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. Angshuman Karmakar

Academic year 2020 – 2021

Preface

Thanks and acknowledgement

Jonas Bertels

Contents

Preface i

Abstract iv

List of Figures and Tables v

List of Abbreviations and Symbols vi

1 Introduction to homomorphic encryption 1
- 1.1 An Introduction To Encryption . . . 1
- 1.2 An Introduction To Learning With Errors . . . 2
- 1.3 Homomorphic Encryption . . . 3
- 1.4 Encryption Libraries And Homomorphic Encryption Schemes . . . 5
- 1.5 FHEW (Fastest Homomorphic Encryption In The West) . . . 6
- 1.6 Hardware Acceleration . . . 6

2 Preliminaries 9
- 2.1 Definitions . . . 9
- 2.2 Ring Learning With Errors . . . 9
- 2.3 Fan Vercauteren . . . 9
- 2.4 FHEW (Fastest Homomorphic Encryption In The West) . . . 10
- 2.5 Notes On Previous Optimisations . . . 17

3 The Number Theoretic Transform 19
- 3.1 Introduction To Number Theoretic Transform . . . 19
- 3.2 Hardware Implementations Of NTT . . . 22
- 3.3 Implementing Bit Reversal and Multiplication With Psi . . . 22
- 3.4 Conclusion . . . 22

4 The Final Chapter 25
- 4.1 The First Topic of this Chapter . . . 25
- 4.2 The Second Topic . . . 26
- 4.3 Conclusion . . . 27

5 Conclusion 29

A FHEW Algorithm flowchart 33

B The Last Appendix 35
- B.1 Lorem 20-24 . . . 35
- B.2 Lorem 25-27 . . . 36

Bibliography **37**

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

1.1 One Time Pad [16] 2
1.2 The jewel box [2] 4

2.1 Binary Message representation of $m * \frac{q}{2} + e$ in $\text{LWE}_s^2(m, \frac{q}{4})$ (taken from[7]) 11
2.2 Messages in $\text{LWE}_s^4(m, \frac{q}{8})$ [7] 11
2.3 Messages in $\text{LWE}_s^4(m, \frac{q}{16})$ with smaller error [7] 12
2.4 Addition of messages [7] 12
2.5 Result of the addition [7] 12
2.6 Rounding the sum [7] 13
2.7 Creating an AND gate [7] 13
2.8 The final NAND gate [7] 13
2.9 Our RLWE ciphertext is encrypted under a RGSW ciphertext so that it can be decrypted using RLWE secret keys, which themselves have been encrypted under RGSW 14

3.1 Cooley Tukey FFT/NTT butterfly [4] 20
3.2 Gentleman Sande FFT/NTT butterfly [4] 20

List of Tables

2.1 Creating an AND gate from the sum of 2 bits 12
2.2 Initial polynomial for v=1 and n=8 16
2.3 Initial polynomial for v=1 and n=8 16

3.1 Hardware implementations of NTT (green is open source, yellow is open source (to be released) and red is closed source) 23

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

1.1 An Introduction To Encryption

Let's say it's 1943 and you are the head of a resistance movement that helps downed pilots escape. To get the pilots out of Europe, you need to be able to communicate to Britain to be able to arrange a pick-up in Spain. The only way to do this quickly and reliably is via radio. However you are met with a serious problem: anyone can listen in to your broadcast. Therefore you need to send your message in a code. The British Special Operations Executive provided booklets called "one-time pads". These contained long rows of letters, as seen on Figure 1.1 and were used to turn the message, which we call "plain-text" into a jumble of seemingly meaningless letters. This process of turning plain-text messages into messages that cannot be understood by eavesdroppers is called encryption. Only the receiver in Britain, with the only copy of the one-time pad in the world, would be able to turn the encrypted message back into the original message.

What has been described is called symmetric key encryption, because both the sender and receiver are using the same key. The use of it in WW2 posed certain problems: records were kept of all intercepted messages by the occupying forces, even when they were not understood. This way, when a radio operator and his one-time pad were captured, earlier messages could still be deciphered. Secondly, some way had to be found to provide radio operators with the one-time pad booklet, and a new booklet had to be created for every radio operator.

How to provide a user with a key without accidentally giving an eavesdropper the opportunity to acquire this key has always been a problem. To solve this, public key cryptography was created. In public key cryptography, there is both a secret and a public key. The receiver (we'll call her Alice) makes her public key available to anyone who wants it, including potential eavesdroppers. However, the public key can only be used to *encrypt* the data. Once a potential sender (we'll call him Bob)

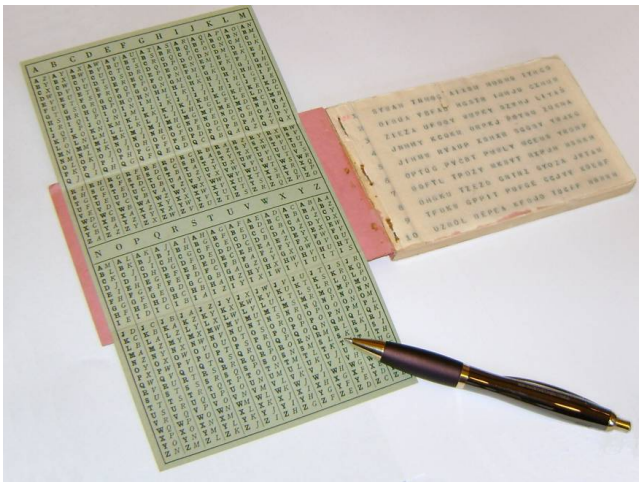


Figure 1.1: One Time Pad [16]

has encrypted his message using the public key, he cannot reverse this process.

It is as if Alice had sent every person in the world a safe with a lock but without a key. If someone like Bob wants to send Alice a message, they simply take the safe, place their message inside of it, and lock the safe. Bob can't unlock the safe anymore, but he can send the safe to Alice. Alice has a key that fits for all the safes she sent out, so she can unlock the safe and retrieve Bob's message, without an eavesdropper (who we'll call Eve from now on) being able to read the message, even if Eve has full access to the safe which contains Bob's message.

1.2 An Introduction To Learning With Errors

Because there are many uses for cryptography, from securing messages to securing bank accounts, there are many cryptographic schemes, i.e., ways of encrypting and decrypting information. As previously mentioned, public/secret key schemes need to have a public key (i.e., a safe in which a sender can store information) and a secret key (i.e., a key to that safe that the receiver can use). If you are the receiver and you've chosen a secret key, you should be able to create a public key from this secret key. However the opposite should obviously be impossible. After all, the public key is being given away to anyone and should not give (much) information about the secret key. Given enough time, you might be able to reverse engineer the public key into a secret key, so we call a scheme "secure" (i.e. good enough) if it takes so long and requires so many resources that it might as well be impossible. Impossible meaning that the required time to break the secret key is on the order of a small country dedicating all its computers for millennia for high enough security needs.

