Hitchhiker's Guide to a Practical Automated TFHE Parameter Setup for Custom Applications

Jakub Klemsa

EURECOM, Sophia-Antipolis, France

jakub.klemsa@eurecom.fr

1 Introduction

Lately, several authors work on practical applications of FHE [3, 8, 10] using the TFHE scheme [5], often with an integer ring as the desired native cleartext space [4, 7, 9][1]. It shows that for the best performance of particular application, the TFHE parameters must be carefully selected with respect to the *bit-size of the cleartexts*, as well as to the *number of homomorphic additions* before bootstrapping is applied to refresh the noise. On top of that, the parameters must be derived with respect to a chosen *security level* λ. To the best of our knowledge, not a single existing TFHE library [6, 11, 12, 13] implements a configurable parameter derivation tool. Instead, they provide a few hard-coded parameter sets, which are only useful/effective in a limited range of applications.

In this contribution, we outline an approach, how to generate a set of tailor-made TFHE parameters, and we present a practical (semi-)automatic tool that executes this process, with particular respect to efficient resource utilization during the TFHE bootstrapping. Finally, we run our tool on several setups and we compare resulting parameters with those hard-coded in existing libraries. E.g., compared to the parameters by Zama in their $\mathbb{Z}/16\mathbb{Z}$ demo [14], we achieved by 38% faster bootstrapping time and by 57% smaller keying material, at a comparable security and noise level.

2 Towards TFHE Parameter Derivation

First, we formally introduce all input parameters, which reflect the needs of a particular application using TFHE. Next, we summarize the limitations that are imposed on the TFHE parameters. Finally, we outline an approach how to derive all the TFHE parameters, given the input parameters and respecting the identified limitations, with a focus on the bootstrapping efficiency.

Input Parameters

Given a specific application, which aims at utilizing TFHE for homomorphic calculations over encrypted data, we need to gather the following requirements: (A) bit-security level denoted by λ; (B) cleartext space bit-precision denoted by π; and (C) a parameterized bound on the number of homomorphic addition/scalar multiplication operations before the sample gets bootstrapped, denoted by $2^{2\Delta}$, referred to as the *quadratic weights*. Let us discuss each input parameter in detail.

[1]Also ongoing discussion within the community.

Security Level λ. To estimate the bit-security of particular instance of TLWE/TRLWE, which are the underlying ciphers of TFHE, we use the *LWE Estimator* by Albrecht et al. [2, 1].

Observation 1. *At given security level λ, the logarithm of the standard deviation of the* LWE *noise, denoted by α, depends roughly linearly on the* LWE *dimension n with a factor denoted by s_λ:*

$$-\log_2(\alpha) \approx s_\lambda \cdot n; \tag{1}$$

cf. Figure 1. Due to the collision attack, the relation is limited to $n \geq 2\lambda$. In addition, the precision of the underlying torus implementation, denoted by τ, changes the behavior for $-\log_2(\alpha) > \tau$.

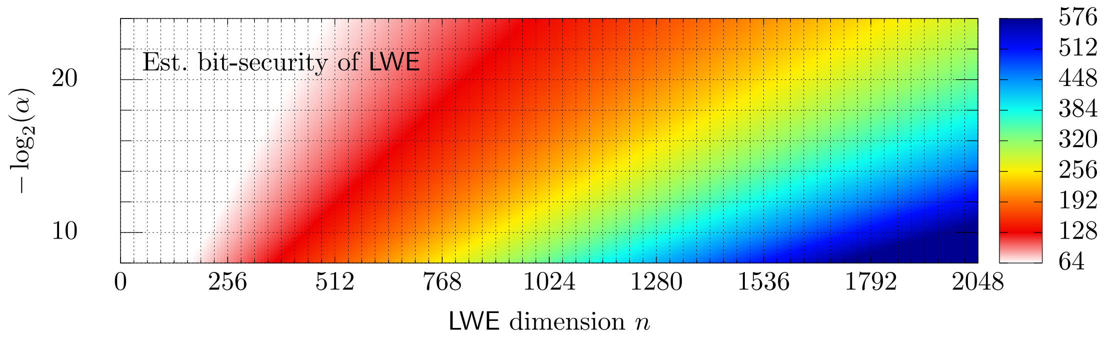

Figure 1: LWE bit-security λ estimated by the *LWE Estimator* by Albrecht et al. [2, 1]. Interpolation between grid points, not calculated for $\lambda < 64$ bits. For $\lambda = 128$ bits, $s_\lambda \approx 0.0235$. N.b., identified values of λ show to be lower than in [5, Fig. 9].

