

國立臺灣大學理學院數學研究所

碩士論文

Institute of Mathematical

College of Science

National Taiwan University

Master Thesis



LWR, MPLWE 和 MPLWR 上的全同態加密

Fully Homomorphic Encryption on LWR, MPLWE and  
MPLWR

洪逸霖

Yi-Lin Hung

指導教授: 陳君明 博士

Advisor: Jiun-Ming Chen Ph.D.

中華民國 109 年 7 月

July, 2020





## Acknowledgements

經過 2 年的時光，終於完成了這篇論文，我要感謝指導教授陳君明老師帶給我的啟發，以及幫我處理一些困難的部分，此外我也很感謝我的同學以及曾經教過我的老師給我一個環境能夠寫出這篇論文，非常的感謝大家。





## 摘要

我們改良了 Zvika Brakerski 研發的全同態加密系統，改成使用難題假設 LWR 以及 RLWR 而不是原先使用的 LWE 以及 RLWE 難題假設。並且我們用類似的方法使得可以在 MPLWR 難題假設上使用同態加密。在過去，Rosca 證明了難題假設 MPLWE 的安全性，我們同樣使用相似於的方法做成全同態加密。

**關鍵字：** LWR 同態加密、環 LWR 同態加密、中間積 LWE 同態加密、中間積 LWR 同態加密





# Abstract

We modified the fully homomorphic encryption (FHE) scheme produced by Zvika Brakerski with the hardness assumption learning with rounding (LWR) and ring learning with rounded (RLWR) instead of the hardness assumption learning with error (LWE) and ring learning with rounding (RLWE). And we use the similar methods on the hardness assumption middle product learning with rounding (MPLWR), i.e. making it into FHE. In present, Rosca proves the hardness assumption middle product learning with error (MPLWE). We also use "similar" Brakerski ideas to make it into FHE.

**Keywords:** Learning with rounding FHE, Ring learning with rounding FHE, Middle product learning with error FHE, Middle product learning with rounding FHE







# Contents

	<b>Page</b>
<b>Acknowledgements</b>	<b>3</b>
<b>摘要</b>	<b>5</b>
<b>Abstract</b>	<b>7</b>
<b>Contents</b>	<b>9</b>
<b>Denotation</b>	<b>11</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Our result . . . . .	2
1.2 Modular Switching . . . . .	2
1.3 FHE Scheme . . . . .	3
1.4 Compare to LWE (RLWE) . . . . .	3
<b>Chapter 2 Preliminaries</b>	<b>5</b>
<b>Chapter 3 Our Construction</b>	<b>9</b>
3.1 Basic LWR (RLWR) encryption scheme . . . . .	9
3.2 Basic MPLWE encryption scheme . . . . .	10
3.3 Basic MPLWR encryption scheme . . . . .	10
3.4 Key Switching for MPLWE and MPLWR based . . . . .	11
3.5 Key Switching for LWR(RLWR) based . . . . .	13

3.6	FHE scheme . . . . .	15
<b>Chapter 4</b>	<b>Correctness</b>	<b>17</b>
4.1	Correctness of LWR (RLWR) scheme . . . . .	17
4.2	Correctness of MPLWE(MPLWR) scheme . . . . .	18
<b>Chapter 5</b>	<b>Optimization</b>	<b>21</b>
5.1	Bootstrapping and Batching . . . . .	21
5.2	Public Key Compression for LWR . . . . .	21
<b>Chapter 6</b>	<b>Zero knowledge proof</b>	<b>23</b>
<b>Chapter 7</b>	<b>Application</b>	<b>25</b>
<b>Chapter 8</b>	<b>Summary</b>	<b>27</b>
<b>Chapter 9</b>	<b>Future Work</b>	<b>29</b>
<b>References</b>		<b>31</b>