The schemes that were used over the last 50 years did a pretty good job of this. However, quantum computers, which may become a commercial reality in the future, could speed up the process of breaking these schemes massively. This means that there is a pressing need for new schemes that are based on public keys that cannot be reverse engineered into their secret keys.

Learning With Errors is a basis for many of these new schemes. The scheme approximately consists of taking the secret key and (part of) the public key, multiplying them together and then adding some random error. The result of this operation doesn't contain enough information to reverse engineer the secret key , and so the result can be distributed to anyone safely. After all, for a given matrix A and a known vector b, finding the secret vector s in $A * s = b$ is easy. But finding s when a random error e is added to form $A * s + e = b$ is hard.

In addition to being resistant to quantum attacks, LWE has another neat feature: it can be used for Homomorphic Encryption.

1.3 Homomorphic Encryption

Homomorphic encryption means encryption on which functions can still be executed. If you encrypt the value 3 and you encrypt the value 5, you can send both these encrypted values to a server and ask the server to add them together, then return the result back to you. Normally, the decrypted result would be some useless random value. Under a homomorphic encryption scheme however, the result after decryption would actually be 8. Fast, working homomorphic encryption would have a wide-ranging impact on the financial sector, the web services sector and many other sectors in which a company in need of computation cannot trust the providers of computation to access their data.

There are two types of homomorphic encryption: Somewhat Homomorphic Encryption and Fully Homomorphic Encryption. Somewhat Homomorphic Encryption allows for a limited number of computations, Fully Homomorphic Encryption allows for an infinite number of computations. In 2009 Gentry [9] showed that Fully Homomorphic encryption, i.e. doing an arbitrary amount of operations on encrypted data, is possible.

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

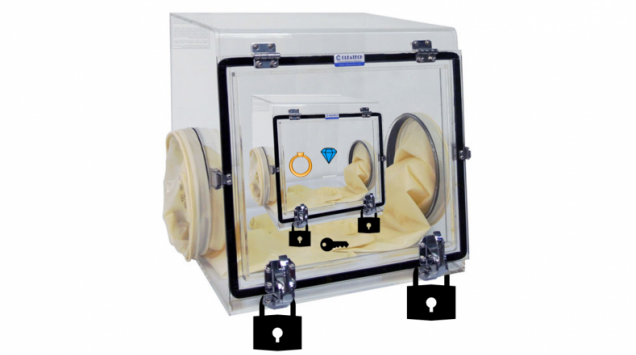


Figure 1.2: The jewel box [2]

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/secret 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).

We call this unlocking while keeping the box locked under a different box "bootstrapping". We call it "bootstrapping" because it allows us to keep working on the encrypted message.

Then our workers can open the inner box and continue working on the jewels (see Figure 1.2). 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, unless of course we bootstrap in time to reduce the noise by performing a decryption.

1.4 Encryption Libraries And Homomorphic Encryption Schemes

~~Because Homomorphic Encryption would be so widely applicable,~~ several software libraries have been created to provide software developers with the tools required to use schemes in their messenger apps, database programs, financial application software, etc ....

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 [8] 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 ($\tilde{1}$00ms)[12]. 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.

1.4.1 Fan-Vercauteren: Usually Somewhat Homomorphic Encryption

FV has been the subject of hardware acceleration by multiple groups [13] [19] [17]. It is also implemented in the homomorphic encryption libraries SEAL and PALISADE among others.
The problem with the second generation schemes as previously mentioned is that bootstrapping takes so long the schemes 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 a certain class of programs with a limited amount of operations but a large amount of data. However, they only allow for a limited number of 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 [17]. 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.

1.5 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[5] [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 128-bit security on an intel i7 [12]. 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.

Because of its flexibility, a FHEW scheme that operated fast enough would have a large advantage in usability over the FV scheme. Programmers using FHEW do not need to worry about the noise growth of the scheme they are using, or when they should return their data back to the client. The entire FHEW scheme can be treated as a black box by those using it, and it is for this reason that this scheme was chosen to be accelerated.

1.6 Hardware Acceleration

We have previously mentioned the term *Hardware Acceleration* when talking about an implementation of Fan-Vercauteren. Hardware acceleration means building a specialized electronic circuit to make a certain function run faster (in less time) or using less energy. Electronics built for one function only are faster than electronics built to execute software. The reason for this is simple: general purpose circuits need to occupy themselves with reading the instructions that are sent to them. Specialized circuits don't need to do this, since they would always get the same instructions, and as such don't need to waste time on understanding what they have to execute and can dedicate all their resources to executing very specific operations. In this thesis, we use a FPGA, which is a form of *programmable* electronics, where different electrical components are wired together according to a netlist that can be programmed from a computer.

The main advantage of hardware acceleration is that it can provide a speedup of a factor 10 to a factor 100 for a given task.[17] The disadvantage is that by its very nature, it requires specialized electronics to be used. Those electronics might one day end up in a consumer device, such as laptop, smartphone, or in the case of homomorphic encryption, a server, but this is unlikely to occur soon, and requires significant investment. Hardware requires a new chip to become part of the server hardware for ASIC, or for cloud servers to already contain a FPGA. Additionally, the encryption schemes which are hardware accelerated are still in flux, and there is no guarantee that a much faster scheme isn't just over the horizon, rendering the investments in a specially built ASIC pointless. Software is much easier to deploy, and requires no expensive investment. However, recently web hosting companies such as Amazon Web Services have started making servers with programmable FPGA's available [18], which makes deploying hardware almost as easy as deploying software. Given that the computational bottlenecks for Homomorphic Encryption schemes mainly lie at the server side, hardware acceleration becomes a perfect solution to the speed problems faced by the homomorphic encryption schemes.

Chapter 2

Preliminaries

2.1 Definitions

~~A lot of~~ lattice-based cryptography uses polynomial rings for various optimizations. The polynomials rings we use are defined as $R = \mathbb{Z}[x]/f(x)$ with $f(x) = x^d + 1$ and $d = 2^m$ with $m \in \mathbb{N}$. We also define $R_q = R/qR = \mathbb{Z}_q[x]/f(x)$, where again $f(x) = x^d + 1$ and $d = 2^m$ but $\mathbb{Z}_q$ defines the set of integers between $(-q/2, q/2]$.~~[8]~~

2.2 Ring Learning With Errors

The Fan-Vercauteren scheme [8] 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. In other words, (a_i, b_i) is a good public key 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.

For our error terms, we consider the discrete Gaussian sampling function $D_{\mathbb{Z},\sigma}$. To get our error term, we can simply sample our coefficients from this Discrete Gaussian function, so that the error distribution $\chi = D^d_{\mathbb{Z},\sigma}$.

2.3 Fan Vercauteren

Consider the following simplified Fan Vercauteren scheme:.

The first part of our public key is:

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

with

$$a \in R_q, s \in R_q \text{ and } e \in \chi \tag{2.2}$$

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

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

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. [8].

