Demystifying Bootstrapping in Fully Homomorphic Encryption

Ahmad Al Badawi and Yuriy Polyakov

Duality Technologies

August 24, 2023

Abstract

Bootstrapping is a term used very often in the context of Fully Homomorphic Encryption (FHE). Anyone who is familiar with FHE knows that bootstrapping is the most sophisticated and compute-intensive component of an FHE scheme. However, very few non-FHE-experts understand what the bootstrapping operation really is and that there are various bootstrapping methods, each with its own tradeoffs. The goal of this paper is to provide a high-level introduction to common bootstrapping methods and evaluate their performance using the existing implementations in OpenFHE and HElib open-source libraries.

Our performance evaluation suggests that the bootstrapping in the Cheon-Kim-Kim-Song (CKKS) scheme provides highest throughput and efficiently achieves large precision for vectors of real numbers, which are often used in machine learning applications. The Ducas-Micciancio (DM) and Chillotti-Gama-Georgieva-Izabachene (CGGI) schemes achieve the smallest latency (typically for small integers or small-precision fixed-point numbers) and provide a general capability for evaluating arbitrary functions (programmable bootstrapping) via lookup tables. The Brakerski-Gentry-Vaikuntanathan (BGV) and Brakerski/Fan-Vercauteren (BFV) schemes provide higher bootstrapping throughput than DM/CGGI for vectors of small integers or finite-field elements but do not support programmable bootstrapping.

The target audience is anyone interested in FHE. We intend to keep this paper up-to-date to include new bootstrapping results as they become available.

1 Introduction

Bootstrapping is a term used very often in the context of Fully Homomorphic Encryption (FHE). Anyone who has read any introductory material on FHE already knows that bootstrapping is the most sophisticated and compute-intensive component of an FHE scheme. Very few, beyond cryptographers working in the field, understand what the bootstrapping operation really is and that there are various bootstrapping methods, each with its own tradeoffs. Our goals in this document are to demystify the concept of bootstrapping, correct misperceptions that seem to be common in the field, and provide a high-level comparison of bootstrapping methods available in common FHE schemes, so that a reader of this document can make well-informed decisions about choosing when and how to deploy FHE implementations.

We start with a brief introduction to homomorphic encryption, highlighting key concepts and historical milestones. Homomorphic encryption (HE) is a type of encryption that enables computations on encrypted data without having access to the secret key. Fully homomorphic encryption is

its most general form that can be used to evaluate arbitrary programs/computations on encrypted data. The idea of an FHE capability was first described by Rivest, Adleman, and Dertouzos back in 1978 (note that the first two were co-authors of the famous RSA scheme introduced around that time) [RAD78]. To describe homomorphic encryption, they used the term *"privacy homomorphisms"* and formulated the following problem: "it remains to be seen whether it is possible to have a privacy homomorphism with a large set of operations which is highly secure" [RAD78]. This problem could not be solved for 30 years, until at the end of 2008 Craig Gentry, a Ph.D. student at Stanford then, proposed the first FHE scheme [Gen09]. This was a monumental breakthrough not just in cryptography, but also in theoretical computer science; and *bootstrapping* was its key ingredient.

To introduce bootstrapping, we will try to answer three basic questions: 1) what is it? 2) why do we need it? and 3) how does it work?

What: A good starting point is to look at the definition of bootstrapping in the Oxford Dictionary: bootstrap is defined as *"pulling yourself up by your (own) bootstraps"*. When we say an HE scheme is bootstrappable, it means that it can homomorphically evaluate its own decryption procedure in addition to at least one extra operation [Gen09]. As shown in Fig. 1, evaluating the decryption procedure in the classical sense requires a ciphertext and secret key as input and ensures the plaintext as output. In FHE, however, we deal with a homomorphic evaluation of the decryption procedure, i.e., bootstrapping, which uses an encrypted secret key and a ciphertext to generate an "equivalent"[1] ciphertext that we can further compute on. The encrypted secret key, also called a bootstrapping or refreshing key, is provided by the secret key holder as part of the public key material.