# Denotation

FHE	全同態加密 (Fully Homomorphic Encryption)
MPC	多方運算 (Mutiparty Computation)
(R)LWE	(環) 從誤差中學習 ((Ring) Learning with Error)
(R)LWR	(環) 從四捨五入學習 ((Ring) Learning with rounding)
MPLWE	中間積從誤差中學習 (Middle Product Learning with Error)
MPLWR	中間積從四捨五入中學習 (Middle Product Learning with Round- ing)





# Chapter 1 Introduction

Fully Homomorphic Encryption (FHE) is a scheme which supports both additions and multiplications on ciphertexts. We can use these properties in cloud computing and multiparty computation (MPC) [2]. In 2009, Craig Gentry, using lattice-based cryptography, described the first homomorphic encryption scheme[4]. It starts from somewhat homomorphic encryption scheme and using Gentry’s ideas to make it bootstrappable , i.e., capable of evaluating its own decryption circuit and then at least one more operation. These schemes are so-called first-generation FHE, i.e., bootstrapping the somewhat homomorphic encryption scheme.

In 2011, Zvika Brakerski, Craig Gentry and Vinod Vaikuntanathan publish ”Fully Homomorphic Encryption without Bootstrapping[11]”, representative of second-generation FHE. The second-generation FHE all feature a much slower growth of the noise during the homomorphic encryption. And they are efficient enough for many applications, even without invoking bootstrapping. The security of most of these schemes (second-generation FHE) based on the hardness of the LWE (RLWE)[10] problem, however, the variant LWR (RLWR)[7] does not describe in second-generation. So, we plan to change the scheme on the hardness assumption LWR (RLWR).



## 1.1 Our result

The LWE version has the form  $b = \mathbf{A}'s' + 2e$  where  $b, \mathbf{A}'$  are known and  $s'$  is the secret key and  $e \in \{0, 1\}$  is the error term. We have to random  $e$  to make it secret. We think the error term can neglect and can decide by  $\mathbf{A}, \mathbf{s}$  with LWR based, which is often the advantages of LWR. The LWR based encryption have the basic form  $b = \lceil \frac{1}{2} \mathbf{A} \mathbf{s} \rceil$ . We think the key point to turn encryption into fully homomorphic encryption is double the error term (error term is  $\lceil \frac{1}{2} \mathbf{A} \mathbf{s} \rceil - \frac{1}{2} \mathbf{A} \mathbf{s}$ ). It will finally be modular in the decryption process.

In chapter 3, we also described the MPLWE and MPLWR, which publishes in 2017 and 2019. We use our idea to make MPLWR into FHE and use "similar" Brakerski ideas to make MPLWE into FHE. It means that we introduce a new inner product compatible to MPLWE and MPLWR and use Brakerski ideas to make them into FHE.

## 1.2 Modular Switching

We use the methods described in [11] and change they compatible to our new inner product  $\langle \cdot, \cdot \rangle^\odot$  since the product used in MPLWE and MPLWR are not standard matrix product, they introduced the "middle product" instead. Our work in modular switching step is to make sure the method described in [11] still work this means that we can also use same methods to make them into FHE. To make our proof clear, the definition of inner product we define will become complex, but we just want to prove the properties written in [11].



### 1.3 FHE Scheme

Since the work in key switching, it looks the same for our LWR (RLWR), MPLWE, MPLWR scheme. Actually, LWR (RLWR) scheme is the same as [11], since it is an equivalent modification from original LWE edition while MPLWE and MPLWR scheme should look different, i.e. we should read section 3.4 to know what should do in MPLWE and MPLWR .

### 1.4 Compare to LWE (RLWE)

We can use the hardness assumption LWR (RLWR), MPLWE and MPLWR which usually think to be harder than LWE hardness assumption. In [8], it proves that the worst cases of LWR (RLWR) is harder than LWE (RLWE) average cases. And in [9], it proves that MPLWE is harder than polynomial LWE problems, also in [10], it proves that MPLWR is harder than polynomial LWR problems. And for LWR and MPLWR based encryption, we expand the dimension of public key size to trade-off the error generation.