2.4 FHEW (Fastest Homomorphic Encryption In The West)

2.4.1 Additional definitions

Our thesis focuses on FHEW. To explain how it works in detail, we must therefore create some additional definitions. Since, for a hardware engineer, the main interest lies in the parameter sizes, we will provide numbers along with every parameter that we define for the parameter set (STD 128 [12]) that we have decided to use.

First we redefine the RLWE encryption of a message $\tilde{m} \in R_q$ with a secret key s as: [12]

$$RLWE_s(\tilde{m}) = (a, as + e + \tilde{m}) \tag{2.5}$$

with as previously defined, $a \leftarrow R_q$ and randomly sampled and $e \leftarrow \chi_\sigma^d$ with χ_σ being a discrete Gaussian distribution with σ as parameter which is sampled d times (once for every coefficient) to form χ_σ^d. The secret key s can be sampled from R_q or from a smaller distribution. Using smaller distributions imporves the performance, but also of course reduces the security. For the purpose of this work we focus on Gaussian (i.e. the most general) secret keys.

We have previously mentioned that our actual message m is a bit, not a integer. We have $\tilde{m} = \frac{q}{t} * m$ with q as always our modulus (=512 in our parameter set) and t being dependent on which LWE scheme we use (we write LWE_s^t). Figure 2.1 shows a representation of such a message. From the representation, it is immediately obvious that the error should not exceed $\frac{q}{4} = 128$.

We can now define our decryption function as:

$$RLWE_s^{-1}(a, b) = b - a * s = \tilde{m} + e \tag{2.6}$$

We can then recover m from $\tilde{m}$ by multiplying with $\frac{t}{q}$ and rounding the error away. We can now describe how to perform a NAND gate using this scheme in such a way that there is only a little bit of error growth.

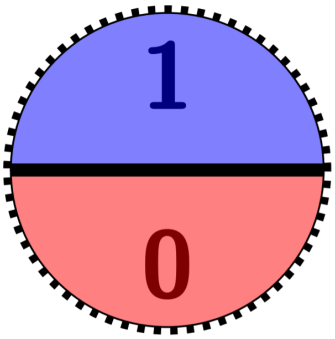


FIGURE 2.1: Binary Message representation of $m * \frac{q}{2} + e$ in $\text{LWE}_s^2(m, \frac{q}{4})$ (taken from[7])

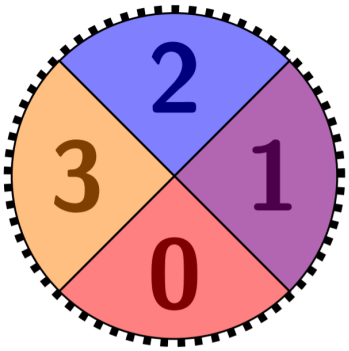


FIGURE 2.2: Messages in $\text{LWE}_s^4(m, \frac{q}{8})$ [7]

2.4.2 The Decryption Function

As previously mentioned, 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 $\frac{q}{2}(= 256)$ with some error added to it, as shown in Figure 2.1. In this case, our maximum error is equal to +/- $\frac{q}{4} = 128$. Thinking back to our parameter t, we see that this scheme uses $t = 2$. We can now define The representation that we uses $t = 4$. This representation allows for 4 possible messages, with the message being multiplied by $\frac{q}{t} = 128$, which is $\frac{q}{4}$ (Figure 2.2). Our maximum error value in this case is +/- $q/8 = 64$. Nothing stops us from reducing the possible error further though, as can been seen on the right of Figure 2.2, so that there are values that are never reached by a given message and error (white space on Figure 2.2).

We can use these messages for addition (Figure 2.4). After addition, the result now has more error than the beginning (Figure 2.5), because the noise adds up.

Our interest lies in making a functional NAND gate. For an NAND gate, consider table 2.1. We first create an AND gate from the addition of 2 encrypted messages. If

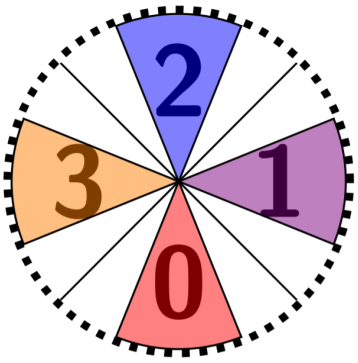


Figure 2.3: Messages in $\mathrm{LWE}_s^4(m, \frac{q}{16})$ with smaller error [7]

Result of the Sum	0	1	2	3
Result of a AND gate	False (0)	False (0)	True (1)	N/A

Table 2.1: Creating an AND gate from the sum of 2 bits

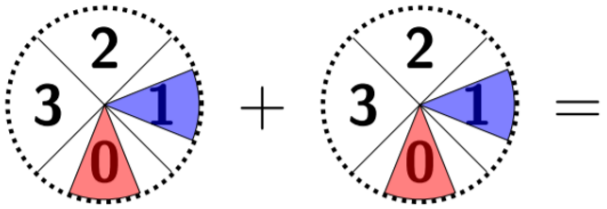


Figure 2.4: Addition of messages [7]

the resulting sum's error stays small enough to decrypt correctly (i.e., if the colours of Figure 2.5 do not shade into each other), we can map a result of 2 to a True outcome of the AND gate, and a result of 0 or 1 to a False outcome of the AND gate.

If we consider everything on the top left to be 1 and everything on the bottom right to be 0, we have an AND gate (Figure 2.7 and table 2.1). To make this an NAND, we rotate by $\frac{q}{2}$, in other words, we consider the bottom right to be 1 and

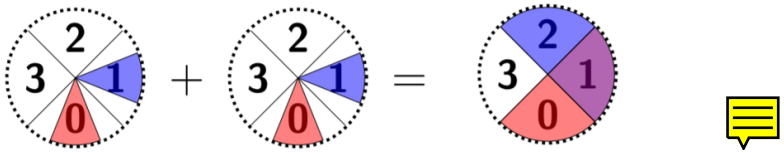


Figure 2.5: Result of the addition [7]

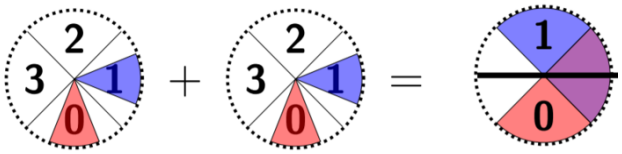


FIGURE 2.6: Rounding the sum [7]

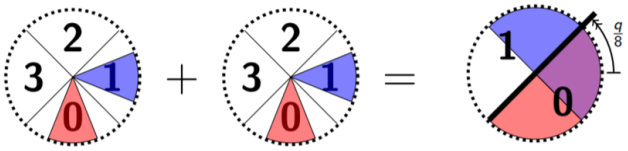


FIGURE 2.7: Creating an AND gate [7]

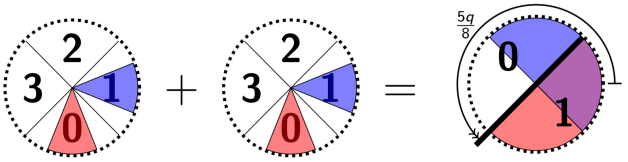


FIGURE 2.8: The final NAND gate [7]

the other half to be a 0, and this results in the results that we expect from a NAND gate for the given input values (Figure 2.8).