Why: All common FHE schemes are based on noisy encryptions (the *noise* is what guarantees the security of fresh encryption) in which evaluating homomorphic operations increases the noise magnitude and lowers the quality, i.e., computational budget, of ciphertexts. The primary usage of bootstrapping is to convert an *exhausted* ciphertext into an "equivalent" *refreshed* ciphertext. Exhausted ciphertexts contain high noise and cannot be operated on further, whereas refreshed ciphertexts can support further homomorphic operations. A secondary purpose of bootstrapping is to evaluate a function on the encrypted message during the bootstrapping operation. In this case, the output ciphertext of bootstrapping encrypts a function of the plaintext message rather than the message itself, in addition to reducing the noise in the input ciphertext. This form of bootstrapping is known as *functional* or *programmable bootstrapping*.

How: All common bootstrapping methods follow the same blueprint introduced by Craig Gentry, that is, homomorphic evaluation of their own decryption procedure. However, the bootstrapping mechanisms vary across FHE schemes. In the following sections of this paper, we will describe the bootstrapping mechanisms in DM (FHEW) / CGGI (TFHE) [DM15, CGGI16, CGGI20], CKKS [CKKS17], and BGV/BFV [Bra12, BGV14, FV12] FHE schemes. We will highlight the

[1] Encrypts the same message but with smaller noise magnitude

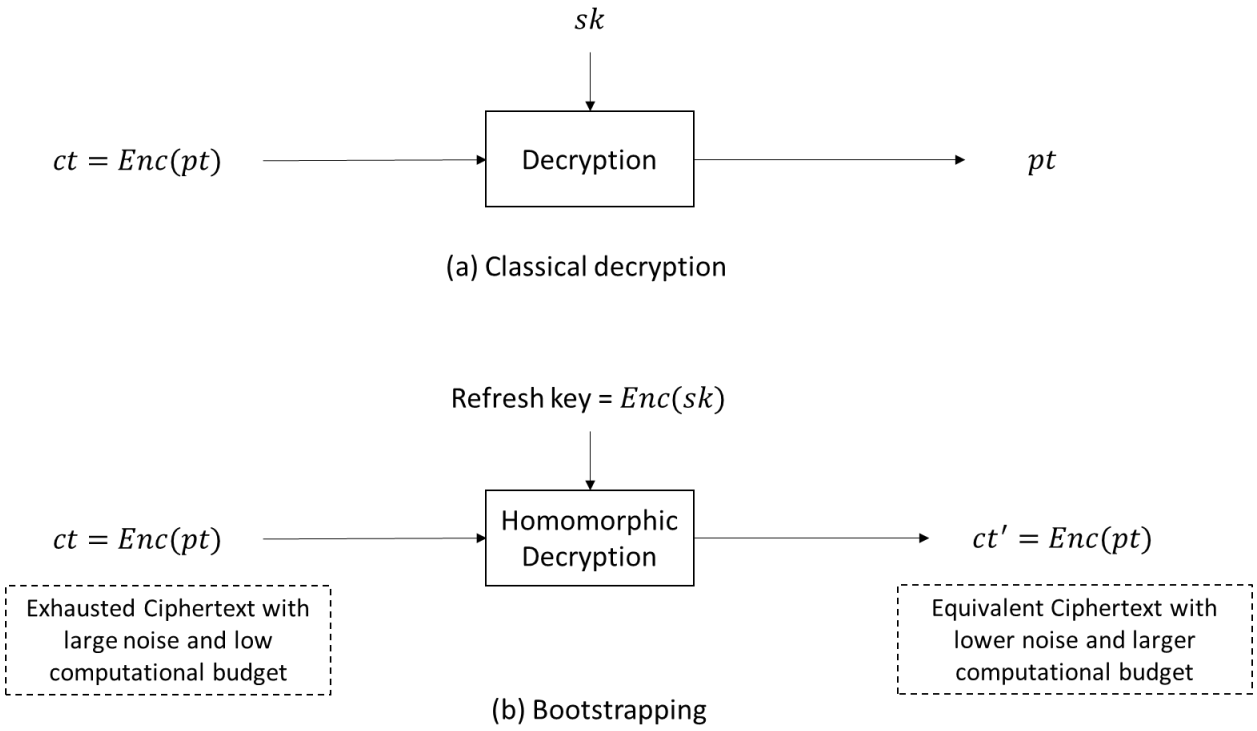

Figure 1: (a) Classical decryption vs (b) homomorphic decryption or bootstrapping. A ciphertext is generated as an encryption (Enc) of a certain plaintext message pt under a certain secret key sk. The refreshing or bootstrapping key is an encryption of the sk and is supposed to be made publicly available to the evaluator under the circular security assumption.

key differences between the methods, and evaluate their performance based on timing experiments using the OpenFHE library [ABBB$^+$22] and results reported in peer-reviewed scientific literature. Based on this analysis, we will provide some practical guidelines. Note that as of the time of writing this paper, OpenFHE includes bootstrapping for CKKS, DM, and CGGI schemes. Bootstrapping for BFV/BGV is currently under development and will be included in a future release of OpenFHE[2].