Cleartext Space Bit-Precision π. Calculations with encrypted data using TFHE are limited by the range of operations offered. First, it shows that the complexity of the TFHE bootstrapping grows roughly exponentially with the cleartext bit-precision – practical bootstrapping times can be achieved for up to about $\pi = 7$ bits only. Next, bootstrapping is on the one hand capable of evaluating a custom *Look-Up Table* (LUT), on the other hand, the values can only be given for the first half of the cleartexts – the rest is given implicitly by a negacyclic extension, i.e.,

$$LUT(2^{\pi-1} + m) = -LUT(m), \quad m \in [\![0, 2^{\pi-1}[\![. \tag{2}$$

Both limitations must be carefully considered before choosing the right cleartext precision π.

Quadratic Weights $2^{2\Delta}$. With each additive homomorphic operation, the noises add to each other. For the noise variance of a weighted sum of independent TLWE samples $\mathbf{c}_i$ with an equal noise variance, denoted by V_0, it holds

$$\mathsf{Var}\Big(\mathsf{Err}\Big(\sum w_i \cdot \mathbf{c}_i\Big)\Big) = \underbrace{\sum w_i^2}_{2^{2\Delta}} \cdot \underbrace{\mathsf{Var}(\mathsf{Err}(\mathbf{c}_i))}_{V_0}, \quad w_i \in \mathbb{Z}. \tag{3}$$

We denote the sum of squares of weights by $2^{2\Delta}$, where Δ expresses the number of additional bits of the standard deviation of the noise (due to the addition).

Parameter Restrictions

The overall goal is to limit the noise of a fresh(ly bootstrapped) sample, so that a limited number of addition operations can be performed and the noise can be refreshed correctly during bootstrapping. N.b., during bootstrapping, there occurs an additional rounding noise with a variance denoted by V_{round}. With 2^π values in the cleartext space, it follows that the maximum error shall be bounded by $1/2^{\pi+1}$. Using the rule of 3σ, it follows that the maximum error variance shall be bounded as

$$V_{\max} \leq 2^{2\Delta} V_0 + V_{round} \overset{!}{\leq} \frac{1}{3^2 \cdot 2^{2\pi+2}}, \qquad \text{where } V_{round} = \frac{n+1}{48N^2}. \tag{4}$$

The bound on V_0 depends on particular implementation, e.g., Bourse et al. [3] propose an improvement of one of the bootstrapping algorithms, which decreases the number of operations, but it increases the noise. Therefore, we present the bound on V_0 as it holds for the basic variant of TFHE [5], with one generalization that is widely implemented in TFHE libraries – the configurable base in the key switching algorithm:

$$\begin{aligned} 2^{2\Delta} V_0 \leq{} & \underbrace{2^{2\Delta} \cdot 2nlN 2^{2(\gamma-1)} V_{\mathsf{BK}}(N)}_{(\heartsuit)} + \underbrace{2^{2\Delta} \cdot n(1+N) 2^{-2(\gamma l+1)}}_{(\diamondsuit)} + \underbrace{2^{2\Delta} \cdot \mathsf{Var}\big(\mathsf{Err}(u,v)\big)}_{=0} + \\ & + \underbrace{2^{2\Delta} \cdot tN 2^{2(\kappa-1)} V_{\mathsf{KS}}(n)}_{(\clubsuit)} + \underbrace{2^{2\Delta} \cdot 2^{-2(\kappa t+1)} N}_{(\spadesuit)}, \end{aligned} \tag{5}$$

where N is the TRLWE polynomial degree, n is the TLWE dimension, γ is the base log of bootstrapping keys (per [5], $\gamma = \log_2(B_g)$), κ is the base log of key switching keys (not considered in [5], implemented in [6, 13]), l, t is the level of bootstrapping keys, or key switching keys, respectively, and V_{BK}, V_{KS} is the error variance used in bootstrapping, or key switching keys, respectively. Note that we assume that the bootstrapping LUT, represented by the sample (u, v), is public, hence it contains zero noise.