2.4.3 Introduction To Bootstrapping In FHEW

What we've discussed until now is the *underlying* encryption system. If we go back to our analogy from the introduction, our RLWE scheme is the box that the workers use to assemble the jewels. The gloves on this box are one-use only (we only do one NAND gate, or a similarly small binary gate operation). The workers now need a larger box in which take can put this RLWE box, so that they can unlock the RLWE box and put the jewels in a new (small, RLWE) box with new gloves. This (larger) box will be provided by RGSW, which we will define in the next paragraph. By encrypting our secret keys under RGSW, we can safely send them to the server without breaking security by giving the server the means to decypher the RLWE ciphertext into plaintext. Instead, we will decypher our RGSW(RLWE(m)) into a RGSW(m), just like the jewels never leave the security of the outer box once the inner box has been unlocked.

As mentioned before this process is called bootstrapping. Bootstrapping is done by executing decryption operations homomorphically, i.e. doing the decryption operations with an encrypted secret key. This reduces the noise back down to the

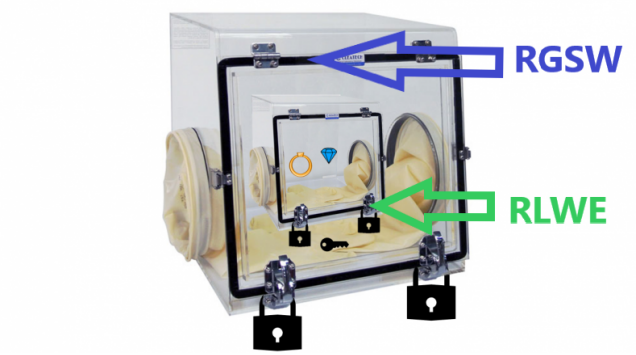


Figure 2.9: Our RLWE ciphertext is encrypted under a RGSW ciphertext so that it can be decrypted using RLWE secret keys, which themselves have been encrypted under RGSW

original level, completing our algorithm.

2.4.4 Definitions For Bootstrapping

We define RLWE' [6] [12]:

$$\text{RLWE}'_s(m) = (\text{RLWE}_s(m), \text{RLWE}_s(B*m), \text{RLWE}_s(B^2*m), ..., \text{RLWE}_s(B^{k-1}*m)) \tag{2.7}$$

with $B^k = q$. (In our parameter set, $B_\text{r} = 128 = 2^7$). The reason for this definition is that for the outer box, we want a system that creates as little noise as possible to ensure that the bootstrapping step doesn't add more noise than it removes. By breaking RGSW into bases, less noise is generated. Multiplications are defined as:

$$(\odot) : R \times \text{RLWE}' \to \text{RLWE} : d \in R, \mathbf{c} \in \text{RLWE}' : d \odot (\mathbf{c}_0, ..., \mathbf{c}_{k-1}) = \sum_i d_i \mathbf{c}_i \tag{2.8}$$

We can extend this multiplication definition to a multiplication that results in RLWE':

$$(\odot') : R \times \text{RLWE}' \to \text{RLWE}' : d \odot \mathbf{C} = ((B^0 * d) \odot \mathbf{C}, (B^1 * d) \odot \mathbf{C}, ..., (B^{k-1} * d) \odot \mathbf{C}) \tag{2.9}$$

And finally, using these definitions, we can define our RGSW scheme, which will allow us to multiply ciphertexts together. We need this ability if we want to be able to run our RLWE decryption while everything is encrypted under RGSW.

$$\text{RGSW}_s(m) = (\text{RLWE}'_s(-s * m), \text{RLWE}'_s(m)) \tag{2.10}$$

To be able to use our secret key s encrypted under RGSW, we need to be able to multiply a RGSW ciphertext RGSW(m_1) = ($\mathbf{c}$, $\mathbf{c}$') with a RLWE' ciphertext

$\text{RLWE}_s(m_0, e_0) = (a, b)$ and get a result that when decrypted gives the product of the 2 messages [12]:

$$(a, b) \diamond (\mathbf{c}, \mathbf{c'}) = \langle (a, b), (\mathbf{c}, \mathbf{c'}) \rangle = a \odot \mathbf{c} + b \odot \mathbf{c}' \tag{2.11}$$

$$= a \odot \text{RLWE'}_s(-s * m_1) + b \odot \text{RLWE'}_s(m_1) \tag{2.12}$$

$$= \text{RLWE}_s((b - a * s) * m_1) = \text{RLWE}_s((m_0 + e_0) * m_1) \tag{2.13}$$

This $\text{RGSW} \times \text{RLWE}' \to \text{RLWE}$ can be turned into a $\text{RGSW} \times \text{RLWE}' \to \text{RLWE}'$, and then we have everything we need to do our bootstrapping.

2.4.5 Bootstrapping Algorithm

We now turn our attention to the bootstrapping process itself, first formally using the algorithm described in [12]. The bootstrapping consists of 3 steps: Initialization (see Algorithm 1), Accumulation (2) and Extraction (which simply consists of taking the first value of the second part of the RGSW result).

Algorithm 1: Initilization[12]

Data: b such that $(\mathbf{a}, b) = c_0 + c_1$ with c_0 and c_1 encrypted input of NAND
Result: $\mathbf{ACC}_f[b] = \text{RLWE}\left(\sum_{i=0}^{q/2-1} f(b-i) * X^i\right) \in R_q^2$
begin
  for $i = 0, 1, ...q/2 - 1$ **do**
    $m_i = f(b - i)$;
    Note that f is our look-up table
  end
  $\mathbf{m} = \sum_{i<q/2} m_i * X^i = (m_0, m_1, ..., m_{q/2-1})$;
  Return $(0, \mathbf{m})$;
end

The way that we implement both our rounding function and NAND gate that we previously described is as a look-up table. In a look-up table, the result of an input i is stored at index i and executing the function thus simple requires looking at the value at index i. In this scheme, a polynomial is used as look-up table. To explain our scheme however, we will use tables, and reduce the length of the table from $n = 512$ to $n = 8$.

Our look-up table is initialized to the formula seen on 1 (see table 2.2): This look-up table is shifted by our initial value so that the result at extraction will be found under index 0.