2 Bootstrapping in DM/CGGI cryptosystems

The Ducas-Micciancio (DM), also known as FHEW, cryptosystem[3] is notable for being the first FHE scheme to perform bootstrapping in a fraction of a second [DM15], which is dramatically faster than the bootstrapping implementation based on Gentry's original scheme, which required up to 30 minutes for a single bootstrapping operation [GH11]. The main motivation behind the DM cryptosystem was to evaluate an elementary bootstrapped computation (a Boolean NAND gate) with the smallest latency. Since NAND is a complete boolean gate, it can be used to implement arbitrary functions represented as Boolean circuits. Note that other Boolean gates can also be

[2]A prototype implementation of bootstrapping for BFV/BGV is already available

[3]We use the term cryptosystem here because DM and CGGI are based on multiple HE schemes

evaluated at the same computational cost [MP21], enabling more efficient Boolean circuits.

As mentioned earlier, bootstrapping is simply the homomorphic evaluation of a scheme's decryption function. The decryption procedure in HE schemes consists of computing a linear function (a product of the ciphertext and the encrypted secret key) and rounding. The DM cryptosystem uses a combination of schemes to evaluate these two operations. The linear function is evaluated via accumulation of the product of the ciphertext and the encrypted bits of the secret key using an intermediate cryptosystem (with small noise growth). Rounding is evaluated by extracting the most significant bit of linear function result via lookup table evaluation.

The Chillotti-Gama-Georgieva-Izabachene (CGGI) cryptosystem is a more memory-efficient variant of the DM cryptosystem. The authors used a different bootstrapping technique and introduced some additional optimizations to achieve a latency of less than 0.1 seconds. At a high level, the CGGI cryptosystem followed the same design as DM but relied on stronger security assumptions (binary secret distribution) and employed a more optimized accumulator to achieve faster bootstrapping. The CGGI cryptosystem was subsequently extended to the same security setting as DM [MP21], and the authors showed that CGGI is faster (and requires less memory) than DM for binary and ternary secrets, but DM achieves better performance for larger secret key distributions.

A unique feature the DM and CGGI cryptosystems share is the ability to evaluate arbitrary functions during the bootstrapping procedure. This can be done as part of the lookup evaluation procedure by replacing the plaintext bits with a function of them in the lookup table. This notion is known as functional or programmable bootstrapping [CJP21, LMP22]. Other FHE schemes do not enjoy this property; however, some special functions can still be evaluated during their bootstrapping procedure.

One major challenge the DM and CGGI cryptosystems face is enabling programmable bootstrapping for high-precision inputs. These schemes deal naturally with boolean data types (zeros and ones) and can be efficiently extended to support a slightly larger plaintext space (up to 3-8 bits for practical parameter sets). However, increasing the plaintext space by an additional bit after this threshold is reached requires doubling the runtime, which makes this approach impractical when higher precision is sought. Recent works show how to reduce this cost to linear (with precision) for some functions, but an evaluation of more general functions can still be computationally expensive [CLOT21, LMP22].

2.1 New bootstrapping methods

Lee at al. [LMK+23] proposed and implemented in OpenFHE a significantly improved variant of the DM scheme. The new variant requires smaller bootstrapping keys (smaller memory) than CGGI even for small secrets, incurs smaller accumulator noise than both DM and CGGI cryptosystems, and achieves runtimes that are same (for binary and ternary secrets) or faster (for larger secrets) than CGGI. This method is also a more efficient option for the threshold (multiparty) instantiation of DM-like schemes (since they work with larger secrets). Starting with v1.1, OpenFHE provides an efficient implementation of this method.

Xiang et al. [XZD+23] proposed and implemented in OpenFHE a cryptosystem based on the NTRU problem for the accumulator (rather than Learning With Errors used in DM and CGGI). They show that their method is more than 2x faster and memory-efficient than CGGI. As of v1.1, this method is not yet included in the official OpenFHE release.

