BI22] Florian Bourse and Malika Izabachène. Plug-and-play sanitization for TFHE. Cryptology ePrint Archive, Report 2022/1438, 2022. URL: https://eprint.iacr.org/2022/1438.

[BLR+18] Shi Bai, Tancrède Lepoint, Adeline Roux-Langlois, Amin Sakzad, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. *Journal of Cryptology*, 31(2):610–640, April 2018. doi:10.1007/s00145-017-9265-9.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-32009-5_50.

[BS23] Katharina Boudgoust and Peter Scholl. Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part I*, volume 14438 of *Lecture Notes in Computer Science*, pages 371–404, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore. doi:10.1007/978-981-99-8721-4_12.

[CdH10] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10: 7th International Conference on Security in Communication Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 182–199, Amalfi, Italy, September 13–15, 2010. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-15317-4_13.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume

3378 of *Lecture Notes in Computer Science*, pages 342–362, Cambridge, MA, USA, February 10–12, 2005. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-30576-7_19.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, Hanoi, Vietnam, December 4–8, 2016. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-53887-6_1.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020. doi:10.1007/s00145-019-09319-x.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *CSCML*, volume 12716 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2021.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland. doi:10.1007/978-3-319-70694-8_15.

[CLO+13] Ashish Choudhury, Jake Loftus, Emmanuela Orsini, Arpita Patra, and Nigel P. Smart. Between a rock and a hard place: Interpolating between MPC and FHE. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 221–240, Bengalore, India, December 1–5, 2013. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-42045-0_12.

[CSS+22] Siddhartha Chowdhury, Sayani Sinha, Animesh Singh, Shubham Mishra, Chandan Chaudhary, Sikhar Patranabis, Pratyay Mukherjee, Ayantika Chatterjee, and Debdeep Mukhopadhyay. Efficient threshold FHE with application to real-time systems. Cryptology ePrint Archive, Report 2022/1625, 2022. URL: https://eprint.iacr.org/2022/1625.

[DDK+23] Morten Dahl, Daniel Demmler, Sarah El Kazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P. Smart, Samuel Tap, and Michael Walter. Noah's ark: Efficient threshold-fhe using noise flooding. In Michael Brenner, Anamaria Costache, and Kurt Rohloff, editors, *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Copenhagen, Denmark, 26 November 2023*, pages 35–46. ACM, 2023.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-40041-4_3.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304, New York, NY, USA, March 4–7, 2006. Springer Berlin Heidelberg, Germany. doi:10.1007/11681878_15.

[DHW23] Yevgeniy Dodis, Shai Halevi, and Daniel Wichs. Security with functional re-encryption from CPA. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023: 21st Theory of Cryptography Conference, Part II*, volume 14370 of *Lecture Notes in Computer Science*, pages 279–305, Taipei, Taiwan, November 29 – December 2, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-48618-0_10.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-32009-5_38.

[DS16] Léo Ducas and Damien Stehlé. Sanitization of FHE ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 294–310, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-49890-3_12.

[Feh98] Serge Fehr. Span programs over rings and how to share a secret from a module, 1998. MSc Thesis, ETH Zurich.

[Fol15] János Folláth. Gaussian sampling in lattice based cryptography. *Tatra Mountains Mathematical Publications*, 60(1):1–23, 2015. URL: https://doi.org/10.2478/tmmp-2014-0022, doi:doi:10.2478/tmmp-2014-0022.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. URL: https://eprint.iacr.org/2012/144.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[GMPW20] Nicholas Genise, Daniele Micciancio, Chris Peikert, and Michael Walter. Improved discrete gaussian and subgaussian analysis for lattice cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12110 of *Lecture Notes in Computer Science*, pages 623–651, Edinburgh, UK, May 4–7, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-45374-9_21.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-40041-4_5.

[Joy22] Marc Joye. SoK: Fully homomorphic encryption over the [discretized] torus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):661–692, 2022. doi:10.46586/tches.v2022.i4.661-692.

[Joy24] Marc Joye. TFHE public-key encryption revisited. In Elisabeth Oswald, editor, *Topics in Cryptology – CT-RSA 2024*, volume 14643 of *Lecture Notes in Computer Science*, pages 277–291, San Francisco, CA, USA, May 6–9, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-58868-6_11.

[JSvL22] Robin Jadoul, Nigel P. Smart, and Barry van Leeuwen. MPC for Q_2 access structures over rings and fields. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021: 28th Annual International Workshop on Selected Areas in Cryptography*, volume 13203 of *Lecture Notes in Computer Science*, pages 131–151, Virtual Event, September 29 – October 1, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-030-99277-4_7.

[Kar16] Charles F. F. Karney. Sampling exactly from the normal distribution. *ACM Trans. Math. Softw.*, 42(1), January 2016. doi:10.1145/2710016.

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In David B. Shmoys, editor, *11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 937–941, San Francisco, CA, USA, January 9–11, 2000. ACM-SIAM.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-29011-4_41.