What we have after initialization is effectively a noiseless RLWE encryption of this polynomial. We can then perform the decryption of this RLWE encryption using

Algorithm 2: A Single Accumulation Step[12]

Data: **ACC**, a RGSW ciphertext and $E(s) = \{\mathbf{Z}_{j,v} = \text{RGSW}(X^{v*B_r^s*s} | j < \log_{B_r}(q), v \in \mathbb{Z}_{B_r}\} \in \text{RGSW}^{B_r*\log_{B_r}(q)}$

Result: **ACC**, updated with $a_i * s_i$

begin

  for $j = 0, 1, ... \log_{B_r}(q) - 1$ **do**

    $c_j = \lfloor c/B_r^j \rfloor \bmod B$;

    if $c_j > 0$ **then**

      $\mathbf{ACC} \leftarrow \mathbf{ACC} \diamond \mathbf{Z}_{j,c_j}$;

    end

  end

  Return **ACC**;

end

Index of Coefficient	7	6	5	4	3	2	1	0
Value of Coefficient	0	0	0	0	f(-2) = f(2)	f(-1) = f(3)	f(0)	f(1)

Table 2.2: Initial polynomial for v=1 and n=8

Index of Coefficient	7	6	5	4	3	2	1	0
Value of Coefficient	0	0	f(-2) = f(2)	f(-1) = f(3)	f(0)	f(1)	0	0

Table 2.3: Initial polynomial for v=1 and n=8

secret keys encrypted under the RGSW scheme because a RLWE cipthertext is ~~after all~~ just a single component of a RGSW ciphertext (table 2.2).

Now, addition on the RLWE level is then performed using multiplication on the RGSW level. The reason for this is again our look-up table. When we rotate the look-up table leftward, from our RLWE perspective we are subtracting $a_j * s_j$ from the index (which was originally v). From a RGSW perspective, a rotation is a multiplication of the polynomial with a value $X^{a_j*s_j}$. This is how we can use the multiplication $RGSW \odot' RLWE' \rightarrow RLWE'$ to perform the decryption $b - \mathbf{a} * \mathbf{s}$. Table 2.3 shows this rotation through a look-up table.

After extracting the correct value from the polynomial (i.e. the coefficient of X^0), which is the constant term or the rightmost value in our table, a key-switching step occurs to change from the keys used in the Accumulator back to the keys of the RLWE. This step does not concern this thesis as we are mainly interested in resolving the bottleneck operation of FHEW, which is in the accumulator loop.

2.5 Notes On Previous Optimisations

The version that is accelerated in this thesis is an optimized version([12]) of the FHEW scheme originally introduced in 2014 [6]. ~~This was chosen because it was the simplest of all third generation schemes~~. The only optimisation paper on this scheme 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. [10]

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 implementations using the NTT it is 0.35 ms/bit [14]. However, this scheme implemented full adders and took advantage of the fact that multiple bits can be packed into one cipher text, then divided the total bootstrapping speed by the number of bits in one adder. No attempt at packing is made in this thesis.

Chapter 3

The Number Theoretic Transform

3.1 Introduction To Number Theoretic Transform

The Number Theoretic Transform [4] [11] [15] is an algorithm for performing polynomial (ring) multiplication. Most variants of it are almost identical to the Fast Fourier Transform, meaning it executes quickly (in $O(\frac{n}{2} \log n)$ time). The idea of the algorithm is to transform a polynomial into a form that allows polynomial multiplication to be done as point-wise multiplication, which executes in linear time. Then the result is transformed back into polynomial form. Given that executing the addition of one secret key to our accumulator requires one Number theoretic transform for every element of the RGSW, it is obvious why high performance for this transformation is necessary.

3.1.1 Mathematics Behind NTT

Since the NTT is a Discrete Fourier Transform with the twiddle factor changed from a complex number so that $\omega^n = 1$ to a $\omega \in \mathbb{Z}_q$ with again $\omega^n = 1$.
The discrete Fourier is given by the expression ([4], equation 3.1).

$$X_r = \sum_{l=0}^{N-1} x_l \omega^{rl} \tag{3.1}$$

As always, N is our ring size (=1024 for our parameters) and Q is the modulus of the coefficients (= a 27-bit number)). The expression for the NTT is identical, but with $\omega \in \mathbb{Z}_q$.

There are now 2 methods that we consider for performing the radix-2 FFT/NTT: The Cooley-Tukey and the Gentleman Sande. The easiest way to show how these work is via 2 butterfly diagrams (Figure 3.1 and Figure 3.2).

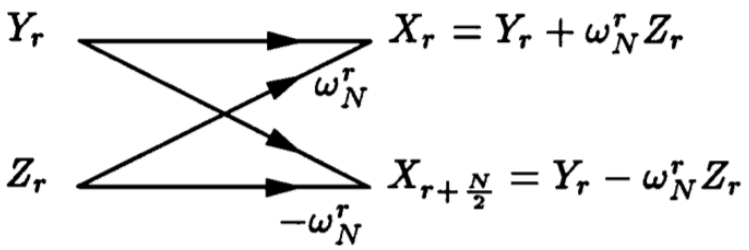


Figure 3.1: Cooley Tukey FFT/NTT butterfly [4]

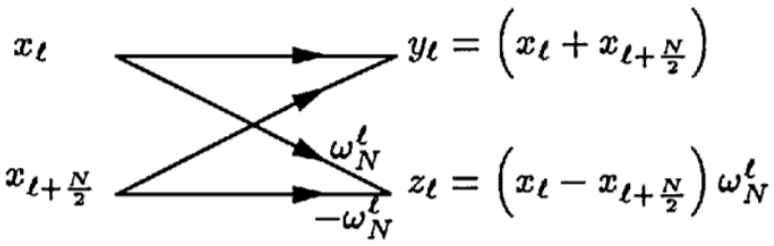


Figure 3.2: Gentleman Sande FFT/NTT butterfly [4]

The Cooley-Tukey Butterfly:

First we split 3.1 into an even and an odd parts:

$$X_r = \sum_{k=0}^{\frac{N}{2}-1} x_{2k}\omega_N^{2rk} + \omega_N^r \sum_{k=0}^{\frac{N}{2}-1} x_{2k+1}\omega_N^{2rk} \tag{3.2}$$

Now we can consider the result of this equation for 2 cases: for the case that $r \in [0, \frac{N}{2} - 1]$ and $r \in [\frac{N}{2}, N-1]$. In the first case, the equation does not vary, in the second case, because $\omega_N^{r+N/2} = -\omega_N^r$, we are subtracting the NTT of the odd coefficients from the NTT of even coefficients instead of adding them together. Note that we use ω^2 instead of ω for calculating these NTT's.