3 Bootstrapping in CKKS

The Cheon-Kim-Kim-Song (CKKS) scheme is the most recent FHE scheme that is optimized for floating-point computations [CKKS17]. It provides the best efficiency for machine learning applications, such as logistic regression training or neural network inference. In contrast to all other common FHE schemes, CKKS is approximate and often uses polynomial approximations to implement nonlinear functions, e.g., logistic or ReLU functions. As CKKS is approximate, the bootstrapping for CKKS is also approximate, i.e., the encrypted message in the refreshed ciphertext is close to the message before bootstrapping, but not equal to it.

The high-level idea of bootstrapping in CKKS follows Genry's blueprint and the decryption procedure includes a linear function and modular reduction (equivalent conceptually to rounding) [CHK+18]. First, the linear function (inner product of ciphertext and secret key) is implicitly evaluated by increasing the ciphertext modulus (a larger modulus means more computations can be done homomorphically). This step adds some "garbage" to the message that is proportional to the ciphertext modulus before bootstrapping. Then modular reduction is performed to remove this garbage. The modular reduction is evaluated using its polynomial approximation (a high-degree polynomial interpolation is needed) and is the most expensive step of the bootstrapping procedure. Some additional steps are performed but they are omitted here for simplicity.

One important difference between CKKS and DM/CGGI is related to how data is encoded in ciphertexts. In DM/CGGI, one ciphertext stores one encrypted scalar. In CKKS, one ciphertext stores a vector of real numbers, and the size of this vector is on the order of thousands. This vector encoding is similar to the Single-Instruction-Multiple-Data (SIMD) concept in parallel processing. Using a single ciphertext multiplication, one can perform thousands of real-number multiplications at once. The same difference equally applies to the bootstrapping procedures. In DM/CGGI, we can bootstrap only one number at a time while in CKKS we can bootstrap thousands of numbers (typically 32K - 64K in bootstrapping scenarios) at once. The bootstrapping latency in CKKS is typically much larger than in DM/CGGI, but the throughput of bootstrapping is much higher due to the SIMD effect, as we will illustrate later in this paper.

The second difference is related to the precision supported by CKKS and DM/CGGI schemes. CKKS efficiently supports much higher precision, e.g., single (23-bit) or even double (52-bit) floating-point precision, while DM/CGGI efficiently supports only a smaller precision (3-8 bits). Extensions to larger precision for DM/CGGI are efficient only for special nonlinear functions, e.g., comparison [LMP22].

The third difference between DM/CGGI and CKKS is related to programmable bootstrapping. CKKS does not natively support programmable bootstrapping. However, CKKS provides an efficient polynomial evaluation mechanism to approximate nonlinear functions, which works well

for relatively smooth functions. Hence, the equivalent functionality in CKKS can be achieved by evaluating polynomial approximation and then executing the bootstrapping procedure.

We will keep all of these differences in mind when comparing the performance of programmable bootstrapping between DM/CGGI and CKKS.

Although CKKS bootstrapping is approximate, a recent elegant technique can be used to make the CKKS bootstrapping error negligible by performing multiple invocations of bootstrapping: first, on the message, next, on the first-order bootstrapping error, then, on the second-order bootstrapping error, etc., as described in [BCC+22].

4 Bootstrapping in BFV/BGV schemes

Brakerski-Gentry-Vaikuntanathan (BGV), Brakerski/Fan-Vercauteren (BFV), and CKKS schemes are usually bundled together as one family of SIMD-enabled FHE schemes [ABBB+22]. While CKKS is inherently suitable for encrypted approximate computations that deal with floating-point numbers, such as those in machine learning, BFV and BGV work well for encrypted exact computations that deal with integer data types, such as database and string manipulating applications. All three schemes share a very similar outline and follow the same strategy for bootstrapping.

The main difference between BGV/BFV and CKKS bootstrapping is related to the procedure for garbage addition and removal. In BGV/BFV, the garbage gets added to a temporarily increased plaintext space, and then so-called digit extraction polynomials are used to clear out the garbage from new digits of the plaintext space, appropriately scaling the message down along the way [HS21]. The main bottleneck in BGV/BFV bootstrapping is digit extraction as it requires homomorphic evaluation of high-degree polynomials.

The use of a special larger plaintext space puts additional constraints on the plaintext algebra, which in practice restricts the size of SIMD-encoded vectors to 2K or less. For this reason, BGV/BFV bootstrapping achieves a significantly smaller throughput than CKKS as the latter supports vectors of size 32K - 64K.