Idea of Parameter Generation

To derive a *good* set of TFHE parameters, we need to (i) satisfy the bound (4) (with the use of (5)), and (ii) check its quality in terms of the bootstrapping time.

For (i), the bound (4) can be viewed as an error budget, which gets consumed by individual error terms V_{round} and ($\heartsuit$)–($\spadesuit$). We suggest to proceed from the most restricted parameters: namely, we first set $n_{\min} = 2\lambda$, we loop $N \in \{256, 512, \ldots, 4096\}$ (or more), and we derive $n_{\max}$ from V_{round} using the entire budget. We continue by setting the minimum on κt from ($\spadesuit$) using the maximum available error budget, then we loop κ from 1 and t from $\lceil \kappa t_{\min}/\kappa \rceil$ until some fixed bound $t_{\max} \approx 10$ (note that t would indeed overflow due to ($\clubsuit$), however from some point, there is no efficiency gain). Next, we loop $n = n_{\min} \ldots n_{\max}$, possibly with a step greater than one[2]. Finally, we use the remaining error budget (if any) to obtain γ and l, analogically to κ and t. This way, we obtain thousands of parameter sets that satisfy the bound (4).

For (ii), to make a good choice of the parameters, we need some measure to compare them with each other. Either we know our implementation of TFHE by heart and we can deduce an analytical relation for the bootstrapping time, or we estimate the bootstrapping time using the sizes of the bootstrapping and the key switching keys, which can be expressed respectively as follows (for [5]):

$$\big|(\mathsf{BK}_i)_{i=1}^{n}\big| = 6nlN\tau \text{ [bits]}, \qquad \text{and} \qquad \big|(\mathsf{KS}_{i,j})_{i=1,j=1}^{N,t}\big| = (n+1)tN\tau \text{ [bits]}. \tag{6}$$

[2]E.g., in Concrete [6], the noise stddev can only be an integer, hence we can deduce the step from (1).

Running a couple of benchmarks with selected TFHE implementation, we derive a heuristic relation between key sizes and respective bootstrapping time.

The final selection can be made either completely automatically, or a human intervention may be applied, e.g., if there is a huge efficiency gap for just a little security reduction.

3 Experimental Results

Finally, in Table 1, we put forward experimental results for selected use cases: the binary TFHE [5], which uses an equivalent of 2-bit cleartext space, selected parallel addition [9, Alg. IIa-F], which employs 5-bit cleartexts, and two integer demos [14] by Zama, operating over $\mathbb{Z}/8\mathbb{Z}$ and $\mathbb{Z}/16\mathbb{Z}$ with 5-bit and 6-bit cleartext space, respectively. Note that in the parallel addition scenario, we originally used parameters with $\lambda = 112$, hence the parameters cannot be directly compared.

	Binary TFHE: $\pi = 2$, $2^{2\Delta} = 2$		Parallel addition: $\pi = 5$, $2^{2\Delta} = 20$	
	Orig. param's [13]	New param's	Orig. param's [9]	New param's
$N, n\ ;\ \gamma, l$	$1\,024, 630\ ;\ 7, 3$	$1\,024, 554\ ;\ 8, 2$	$1\,024, 680\ ;\ 7, 3$	$2\,048, 766\ ;\ 21, 1$
$\kappa, t\ ;\ \log(\alpha_{\mathsf{BK},\mathsf{KS}})$	$2, 8\ ;\ -25, -15$	$3, 3\ ;\ -24, -13$	$1, 16\ ;\ -29, -18$	$3, 5\ ;\ -48, -18$
$\lambda\ ;\ t_{BS}$	$127.1\ ;\ 81.4\,\mathrm{ms}$	$127.1\ ;\ 52.1\,\mathrm{ms}$	$111.5\ ;\ 103.0\,\mathrm{ms}$	$131.2\ ;\ 122.1\,\mathrm{ms}$
$\eta_C, \eta_m[\%]$	$16.1, 14.7$	$73.6, 72.5$	$78.3, 75.2$	$90.2, 86.9$