## Chapter 2 Preliminaries

### Definition 1. (LWE)

For security parameter  $\lambda$ , let  $n = n(\lambda)$  be an integer dimension, let  $q = q(\lambda) \geq 2$  be an integer, and let  $\chi = \chi(\lambda)$  be a distribution over  $\mathbb{Z}$ . The  $\text{LWE}_{n,q,\chi}$  problem is to distinguish the following two distributions: In the first distribution, one samples  $(a_i, b_i)$  uniformly from  $\mathbb{Z}_q^{n+1}$ . In the second distribution, one first draws  $s \leftarrow \mathbb{Z}_q^n$  uniformly and then samples  $(a_i, b_i) \in \mathbb{Z}_q^{n+1}$  by sampling  $a_i \leftarrow \mathbb{Z}_q^n$  uniformly,  $e_i \leftarrow \chi$ , and setting  $b_i = \langle a, a \cdot s_i + e \rangle$ . The  $\text{LWE}_{n,q,\chi}$  assumption is that the  $\text{LWE}_{n,q,\chi}$  problem is infeasible.

### Definition 2. (LWR)

For security parameter  $\lambda$ , let  $n = n(\lambda)$  be an integer dimension, let  $q = q(\lambda) \geq 2$  be an integer and  $p = p(\lambda) < q$ , and let  $\chi = \chi(\lambda)$  be a distribution over  $\mathbb{Z}$ . The  $\text{LWR}_{n,q,\chi}$  problem is to distinguish the following two distributions: In the first distribution, one samples  $(a_i, b_i)$  uniformly from  $\mathbb{Z}_q^{n+1}$ . In the second distribution, one first draws  $s \leftarrow \mathbb{Z}_q^n$  uniformly and then samples  $(a_i, b_i) \in \mathbb{Z}_q^{n+1}$  by sampling  $a_i \leftarrow \mathbb{Z}_q^n$  uniformly and setting  $b_i = \lceil \frac{p}{q} \cdot a \cdot s_i \rceil$ . The  $\text{LWR}_{n,q,\chi}$  assumption is that the  $\text{LWR}_{n,q,\chi}$  problem is infeasible.



**Definition 3.** Let  $d_a, d_b, d, k$  be integers such that  $d_a + d_b - 1 = d + 2k$ . The middle-product

$\odot_d: R_a^{<d}[x] \times R_b^{<d}[x] \rightarrow R^{<d}[x]$  is the map:

$$(a, b) \mapsto a \odot_d b = \left\lfloor \frac{(a \cdot b) \bmod x^{k+d}}{x^k} \right\rfloor$$

We use the same notation  $\odot_d$  for every  $d_a, d_b$  such that  $d_a + d_b - 1 - d$  is non-negative and even.

**Definition 4.** (modular rounding function)

Let  $p$  and  $q$  be integers both larger than 1. A modular rounding function  $[\cdot]_p : \mathbb{Z}_q \rightarrow \mathbb{Z}_p$  as  $[x]_p = \lfloor \frac{p}{q} \cdot x \rfloor \bmod p$ . The rounding function extends component-wise to vectors over  $\mathbb{Z}_q$  and coefficient-wise to polynomials in  $\mathbb{Z}_q[x]$ . Note that we use the same notation as Banerjee et al. [1] for the purpose of coherence. It is also possible to use the floor rounding function  $\mathbf{b} \cdot \mathbf{c}$ , where each element is rounded down to the next smaller integer, as for instance done by Chen et al. [8].

**Definition 5.** (MP distribution)

Let  $n, d > 0, q \geq 2$ , and  $\chi$  a distribution over  $\mathbb{R}^{<d}[x]$ . For  $s \in \mathbb{Z}_q^{<n+d-1}[x]$ , we define the distribution  $\text{MP}_{q,n,d,\chi}(s)$  over  $\mathbb{Z}_q^{<n}[x] \times \mathbb{R}_q^{<d}[x]$  as the one obtained by: sampling  $a \leftarrow U(\mathbb{Z}_q^{<n}[x])$ ,  $e \leftarrow \chi$  and returning  $(a, b = a \odot_d s + e)$ .