Note that BGV/BFV bootstrapping supports two additional operations that can be performed as part of bootstrapping: scaling down and sign evaluation (comparison) [CGBH+18]. So there is limited programmable bootstrapping support, but it is not as general as in the case of DM/CGGI.

5 Experimental setup

We present the performance evaluation of bootstrapping for CGGI, CKKS, and BGV schemes. We chose CGGI over DM because the former is slightly faster than DM for the typical security setting of uniform ternary secret distribution. The complexity of BGV and BFV bootstrapping is very similar, so it is sufficient to consider only BGV here. We used the CGGI and CKKS implementations in OpenFHE to run our experiments. For BGV, we used reported results from [HS21]. Security parameters were chosen to correspond to the 128-bit security level.

We ran our benchmarks for CKKS and CGGI on a commodity desktop computer system with an Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz and 64 GB of RAM, running Ubuntu 20.04.5 LTS. The compiler used was clang version 12.0.0-3ubuntu1. We compiled OpenFHE with the following CMake flags: NATIVE_SIZE=32 for CGGI with single-bit plaintext space, NATIVE_SIZE=64 for CGGI with multi-bit plaintext space and CKKS, and WITH_NATIVEOPT=ON (machine-specific optimizations were applied by the compiler). We report only the single-threaded time (by setting the environment variable OMP_NUM_THREADS=1 or turning OpenMP off using WITH_OPENMP=OFF; both options produce roughly the same benchmarking results); no parallelization due to multi-threading or AVX-512 optimization was used. Note that both schemes in OpenFHE use the same underlying mathematical library implementation, which helps us avoid discrepancies due to various implementation specifics and gives us an "apple-to-apple" comparison.

5.1 Performance evaluation

To provide a fair comparison of CGGI with SIMD-capable CKKS and BGV, we include the *number of slots* (size of vector encoded in a single ciphertext), bits or precision, latency, and throughput. Moreover, for CKKS and BGV we reserve a reasonable number of multiplicative levels (10 or so) to support an evaluation of a nonlinear function between the invocations of bootstrapping, thus emulating programmable bootstrapping in CGGI.

Table 1 shows that CGGI has the lowest bootstrapping latency. CGGI bootstrapping is the most efficient option when the number of slots is small and precision is low. However, as the number of slots is increased, CKKS bootstrapping becomes the most efficient option (higher throughput and higher precision are achieved). BGV bootstrapping has throughput similar to CGGI for Boolean arithmetic, but is more efficient than CGGI for larger plaintext spaces, i.e., higher precision.

It is important to note that CKKS does not require bootstrapping for many simpler operations, such as multiplication. The multiplication only consumes one level and can be done for thousands of real numbers at a time. On the other hand, integer multiplication can be expensive in CGGI (may require additional bootstrapping calls), especially if the multiplication result should not wrap around the current plaintext modulus. In summary, the number of bootstrapping invocations in CKKS is typically significantly smaller than in CGGI for many practical use cases.

Notes on parallelization and hardware acceleration. We intentionally chose the single-threaded CPU setup to provide a fair comparison of the bootstrapping methods using the same implementation. The runtimes achieved in practice can be much smaller (even by orders of magnitude) by either using multithreading on a multi-core/socket CPU system or hardware acceleration. Both of these performance improvements can be applied to all bootstrapping methods considered here. For instance, cuFHE achieved a 26x speed-up for CGGI by executing many gates in parallel on a GPU [DS15] and [JKA+21] achieved a speed-up of 100x for CKKS bootstrapping. The discussion of accelerated implementations of bootstrapping methods is beyond the scope of our introductory paper.

Table 1: Performance of bootstrapping in CGGI (single- and multi-bit plaintext space), CKKS (fully- and sparsely-packed ciphertexts), and BGV schemes[5]. CGGI and CKKS experiments were run in OpenFHE v1.1.1; BGV experiments were obtained using HElib; all experiments were performed in the single-threaded mode and did not use AVX-512 acceleration

Scheme	Library	Number of Slots	Bits of Precision	Latency (sec)	Throughput slots/sec	Notes
CGGI TFHE	OpenFHE	1	1	0.0295	33.9	Latency of NAND
			3-4	0.450	2.22	Latency of f(x) mod p
CKKS	OpenFHE	32,768	13	44.0	745	Full packing
			28	92.4	355	2 bootstrapping iterations to increase precision [BCC+22]
CKKS	OpenFHE	32	18	21.5	1.49	Sparse packing
			36	45.3	0.71	2 bootstrapping iterations to increase precision [BCC+22]
BGV	HElib	1,024	1	15	68.3	Thin bootstrapping [HS21]
			16	163	6.28	Full bootstrapping [HS21]

