KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Fastest Homomorphic Encryption in the West Made Faster

Jonas Bertels

Thesis submitted for the degree of
Master of Science in
Electrical Engineering, option
Electronics and Chip Design

Thesis supervisor:
Prof. dr. ir. Ingrid Verbauwhede

Mentors:
Ir. Michiel Van Beirendonck
Ir. Furkan Turan
Ir. Jose Maria Bermudo Mera
Ir. Jose Maria Bermudo Mera

Academic year 2020 – 2021

Preface

Thanks and acknowledgement

Jonas Bertels

Contents

Abstract

The `abstract` environment contains a more extensive overview of the work. But it should be limited to one page.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

LoG	Laplacian-of-Gaussian
MSE	Mean Square error
PSNR	Peak Signal-to-Noise ratio

Symbols

42	"The Answer to the Ultimate Question of Life, the Universe, and Everything" according to [1]
c	Speed of light
E	Energy
m	Mass
π	The number pi

Chapter 1

Introduction to homomorphic encryption

Homomorphic encryption means encryption on which functions can still be executed. In 2009 Gentry [7] showed that Fully Homomorphic encryption, i.e. doing an arbitrary amount of operations on encrypted data, is possible.

1.1 An Analogy

Gentry explained this using an analogy: imagine a jeweler, Alice, who wants to let her workers work on precious gems without them being able to steal the gems. She creates a box that allows the workers to manipulate the jewels. This box represents Somewhat Homomorphic Encryption, because while the workers can do some operations, eventually the gloves stop working (so that the workers can no longer continue the work, although the jewels are still safe). Thus, the workers have to bring the box back to Alice so that she can unlock the box and retrieve the finished jewels. If the workers haven't finished their work yet, she must take the half-finished jewels from the box with the "worn" gloves and place them in a box with new gloves so that the workers can continue their work.[2]

The "box" in this analogy is of course our encryption scheme. "Alice" is the user while the workers represent a server that can do operations (such as addition, multiplication or others) on the encrypted data without being able to access it. However, because the "box" in our case is the encryption scheme "Learning With Errors", every operation that is done will increase the amount of error added to the final result. In the beginning, this error is small, meaning that we can still recover our result simply by rounding. At some point, the number of operations will cause the noise to exceed the threshold at which we can remove it. At that point, our "worker" (i.e. the server which is doing the operations) has to stop or the output it gives will be wrong.

The idea behind bootstrapping is to turn Somewhat Homomorphic Encryption into Fully homomorphic encryption. This can be done by performing the decryption, i.e. the unlocking of the box, while everything is encrypted under another encryption. In our analogy, this would be like locking our box in another box that already has a key for our box inside of it. (Since we are using public/private key encryption, we can place things into a locked box with the public key without having to unlock said box. This is a feature of public key encryption).

Then our workers can open the inner box and continue working on the jewels. For an arbitrary amount of boxes, we can thus compute an arbitrary amount of functions on data that stay encrypted. The fact that gloves wear out in our analogy is representative of the fact that the homomorphic encryption schemes use some form of numerical noise to mask the message. As we do addition, or multiplication, the amount of noise increases. Multiplication increases the noise at a much greater rate than addition. Due to this noise increase, decryption of the data (whose last step is usually a sort of rounding operation to remove noise) will at some point no longer return the correct answer.

Chapter 2

Learning With Errors

We can split the homomorphic schemes that are currently the focus of most research into 2 (or 3) generations. The first generation consists of older schemes that are so inefficient they are no longer being used. The second generation consists of schemes like the Fan-Vercauteren [6] scheme, which is primarily used for Somewhat Homomorphic Encryption. The second-generation schemes allow for multiplication and addition but have a very compute-intensive bootstrapping process. The third generation schemes only allow for 1 gate (such as a AND/OR gate), and bootstrap immediately, but the bootstrapping happens relatively fast (100ms)[9]. In this master thesis, I have accelerated one of the third generation schemes, namely Fastest Homomorphic Encryption in the West (FHEW) but it is still worth to first consider the Fan Vercautren scheme.

2.1 Second Generation Schemes

First comes the introduction to this topic. The Fan-Vercauteren scheme [6] is based on the Ring Learning With Errors Problem, which states the following: For many vectors $(a_i, b_i) \in (R_q, R_q), b_i = a_i * s + e_i$ with e_i being a randomly sampled error term and s being our secret key and a_i being a randomly sampled but known vector, it is not possible to find s (" computationally unfeasible" (Roy et al.) [13]) In other words, (a_i, b_i) is a good public key for since we can't reverse engineer the secret from the public key. If we consider the problem from the case of the server: we cannot reverse engineer (a_i, b_i) into s, even if we have a very large quantity of a_i's and b_i's.

Consider the following simplified Fan Vercauteren scheme: (equations showing addition and multiplication are still valid).