The Gentleman-Sande Butterfly:

The Gentleman-Sande works by splitting 3.1 into a first half and second half:

$$X_r = \sum_{k=0}^{\frac{N}{2}-1} x_k\omega_N^{rk} + \sum_{k=\frac{N}{2}}^{N-1} \omega_N^{rk} = \sum_{k=0}^{\frac{N}{2}-1} (x_k + x_{k+\frac{N}{2}}\omega_N^{r\frac{N}{2}})\omega_N^{rk} \tag{3.3}$$

For even $r = 2 * l$, we have $\omega_N^{r\frac{N}{2}} = \omega_N^{2*l*\frac{N}{2}} = 1$, so this equation becomes:

$$X_{2l} = \sum_{k=0}^{\frac{N}{2}-1} (x_l + x_{l+\frac{N}{2}})\omega_N^{2kl} \tag{3.4}$$

and for odd $r = 2 * l + 1$ we have $\omega_N^{r\frac{N}{2}} = \omega_N^{(2*l+1)*\frac{N}{2}} = -1$ so:

$$X_{2l+1} = \sum_{k=0}^{\frac{N}{2}-1} (x_l - x_{l+\frac{N}{2}})\omega_N^{(2l+1)k} \tag{3.5}$$

3.1.2 The Polynomial Multiplication And Negatively Wrapped Convolution

To perform the polynomial multiplication, we must not just perform an NTT but also an INTT. ~~Luckily,~~ the INTT is defined as: [4]

$$X_r = N^{-1} \sum_{l=0}^{N-1} x_l \omega^{-rl} \tag{3.6}$$

Therefore, the INTT can be performed as an NTT by simply changing the ω to their inverse and performing multiplication with the modular inverse of n. ($N^{-1} * N = 1 \pmod q$).
Most implementations of NTT result in an output with bit-reversed indices [4]. This means that to find the correct value, it is necessary to take the index of that value, write it in binary form for $\log_2 N$, and "flip" this index in such a way that for example 1010000000 become 0000000101. Then we use this flipped index to find the correct value. Indeed, in the PALISADE library's implementation, the NTT implementation takes a normally-order polynomial and returns a bit-reversed, transformed polynomial. The INTT does the opposite. As such, in the PALISADE library implementation there is no need for extra steps to undo this bit-reversal.
Normally the following optimisation is applied to polynomial multiplication with NTT. In the standard case we multiply together 2 polynomials of ring size $2N$ after an NTT with half of the inputs being zeros, and do a reduction step of the polynomial modulo $X^N + 1$ [15]. Instead we use the *negatively wrapped convolution*, where we use a ψ so that $\psi^2 = \omega \pmod q$, and we calculate

$$\hat{a} = (a[0], a[1] * \psi, a[2] * \psi^2, ..., a[n-1] * \psi^{n-1}) \tag{3.7}$$

and

$$\hat{b} = (b[0], b[1] * \psi, b[2] * \psi^2, ..., b[n-1] * \psi^{n-1}) \tag{3.8}$$

We can then calculate

$$\hat{c} = \mathbf{INTT}(\mathbf{NTT}(\hat{a}) \circ \mathbf{NTT}(\hat{b})) \tag{3.9}$$

and it turns out that for

$$c = \mathbf{a} * \mathbf{b} \pmod{X^N + 1}) \tag{3.10}$$

we have

$$c = (\hat{c}[0], \hat{c}[1] * \psi^{-1}, \hat{c}[2] * \psi^{-2}, ..., \hat{c}[n-1] * \psi^{-(n-1)} \tag{3.11}$$

This multiplication with ψ or ψ^{-1} can be integrated in our multiplication with the ω factors [15]. How this is done depends on how the algorithm is implemented, and as such, we first have to determine which implementation of NTT to choose from.

3.2 Hardware Implementations Of NTT

Because NTT is a common feature among many post-quantum cryptography implementations, there are many HW implementations for it available. As such we created the following tables:

3.3 Implementing Bit Reversal and Multiplication With Psi

3.4 Conclusion

The final section of the chapter gives an overview of the important results of this chapter. This implies that the introductory chapter and the concluding chapter don't need a conclusion.

Author	Year	FPGA	NTT Area	#CC NTT	Frequency	$\mathbf{c} = \mathbf{a} \odot \mathbf{b}$ Area	#CC $\mathbf{c} = \mathbf{a} \odot \mathbf{b}$	N	$\log_2 q$
	2017	Arktix-7	N/A	35840	125			1024	13.6
Roy	2018	UltraScale		87582	200	64K/25K/0.4K/0.2K	5349567	4096	180
Wang	2020	Arktix-7			126	1735/758/6/0	11,455	1024	28.35
	2020	Arktix-7	2908/170/9/0	18537	45.47		36,102	1024	13.6
Roy	2014	Virtex 6	1536/953/1/3	2304	278			512	13.6
Kuo	2017	Artix-7	2832/1381/8/10	2616	131			1024	13.6
Rashmi	2019	Artix-7			0.1	no area for NTT	33792	1024	13.6
Du	2016	Spartan-6			241	251slice/1/4.5	11826	1024	16
Hartshorn	2020	Virtex 7	9233/8585/128/8	1294	179		1936	1024	N/A
Mert	2019	Virtex 7			211.5	1208/556/14/14	7970	1024	32
Mert	2020	Virtex-7	575/0/3/11	5160	125			1024	14
Mert	2020	Virtex-7	2584/0/24/16	680	125			1024	14
Mert	2020	Virtex-7	17188/0/96/48	200	125			1024	14
Mert	2020	Virtex-7	38093/0/224/48	250	125			1024	29

TABLE 3.1: Hardware implementations of NTT (green is open source, yellow is open source (to be released) and red is closed source)

Chapter 4

The Final Chapter

Morbi malesuada hendrerit dui. Nunc mauris leo, dapibus sit amet, vestibulum et, commodo id, est. Pellentesque purus. Pellentesque tristique, nunc ac pulvinar adipiscing, justo eros consequat lectus, sit amet posuere lectus neque vel augue. Cras consectetuer libero ac eros. Ut eget massa. Fusce sit amet enim eleifend sem dictum auctor. In eget risus luctus wisi convallis pulvinar. Vivamus sapien risus, tempor in, viverra in, aliquet pellentesque, eros. Aliquam euismod libero a sem.

4.1 The First Topic of this Chapter

4.1.1 Item 1

Sub-item 1

Nunc velit augue, scelerisque dignissim, lobortis et, aliquam in, risus. In eu eros. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Curabitur vulputate elit viverra augue. Mauris fringilla, tortor sit amet malesuada mollis, sapien mi dapibus odio, ac imperdiet ligula enim eget nisl. Quisque vitae pede a pede aliquet suscipit. Phasellus tellus pede, viverra vestibulum, gravida id, laoreet in, justo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer commodo luctus lectus. Mauris justo. Duis varius eros. Sed quam. Cras lacus eros, rutrum eget, varius quis, convallis iaculis, velit. Mauris imperdiet, metus at tristique venenatis, purus neque pellentesque mauris, a ultrices elit lacus nec tortor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent malesuada. Nam lacus lectus, auctor sit amet, malesuada vel, elementum eget, metus. Duis neque pede, facilisis eget, egestas elementum, nonummy id, neque.

Sub-item 2

Proin non sem. Donec nec erat. Proin libero. Aliquam viverra arcu. Donec vitae purus. Donec felis mi, semper id, scelerisque porta, sollicitudin sed, turpis. Nulla in urna. Integer varius wisi non elit. Etiam nec sem. Mauris consequat, risus nec

congue condimentum, ligula ligula suscipit urna, vitae porta odio erat quis sapien. Proin luctus leo id erat. Etiam massa metus, accumsan pellentesque, sagittis sit amet, venenatis nec, mauris. Praesent urna eros, ornare nec, vulputate eget, cursus sed, justo. Phasellus nec lorem. Nullam ligula ligula, mollis sit amet, faucibus vel, eleifend ac, dui. Aliquam erat volutpat.