	$\mathbb{Z}/8\mathbb{Z}$ Demo: $\pi = 5$, $2^{2\Delta} = 2$		$\mathbb{Z}/16\mathbb{Z}$ Demo: $\pi = 6$, $2^{2\Delta} = 2$	
	Orig. param's [14]	New param's	Orig. param's [14]	New param's
$N, n\ ;\ \gamma, l$	$1\,024, 750\ ;\ 7, 3$	$1\,024, 724\ ;\ 6, 3$	$2\,048, 750\ ;\ 7, 3$	$2\,048, 766\ ;\ 21, 1$
$\kappa, t\ ;\ \log(\alpha_{\mathsf{BK},\mathsf{KS}})$	$2, 7\ ;\ -25, -18$	$2, 8\ ;\ -24, -17$	$2, 7\ ;\ -52, -18$	$3, 5\ ;\ -48, -18$
$\lambda\ ;\ t_{BS}$	$128.2\ ;\ 100.0\,\mathrm{ms}$	$130.5\ ;\ 98.9\,\mathrm{ms}$	$128.2\ ;\ 199.6\,\mathrm{ms}$	$131.2\ ;\ 124.6\,\mathrm{ms}$
$\eta_C, \eta_m[\%]$	$99.3, 101.4$	$100.8, 99.6$	$86.0, 85.5$	$91.2, 90.5$

Table 1: Comparison of selected original and newly identified TFHE parameters. 500 bootstraps with Concrete [6] on Intel Core i7-7800X were executed. η_C and η_m stand for the usage of the $3\sigma_{\max}$ error budget as calculated by Concrete and as measured after decryption, respectively. Find our experimental code at `https://gitlab.eurecom.fr/fakub/tfhe-param-testing`.

Discussion & Conclusion

We commented on and we also practically demonstrated the importance of careful TFHE parameter generation, given a particular usage scenario. We defined three input parameters that describe the scenario and we presented a tool that automates the parameter generation process. In the experimental results, we showed that for the binary TFHE and for the $\mathbb{Z}/16\mathbb{Z}$ demo, we can achieve by 36–38% faster bootstrapping times, only for the $\mathbb{Z}/8\mathbb{Z}$ demo, we did not achieve much improvement.

Acknowledgements

This work was supported by the MESRI-BMBF French-German joint project UPCARE (ANR-20-CYAL-0003-01).

References

[1] Martin R Albrecht, Benjamin R Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W Postlethwaite, Fernando Virdia, and Thomas Wunderer. LWE Estimator. `https://bitbucket.org/malb/lwe-estimator`, 2018.

[2] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[3] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, pages 483–512. Springer, 2018.

[4] Florian Bourse, Olivier Sanders, and Jacques Traoré. Improved secure integer comparison via homomorphic encryption. In *Cryptographers' Track at the RSA Conference*, pages 391–416. Springer, 2020.

[5] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[6] CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TfhE. `https://concrete.zama.ai/`, 2021.

[7] Antonio Guimarães, Edson Borin, and Diego F Aranha. Revisiting the functional bootstrap in tfhe. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 229–253, 2021.

[8] Malika Izabachène, Renaud Sirdey, and Martin Zuber. Practical fully homomorphic encryption for fully masked neural networks. In *International Conference on Cryptology and Network Security*, pages 24–36. Springer, 2019.

[9] Jakub Klemsa and Melek Önen. Parallel operations over tfhe-encrypted multi-digit integers. *Cryptology ePrint Archive*, 2022.

[10] Qian Lou and Lei Jiang. She: A fast and accurate deep neural network for encrypted data. *arXiv preprint arXiv:1906.00148*, 2019.

[11] NuCypher. A GPU implementation of fully homomorphic encryption on torus. `https://github.com/nucypher/nufhe`, 2022.

[12] Palisade. PALISADE Lattice Cryptography Library. `https://gitlab.com/palisade/palisade-release`, 2022.

[13] TFHE: Fast Fully Homomorphic Encryption Library over the Torus. `https://github.com/tfhe/tfhe`, 2016.

[14] Zama. Demo Z/8Z. `https://github.com/zama-ai/demo_z8z/`, 2021.