Our public key is:

$$b = (-a * s - e) \bmod q \tag{2.1}$$

and

$$a \text{ (randomly sampled)} \tag{2.2}$$

which we put together to form (a, b). Our cipher text is then:

$$c = (((-a * s - e) \bmod q) * u + e_1 + m, a * u + e_2) = (c_0, c_1) \tag{2.3}$$

with u randomly sampled and e, e_0 and e_1 randomly sampled error terms.
To decrypt , we then simply multiply the second part of the ciphertext c_1 with the secret key s and add this to the first part c_0.

$$m = \lfloor c_0 + c_1 * s \rceil \tag{2.4}$$

Adding or multiplying ciphertexts together will increase the noise but will (for low enough noise values) still return the correct answer when decrypted.

2.1.1 Fan-Vercauteren: usually Somewhat Homomorphic Encryption

FV has been the subject of hardware acceleration by multiple groups [10] [14] [13]. It is also implemented in the homomorphic encryption libraries SEAL and PALISADE among others.
The problem with the second generations schemes as previously mentioned is that they usually forgo the bootstrapping for practical reasons. This means that any user of the libraries has a limited number of multiplications to work with (as we have seen, the limited noise growth of additions is usually negligible). This imposes constraints on the programmer.

In summary then: second-generation schemes execute fairly quickly, and as such schemes like FV are good for Single Instruction, Multiple Data-type (SIMD) programs. However, they only allow for a limited numberof computations, and algorithms that are already out there and require a lot of instructions will thus not easily translate into programs that can homomorphically execute data. This can also be seen in a hardware acceleration for the Fan-Vercauteren scheme done by COSIC in 2019 [13]. This implementation could do FV very quickly, at 400 homomorphic multiplications per second, but is limited to a multiplicative depth of 4. Although this is enough for a lot of applications, it can limit its flexibility in a general role.

2.2 FHEW (Fastest Homomorphic Encryption in the West)

FHEW (the name is a reference to the Fastest Fourier Transform in the West library) is part of the so-called third-generation schemes[4] [3]. It attempts to tackle the long length of the bootstrapping process by only executing one NAND gate (other simple gate functions are also possible) and then immediately bootstrapping. This makes it a true fully homomorphic scheme, as the bootstrapping can actually be done within a reasonable amount of time, namely 137 milliseconds for NAND + bootstrap for

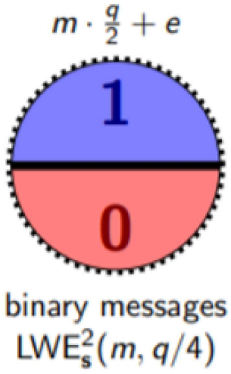

FIGURE 2.1: Binary Message representation.

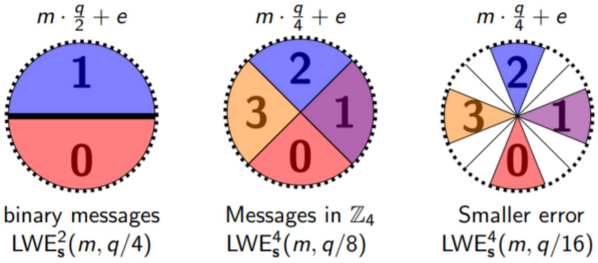

FIGURE 2.2: Binary Message representation.

128-bit security on an intel i7 [9]. In other words, while only one NAND gate can be done at a time (later papers show parallelisation is possible), there is no limit on the depth of our circuit, and as all functions can be written as a combination of logic gates, no limit on the functions that can be executed.

2.3 Figures

FHEW is also based on RLWE. The figures show how we use RLWE to represent bits. ([5]). Our message is a single bit. If it is 0, we could represent it as some error, if it is 1, we could represent it as 256 with some error added to it. Figure 2.1 (if 512 is our modulus). In this case, our maximum error is equal to +/- 128. The representation that we use is one where there are 4 possible messages, with the message being multiplied by 128, or ¼ of the modulus q (Figure 2.2). We also have a smaller error of value 64 or $q/8$, so that there are values that are never reached by a given message and error. (these white spaces).

We can use this for addition (Figure 2.3). As you can see, the result now has more error than the beginning (Figure 2.4). Now if we consider everything on the

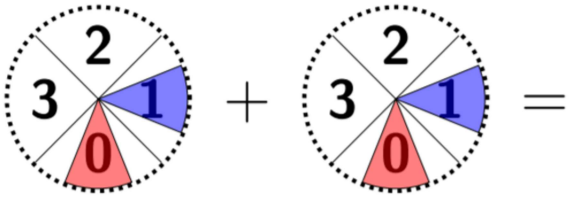

Figure 2.3: Binary Message representation.