6 Recommendations and concluding remarks

Our goal was to explain various bootstrapping methods and provide some guidelines that can be used in practice. Our main results can be summarized as follows:

- CKKS bootstrapping has the best performance when the ciphertext contains a large number of slots (more than 100) and/or higher precision is sought (more than 3-8 bits).
- DM/CGGI bootstrapping is more efficient when the number of slots is small (up to 100) and the desired precision is low (up to 3-8 bits).
- DM/CGGI bootstrapping can be used to efficiently evaluate arbitrary functions (over small integers) using lookup tables, whereas CKKS can evaluate relatively smooth functions (over real numbers) that can be adequately approximated using polynomials, e.g., Chebyshev interpolation.
- BGV/BFV bootstrapping is somewhat faster than DM/CGGI (if we consider slot-amortized time), but slower than CKKS. However, BGV bootstrapping does not natively support arbitrary function evaluation.

[5]Specific OpenFHE benchmarks or examples used to generate these figures:
CGGI benchmark (1-bit): benchmark/src/binfhe-ginx.cpp
CGGI benchmark (3- or 4-bit): src/binfhe/examples/eval-function.cpp; 3 bits of precision ($p = 2^3$) for arbitrary functions and 4 bits ($p = 2^4$) for periodic functions [LMP22]
CKKS full packed benchmark: src/pke/examples/simple-ckks-bootstrapping.cpp
CKKS sparse benchmark: src/pke/examples/advanced-ckks-bootstrapping.cpp
CKKS multi-iteration bootstrapping: src/pke/examples/iterative-ckks-bootstrapping.cpp

- Hardware acceleration can be applied to all these bootstrapping methods, and the available literature suggests the expected speed-up should be similar for all these methods.

Many applications may require both large problem sizes (where CKKS works well) and arbitrary function evaluation (where CGGI performs well), e.g., decision tree training. In this case, scheme switching from CKKS to CGGI and back may be beneficial. The scheme switching between CKKS and CGGI is a new feature of OpenFHE that was added in v1.1.

To have a further discussion on this paper, go to the OpenFHE Discourse forum.

References

[ABBB+22] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. In *Proceedings of the 10th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'22, page 53–63, New York, NY, USA, 2022. Association for Computing Machinery.

[BCC+22] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. META-BTS: Bootstrapping precision beyond the limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 223–234, 2022.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 868–886. Springer, 2012.

[CGBH+18] H. Chen, R. Gilad-Bachrach, K. Han, Z. Huang, A. Jalali, K. Laine, and K. Lauter. Logistic regression over encrypted data from fully homomorphic encryption. *BMC Med Genomics*, 11(Suppl 4):81, Oct 2018.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*, pages 3–33. Springer, 2016.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *J. Cryptol.*, 33(1):34–91, 2020.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 360–384, Cham, 2018. Springer International Publishing.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann, editors, *Cyber Security Cryptography and Machine Learning*, pages 1–19, Cham, 2021. Springer International Publishing.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[CLOT21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 670–699, Cham, 2021. Springer International Publishing.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*, pages 617–640. Springer, 2015.

[DS15] Wei Dai and Berk Sunar. cuhe: A homomorphic encryption accelerator library. In *Cryptography and Information Security in the Balkans: Second International Conference, BalkanCryptSec 2015, Koper, Slovenia, September 3-4, 2015, Revised Selected Papers 2*, pages 169–186. Springer, 2015.

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

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[GH11] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology–EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings 30*, pages 129–148. Springer, 2011.

[HS21] Shai Halevi and Victor Shoup. Bootstrapping for HElib. *Journal of Cryptology*, 34(1):7, 2021.

[JKA+21] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 114–148, 2021.

[LMK+23] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 227–256, Cham, 2023. Springer Nature Switzerland.

[LMP22] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II*, pages 130–160. Springer, 2022.

[MP21] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like cryptosystems. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '21, page 17–28, New York, NY, USA, 2021. Association for Computing Machinery.

[RAD78] Ronald L Rivest, Len Adleman, and Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[XZD+23] Binwu Xiang, Jiang Zhang, Yi Deng, Yiran Dai, and Dengguo Feng. Fast blind rotation for bootstrapping fhes. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 3–36, Cham, 2023. Springer Nature Switzerland.