**Definition 6.** (MP-LWE)

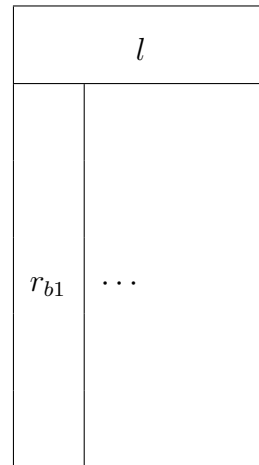
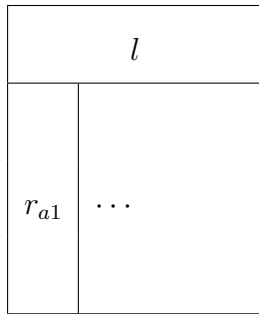
Let  $n, d > 0, q \geq 2$ , and a distribution  $\chi$  over  $\mathbb{R}^{<d}[x]$ . The (decision)  $\text{MP-LWE}_{n,d,q,\chi}$  consists in distinguishing between arbitrarily many samples from  $\text{MP}_{q,n,d,\chi}(s)$  and the same number of samples from  $U(\mathbb{Z}_q^{<n}[x] \times \mathbb{R}_q^{<d}[x])$ , with non-negligible probability over the choices of  $s \leftarrow U(\mathbb{Z}_q^{<n+d-1}[x])$ .

**Definition 7.** (MP-CLWR assumption)

Let  $d, n, p, q$  and  $t$  be positive integers fulfilling  $0 < d \leq n$  and  $q \geq p \geq 2$ . Choose

$s$  uniformly at random over  $(\mathbb{Z}_q^{<n+d-1}[x])^\times$ . Denote by  $\chi_s$  the distribution of  $(a, [a \odot_d s]_p)$ , where  $a \leftarrow U(\mathbb{Z}_q^{<n}[x])$ , and denote by  $\mathcal{U}$  the distribution of  $(a, [b]_p)$ , where  $a \leftarrow U(\mathbb{Z}_q^{<n}[x])$  and  $b \leftarrow U(\mathbb{Z}_q^{<d}[x])$ . For  $i \in \{1, 2\}$  define the input for  $S_i$  as  $(var_i, con)$ , where  $var_1$  denotes the distribution  $\chi_s^t$ , and  $var_2$  the distribution  $\mathcal{U}^t$ , and  $con$  is an arbitrary distribution over  $\{0, 1\}^*$  which is independent from  $var_1$  and  $var_2$ . For a fixed challenger  $\mathcal{C}$  let  $\mathcal{P}_{\mathcal{C}, \mathcal{A}}$  be the probability for an adversary  $\mathcal{A}$  to win  $Exp_1(\mathcal{C}, \mathcal{A}, \mathcal{S}_1)$ , while  $\mathcal{Q}_{\mathcal{C}, \mathcal{A}}$  be that for  $\mathcal{A}$  to win  $Exp_2(\mathcal{C}, \mathcal{A}, \mathcal{S}_2)$ .

**Definition 8.** Let  $l, r_a, r_b$  be the integers. The middle inner product  $\langle a, b \rangle_{(l, r_a, r_b)}^{\odot_d} : R_a^{l+<d}[x] \times R_b^{l+<d}[x] \rightarrow R^{l+<d}[x]$  is the function such that  $a, b$  do standard inner product for first  $l$  entries of  $a, b$ , and the remainders do middle product for  $r_a$  each rows of remainder  $a$  products  $r_b$  each rows of remainder  $b$ .