4.1.2 Item 2

Fusce vehicula, tortor et gravida porttitor, metus nibh congue lorem, ut tempus purus mauris a pede. Integer tincidunt orci sit amet turpis. Aenean a metus. Aliquam vestibulum lobortis felis. Donec gravida. Sed sed urna. Mauris et orci. Integer ultrices feugiat ligula. Sed dignissim nibh a massa. Donec orci dui, tempor sed, tincidunt nonummy, viverra sit amet, turpis. Quisque lobortis. Proin venenatis tortor nec wisi. Vestibulum placerat. In hac habitasse platea dictumst. Aliquam porta mi quis risus. Donec sagittis luctus diam. Nam ipsum elit, imperdiet vitae, faucibus nec, fringilla eget, leo. Etiam quis dolor in sapien porttitor imperdiet.

4.2 The Second Topic

Cras pretium. Nulla malesuada ipsum ut libero. Suspendisse gravida hendrerit tellus. Maecenas quis lacus. Morbi fringilla. Vestibulum odio turpis, tempor vitae, scelerisque a, dictum non, massa. Praesent erat felis, porta sit amet, condimentum sit amet, placerat et, turpis. Praesent placerat lacus a enim. Vestibulum non eros. Ut congue. Donec tristique varius tortor. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nam dictum dictum urna.

Phasellus vestibulum orci vel mauris. Fusce quam leo, adipiscing ac, pulvinar eget, molestie sit amet, erat. Sed diam. Suspendisse eros leo, tempus eget, dapibus sit amet, tempus eu, arcu. Vestibulum wisi metus, dapibus vel, luctus sit amet, condimentum quis, leo. Suspendisse molestie. Duis in ante. Ut sodales sem sit amet mauris. Suspendisse ornare pretium orci. Fusce tristique enim eget mi. Vestibulum eros elit, gravida ac, pharetra sed, lobortis in, massa. Proin at dolor. Duis accumsan accumsan pede. Nullam blandit elit in magna lacinia hendrerit. Ut nonummy luctus eros. Fusce eget tortor.

Ut sit amet magna. Cras a ligula eu urna dignissim viverra. Nullam tempor leo porta ipsum. Praesent purus. Nullam consequat. Mauris dictum sagittis dui. Vestibulum sollicitudin consectetuer wisi. In sit amet diam. Nullam malesuada pharetra risus. Proin lacus arcu, eleifend sed, vehicula at, congue sit amet, sem. Sed sagittis pede a nisl. Sed tincidunt odio a pede. Sed dui. Nam eu enim. Aliquam sagittis lacus eget libero. Pellentesque diam sem, sagittis molestie, tristique et, fermentum ornare, nibh. Nulla et tellus non felis imperdiet mattis. Aliquam erat volutpat.

4.3 Conclusion

Vestibulum sodales ipsum id augue. Integer ipsum pede, convallis sit amet, tristique vitae, tempor ut, nunc. Nam non ligula non lorem convallis hendrerit. Maecenas hendrerit. Sed magna odio, aliquam imperdiet, porta ac, aliquet eget, mi. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Vestibulum nisl sem, dignissim vel, euismod quis, egestas ut, orci. Nunc vitae risus vel metus euismod laoreet. Cras sit amet neque a turpis lobortis auctor. Sed aliquam sem ac elit. Cras velit lectus, facilisis id, dictum sed, porta rutrum, nisl. Nam hendrerit ipsum sed augue. Nullam scelerisque hendrerit wisi. Vivamus egestas arcu sed purus. Ut ornare lectus sed eros. Suspendisse potenti. Mauris sollicitudin pede vel velit. In hac habitasse platea dictumst.

Suspendisse erat mauris, nonummy eget, pretium eget, consequat vel, justo. Pellentesque consectetuer erat sed lacus. Nullam egestas nulla ac dui. Donec cursus rhoncus ipsum. Nunc et sem eu magna egestas malesuada. Vivamus dictum massa at dolor. Morbi est nulla, faucibus ac, posuere in, interdum ut, sapien. Proin consectetuer pretium urna. Donec sit amet nibh nec purus dignissim mattis. Phasellus vehicula elit at lacus. Nulla facilisi. Cras ut arcu. Sed consectetuer. Integer tristique elit quis felis consectetuer eleifend. Cras et lectus.

Ut congue malesuada justo. Curabitur congue, felis at hendrerit faucibus, mauris lacus porttitor pede, nec aliquam turpis diam feugiat arcu. Nullam rhoncus ipsum at risus. Vestibulum a dolor sed dolor fermentum vulputate. Sed nec ipsum dapibus urna bibendum lobortis. Vestibulum elit. Nam ligula arcu, volutpat eget, lacinia eu, lobortis ac, urna. Nam mollis ultrices nulla. Cras vulputate. Suspendisse at risus at metus pulvinar malesuada. Nullam lacus. Aliquam tempus magna. Aliquam ut purus. Proin tellus.

Chapter 5

Conclusion

The final chapter contains the overall conclusion. It also contains suggestions for future work and industrial applications.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In

hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

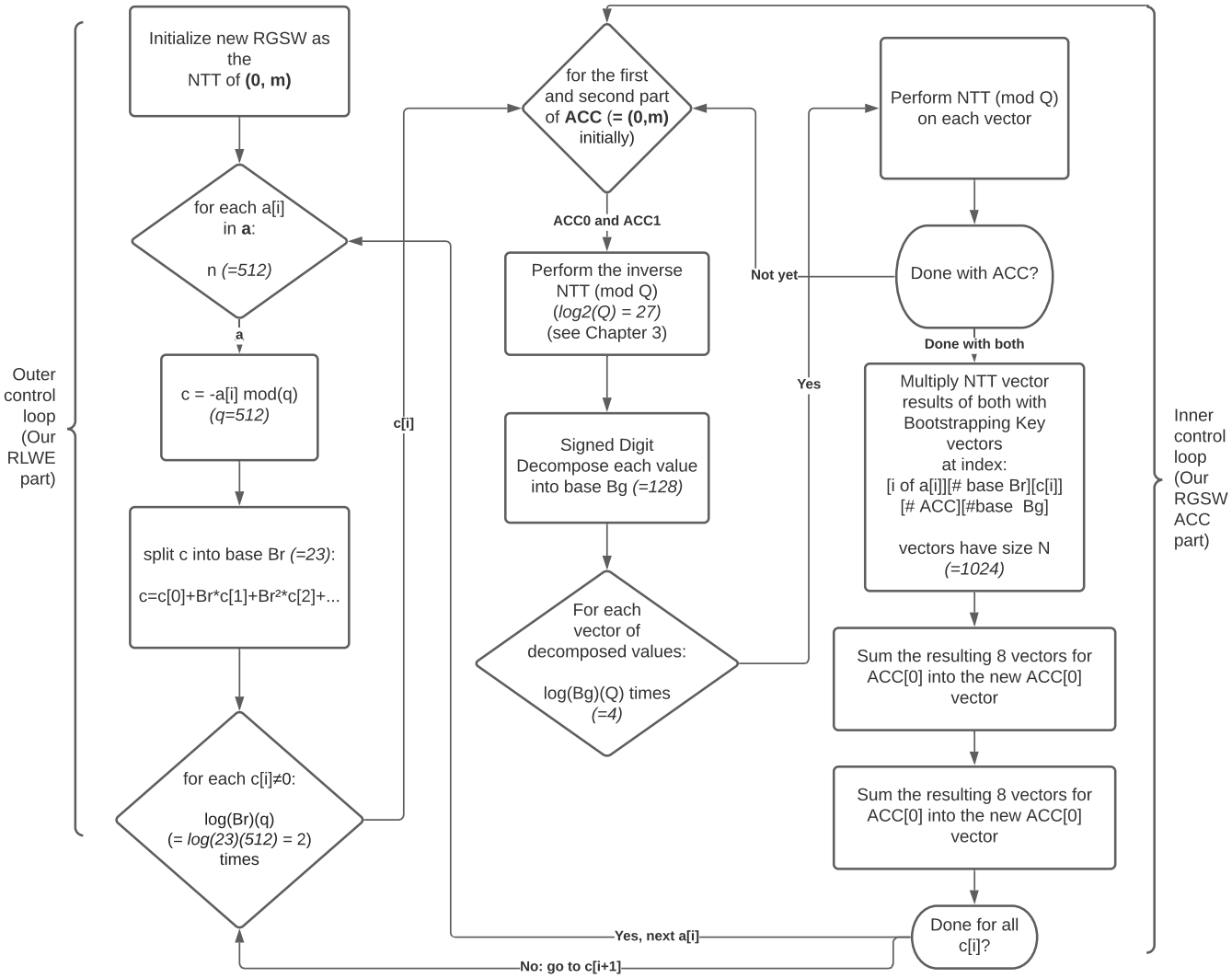
Appendices

Appendix A

FHEW Algorithm flowchart

This flowchart was created from the algorithm found in [12].

perhaps insert full algorithm here as well?


Initialize new RGSW as the NTT of (0, m)
for each a[i] in a:
n (=512)
a
c = -a[i] mod(q) (q=512)
split c into base Br (=23):
c=c[0]+Br*c[1]+Br²*c[2]+...
for each c[i]≠0:
log(Br)(q) (= log(23)(512) = 2) times
c[i]
for the first and second part of ACC (= (0,m) initially)
ACC0 and ACC1
Perform the inverse NTT (mod Q) (log2(Q) = 27) (see Chapter 3)
Signed Digit Decompose each value into base Bg (=128)
For each vector of decomposed values:
log(Bg)(Q) times (=4)
Not yet
Yes
Perform NTT (mod Q) on each vector
Done with ACC?
Done with both
Multiply NTT vector results of both with Bootstrapping Key vectors at index: [i of a[i]][# base Br][c[i]] [# ACC][#base Bg]
vectors have size N (=1024)
Sum the resulting 8 vectors for ACC[0] into the new ACC[0] vector
Sum the resulting 8 vectors for ACC[0] into the new ACC[0] vector
Done for all c[i]?
Yes, next a[i]
No: go to c[i+1]
Outer control loop (Our RLWE part)
Inner control loop (Our RGSW ACC part)

Appendix B

The Last Appendix

Appendices are numbered with letters, but the sections and subsections use arabic numerals, as can be seen below.

B.1 Lorem 20-24

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel

magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

Maecenas non massa. Vestibulum pharetra nulla at lorem. Duis quis quam id lacus dapibus interdum. Nulla lorem. Donec ut ante quis dolor bibendum condimentum. Etiam egestas tortor vitae lacus. Praesent cursus. Mauris bibendum pede at elit. Morbi et felis a lectus interdum facilisis. Sed suscipit gravida turpis. Nulla at lectus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Praesent nonummy luctus nibh. Proin turpis nunc, congue eu, egestas ut, fringilla at, tellus. In hac habitasse platea dictumst.

B.2 Lorem 25-27

Vivamus eu tellus sed tellus consequat suscipit. Nam orci orci, malesuada id, gravida nec, ultricies vitae, erat. Donec risus turpis, luctus sit amet, interdum quis, porta sed, ipsum. Suspendisse condimentum, tortor at egestas posuere, neque metus tempor orci, et tincidunt urna nunc a purus. Sed facilisis blandit tellus. Nunc risus sem, suscipit nec, eleifend quis, cursus quis, libero. Curabitur et dolor. Sed vitae sem. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Maecenas ante. Duis ullamcorper enim. Donec tristique enim eu leo. Nullam molestie elit eu dolor. Nullam bibendum, turpis vitae tristique gravida, quam sapien tempor lectus, quis pretium tellus purus ac quam. Nulla facilisi.

Duis aliquet dui in est. Donec eget est. Nunc lectus odio, varius at, fermentum in, accumsan non, enim. Aliquam erat volutpat. Proin sit amet nulla ut eros consectetuer cursus. Phasellus dapibus aliquam justo. Nunc laoreet. Donec consequat placerat magna. Duis pretium tincidunt justo. Sed sollicitudin vestibulum quam. Nam quis ligula. Vivamus at metus. Etiam imperdiet imperdiet pede. Aenean turpis. Fusce augue velit, scelerisque sollicitudin, dictum vitae, tempor et, pede. Donec wisi sapien, feugiat in, fermentum ut, sollicitudin adipiscing, metus.

Donec vel nibh ut felis consectetuer laoreet. Donec pede. Sed id quam id wisi laoreet suscipit. Nulla lectus dolor, aliquam ac, fringilla eget, mollis ut, orci. In pellentesque justo in ligula. Maecenas turpis. Donec eleifend leo at felis tincidunt consequat. Aenean turpis metus, malesuada sed, condimentum sit amet, auctor a, wisi. Pellentesque sapien elit, bibendum ac, posuere et, congue eu, felis. Vestibulum mattis libero quis metus scelerisque ultrices. Sed purus.

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] E. Chu, E. Chu, and A. George. *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*. Computational Mathematics Series. CRC-Press, 2000.

[5] 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.

[6] 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.

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

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

[9] 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.

[10] 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.

[11] P. Longa and M. Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. Cryptology ePrint Archive, Report 2016/504, 2016. https://eprint.iacr.org/2016/504.

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

[13] 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.

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

[15] T. Pöppelmann, T. Oder, and T. Güneysu. High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. Cryptology ePrint Archive, Report 2015/382, 2015. https://eprint.iacr.org/2015/382.

[16] D. Rijmenants. One-time pad. URL: http://users.telenet.be/d.rijmenants/en/onetimepad.htm, last checked on 2020-04-13.

[17] 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.

[18] A. W. Services. User guide for linux instances: Fpga instances. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/accelerated-computing-instances.html#fpga-instances, last checked on 2020-04-14.

[19] 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.