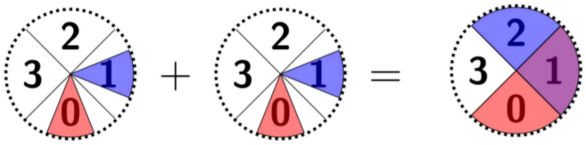

Figure 2.4: Binary Message representation.

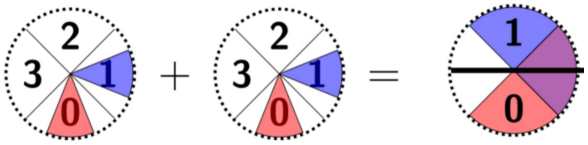

Figure 2.5: Binary Message representation.

top left to be 1 and everything on the bottom right to be 0, we have a AND gate (Figure 2.5). To make this a NAND, we rotate by 5q/8, in other words, we consider the bottom right to be 1 and the other 0 and this results in the results that we expect from a NAND gate for given input values (Figure 2.6).

Then we need to bootstrap. Bootstrapping is doing decryption operations homomorphically, i.e. doing the decryption operations with an encrypted secret key. This reduces the noise back down to the original level, completing our algorithm. For most hardware designers, the most important part of any algorithm is the size of the parameters that we work with, so as such we will give them here. The version that I

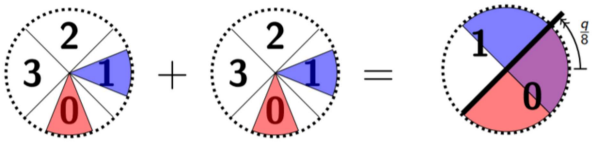

Figure 2.6: Binary Message representation.

accelerated is an optimized version of the FHEW scheme originally introduced in 2015. This was chosen because it was the simplest of all third-generation schemes. The only optimisation paper yet was an effort to make FHEW run on multicore CPU and GPU's, where they achieved a speedup of about 2.2 using CUDA on the 2015 version of FHEW. [8]

Another optimisation paper introduced NuFHE, which uses another third-generation scheme called TFHE with GPU acceleration and succeeded in a 100 times speedup, bringing the cost of homomorphic encryption down to 0.13 ms/bit for binary gates (for FFT implementation). For NNT implementations it is 0.35 ms/bit [11].

Bibliography

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy*. Del Rey (reprint), 1995. ISBN-13: 978-0345391803.

[2] C. Bonte. Co6gc: Homomorphic encryption (part 1): Computing with secrets. URL: https://www.esat.kuleuven.be/cosic/blog/co6gc-homomorphic-encryption-part-1-computing-with-secrets/, last checked on 2020-03-25.

[3] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Report 2018/421, 2018. https://eprint.iacr.org/2018/421.

[4] L. Ducas and D. Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Report 2014/816, 2014. https://eprint.iacr.org/2014/816.

[5] L. Ducas and D. Micciancio. Fhew: Homomorphic encryption bootstrapping in less than a second1. University Lecture, 2015.

[6] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[7] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[8] X. Lei, R. Guo, F. Zhang, L. Wang, R. Xu, and G. Qu. Accelerating homomorphic full adder based on fhew using multicore cpu and gpus. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 2508–2513, 2019.

[9] D. Micciancio and Y. Polyakov. Bootstrapping in fhew-like cryptosystems. *IACR Cryptol. ePrint Arch.*, 2020:86, 2020.

[10] V. Migliore, C. Seguin, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, G. Gogniat, and R. Tessier. A high-speed accelerator for homomorphic encryption

using the karatsuba algorithm. *ACM Trans. Embed. Comput. Syst.*, 16(5s), Sept. 2017.

[11] NuCypher. Nufhe, a gpu-powered torus fhe implementation. URL: https://nufhe.readthedocs.io/en/latest/, last checked on 2020-03-26.

[12] T. Pratchett and N. Gaiman. *Good Omens:* The Nice and Accurate Prophecies of Agnes Nutter, Witch. HarperTorch (reprint), 2006. ISBN-13: 978-0060853983.

[13] S. Roy, F. Turan, K. Järvinen, F. Vercauteren, and I. Verbauwhede. Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data. In *2019 25TH IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE COMPUTER ARCHITECTURE (HPCA)*, International Symposium on High-Performance Computer Architecture-Proceedings, pages 387–398, United States, 2019. IEEE. International symposium on high performance computer architecture, HPCA ; Conference date: 16-02-2019 Through 20-02-2019.

[14] S. Sinha Roy, K. Järvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede. Hepcloud: An fpga-based multicore processor for fv somewhat homomorphic function evaluation. *IEEE Transactions on Computers*, 67(11):1637–1650, 2018.

[15] Wikipedia. Thesis or dissertation. URL: http://en.wikipedia.org/wiki/Thesis_or_dissertation, last checked on 2010-01-07.