(The first  $l$  entries compute standard inner product and plus  $r_{a1} \odot_d r_{b2} + \dots + r_{ak} \odot_d r_{bk}$ , where  $l = k \times h$ )

Notice that we use the notation  $\langle a, b \rangle^{\odot_d}$  to denote  $\langle a, b \rangle_{(d, t, <n+d+k-1)}^{\odot_d}$





## Chapter 3 Our Construction

We describe each basic encryption scheme on section 3.1 to section 3.3 and described the key switching for each scheme in section 3.4 and section 3.5. In section 3.6 we lay out our FHE scheme for each encryption scheme.

### 3.1 Basic LWR (RLWR) encryption scheme

In this section, our encryption scheme do not need to generate the error terms and we use the hardness LWR (RLWR) instead. Basic encryption scheme:

- E.Setup: Choose  $d = d(\lambda, \mu, b)$ ,  $n = n(\lambda, \mu, b)$ ,  $\chi = \chi(\lambda, \mu, b)$   $N = \lceil (2n + 3) \log q \rceil$  same as the [11] scheme. And let  $R = \mathbb{Z}[x]/(x^d + 1)$
- E.SecretKeyGen:  $s' \leftarrow \chi^n$ .  $sk = \mathbf{s} = (1, s'[1], s'[2], \dots, s'[n])$ .  $\mathbf{s} \in R_q^{n+1}$
- E.PublicKeyGen: Generate matrix  $\mathbf{A}' \in R_q^{N \times n}$  and set  $\mathbf{b} = 2^2 \times \lceil \frac{1}{2} \mathbf{A}' s' \rceil - (2 - 1) \mathbf{A}' s' \in R_q$ . Set  $\mathbf{A} = [\mathbf{b} \mid -\mathbf{A}']$ , notice that  $\langle \mathbf{A}, \mathbf{s} \rangle = 2^2 (\lceil \frac{1}{2} \mathbf{A}' s' \rceil - \frac{1}{2} \mathbf{A}' s')$
- E.Encryption: To encryption a message  $m \in R_2$ . Set  $\mathbf{m} = (m, 0, \dots, 0) \in R_2^{n+1}$ , random  $\mathbf{r} \in R_2^N$  and output the ciphertext  $\mathbf{c} = \mathbf{m} + \mathbf{A}^T \mathbf{r}$
- E.Decryption: Output  $m = \langle \mathbf{c}, \mathbf{s} \rangle_q \pmod{2}$



### 3.2 Basic MPLWE encryption scheme

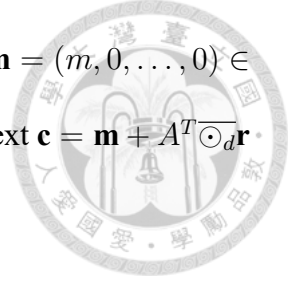
In this section, we adjust the encryption scheme describe in [9] and compatible to our homomorphic encryption scheme. Let  $\chi = \lfloor D_{\alpha q} \rfloor$  denote the distribution over  $\mathbb{Z}^{<d+k}[x]$  where each coefficient is sampled from  $D_{\alpha q}$  and then rounded to nearest integer. And let  $t \geq 2$ .

- E.KeyGen: Random  $s' \leftarrow U(\mathbb{Z}_q^{<n+d+k-1}[x])$ . For every  $i \leq t$ , random  $a'_i \leftarrow U(\mathbb{Z}_q^{<n}[x])$ ,  $e_i \leftarrow \chi$  and compute  $b_i = a'_i \odot_{d+k} s' + 2e_i \in \mathbb{Z}_q^{<d+k}[x]$ . The secret key  $sk = \mathbf{s} = (1, s') \in \mathbb{Z}_q^{<n+2d+k-1}[x]$  and the public key  $pk = (a'_i, b_i)_{i \leq t}$ . Set  $\mathbf{A} = [\mathbf{b} \mid -\mathbf{A}']$
- E.Encryption: To encryption a message  $m \in \{0, 1\}^{<d}[x]$ . Set  $\mathbf{m} = (m, 0, \dots, 0) \in \mathbb{Z}^{<d+t}[x]$ , random  $\mathbf{r}_i \leftarrow U(\{0, 1\}^{k+1}[x])$  and output the ciphertext  $\mathbf{c} = \mathbf{m} + A^T \overline{\odot}_d \mathbf{r}$
- E.Decryption: Output  $m = \langle \mathbf{c}, \mathbf{s} \rangle_q^{\odot d} \pmod 2$