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th Annual Symposium on Foundations of Computer Science*, pages 372–381, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press. doi:10.1109/FOCS.2004.72.

[MS25] Daniele Micciancio and Adam Suhl. Simulation-secure threshold PKE from LWE with polynomial modulus. *IACR Communications in Cryptology*, 1(4), 2025. doi:10.62056/a0zogy4e-.

[MW17] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 455–485, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland. doi:10.1007/978-3-319-63715-0_16.

[NO07] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 343–360, Beijing, China, April 16–20, 2007. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-71677-8_23.

[PDG14] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. Cryptology ePrint Archive, Report 2014/254, 2014. URL: https://eprint.iacr.org/2014/254.

[Pei10] Chris Peikert. An efficient and parallel Gaussian sampler for lattices. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 80–97, Santa Barbara, CA, USA, August 15–19, 2010. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-14623-7_5.

[PS24] Alain Passelègue and Damien Stehlé. Low communication threshold fully homomorphic encryption. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part I*, volume 15484 of *Lecture Notes in Computer Science*, pages 297–329, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore. doi:10.1007/978-981-96-0875-1_10.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press. doi:10.1145/1060590.1060603.

[RST+22] Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for SPDZ. *Journal of Cryptology*, 35(1):5, January 2022. doi:10.1007/s00145-021-09416-w.

[sec09] secureSCM EU Project. Deliverable D9.2: Security Analysis, 2009. https://faui1-files.cs.fau.de/filepool/publications/octavian_securescm/SecureSCM-D.9.2.pdf.

[Wal19] Michael Walter. Sampling the integers with low relative error. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19: 11th International Conference on Cryptology in Africa*, volume 11627 of *Lecture Notes in Computer Science*, pages 157–180, Rabat, Morocco, July 9–11, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-23696-0_9.

A Definitional Issues of Sanitization

As hinted at in Section 6, there is a technical issue in the definition of sanitization. For convenience, we now reproduce the definition from Section 6 (which is the one given in [DS16]) to make the discussion easier to follow.

Definition 8. An algorithm $\mathsf{Sanitize}$ is sanitizing for a scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, if for all $(\mathfrak{ct}_1, \mathfrak{ct}_2) \in \mathcal{C}^2$ we have

- $\mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct}_1) = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Sanitize}(\mathfrak{ct}_1))$ (*message preservation*); and
- $\mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct}_1) = \mathsf{Dec}_{\mathsf{sk}}(\mathfrak{ct}_2)$ implies $(\mathsf{Sanitize}(\mathfrak{ct}_1), \mathsf{pk}, \mathsf{sk}) \approx_s (\mathsf{Sanitize}(\mathfrak{ct}_2), \mathsf{pk}, \mathsf{sk})$ (*sanitization*).

with overwhelming probability over the choice of $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$.

The main focus of this section is on the message preservation property. Note that it needs to hold over all ciphertexts, i.e. even maliciously chosen ciphertext. Lemma 3.2 of [DS16] claims that the generic sanitization algorithm based on bootstrapping (i.e. the *Refresh* operation) is message-preserving, which "follows from the definitions". This overlooks the

fact that bootstrapping is only required to be correct for honestly generated ciphertexts. From this it follows that the message preservation property (and thus sanitization) only holds for honestly generated ciphertexts. We stress that this does not invalidate the results of [DS16] with respect to their main application, namely honest-but-curious circuit privacy. In that setting, all ciphertexts are honestly generated and such a version of sanitization suffices. So one can view the definition of sanitization in [DS16] as too strong: it is neither required for the application nor achieved by the construction.

The paper [BI22] uses a simulatable sanitization definition similar to Definition 7, but where for each secret key, the set of ciphertexts associated to a message μ is defined explicitly and denoted by $\mathcal{C}_\mu$. Message preservation then requires that for each ciphertext in $\mathcal{C}_\mu$ the result of sanitization is also in $\mathcal{C}_\mu$. Initially, $\mathcal{C}_\mu$ is defined as the set of all ciphertexts that decrypt to μ. However, [BI22] observes that the ModSwitch operation in the bootstrap introduces noise, which could yield incorrect decryptions for maliciously generated ciphertexts and thus violate sanitization. They then re-define $\mathcal{C}_\mu$ to essentially be the set of ciphertexts that decrypt to μ *after a* ModSwitch, without explicitly re-defining decryption. One could argue that the adjustment of $\mathcal{C}_\mu$ implicitly re-defines decryption, which would mean they resolve the issue in a similar way as we do. Again, for their application of honest-but-curious privacy, this is irrelevant, so a relaxed definition of sanitization may have provided a simpler and cleaner solution in their case.

This is in contrast to the application of sanitization to funcCPA security, as in [AGHV22]. Here, sanitization is indeed required to hold over all ciphertexts in the ciphertext space. [AGHV22] seems to suggest that the work of [DS16] can be readily plugged in, but, as we saw, this is not necessarily the case. The work of [BI22] (with re-defined decryption) can be plugged into [AGHV22] to yield a funcCPA secure scheme (with costly encryption).