### 3.3 Basic MPLWR encryption scheme

In this section, we lay out our encryption scheme of MPLWR with the hardness assumption describe in [10]. Let  $\chi = \lfloor D_{\alpha q} \rfloor$  denote the distribution over  $\mathbb{Z}^{<d+k}[x]$  where each coefficient is sampled from  $D_{\alpha q}$  and then rounded to nearest integer. And let  $t \geq 3$ .

- E.KeyGen: Random  $s' \leftarrow U(\mathbb{Z}_q^{<n+d+k-1}[x])$ . For every  $i \leq t$ , random  $a'_i \leftarrow U(\mathbb{Z}_q^{<n}[x])$ ,  $e_i \leftarrow \chi$  and compute  $b_i = 4[a'_i \odot_{d+k} s']_2 - a'_i \odot_{d+k} s' \in \mathbb{Z}_q^{d+k}[x]$ . The secret key  $sk = \mathbf{s} = (1, s') \in \mathbb{Z}_q^{<n+2d+k-1}[x]$  and the public key  $pk = (a'_i, b_i)_{i \leq t}$ . Set  $\mathbf{A} = [\mathbf{b} \mid -\mathbf{A}']$



- E.Encryption: To encryption a message  $m \in \{0, 1\}^{<d[x]}$ . Set  $\mathbf{m} = (m, 0, \dots, 0) \in \mathbb{Z}^{<d+t[x]}$ , random  $\mathbf{r}_i \leftarrow U(\{0, 1\}^{k+1[x]})$  and output the ciphertext  $\mathbf{c} = \mathbf{m} + A^T \odot_d \mathbf{r}$ .
- E.Decryption: Output  $m = \langle \mathbf{c}, \mathbf{s} \rangle_q^{\odot_d} \pmod 2$

### 3.4 Key Switching for MPLWE and MPLWR based

FHE is a scheme which supports both additions and multiplications on ciphertexts. It is obviously that the addition property is barely not support, so we focus on how to satisfy multiplication property. Kroncker product is a heuristic way in our scheme. That is, to compute  $m_1 \times m_2$ , we compute  $\langle \mathbf{c}_1 \otimes \mathbf{c}_2, \mathbf{s} \otimes \mathbf{s} \rangle$ . However, if the levels of the multiplication are too depth, the noise will become huge. In [11], its solution is the switch key method, which refreshes the ciphertexts to fix long. The scheme is evidently compatible to our LWE (RLWE) scheme. And our work is to prove that it can be compatible to MPLWE and MPLWR scheme. We use the notation denote in [11].

- BitDecomp ( $\mathbf{x} \in \mathbb{Z}_q^n[x], q$ ) decomposes  $\mathbf{x}$  into its bit representation for each entries. Namely, write  $\mathbf{x} = \sum_{j=0}^{\lceil \log q \rceil} 2^j \cdot \mathbf{u}_j$ , where all of the  $\mathbf{u}_j \in \{0, 1\}^n[x]$ , and output  $(\mathbf{u}_0, \dots, \mathbf{u}_{\lceil \log q \rceil}) \in \mathbb{Z}_2^{\lceil \log q \rceil \times n}[x]$
- Powerof2 ( $\mathbf{x} \in \mathbb{Z}_q^m[x], q$ ) Output  $(\mathbf{x}, 2 \cdot \mathbf{x}, \dots, 2^{\lceil \log q \rceil} \cdot \mathbf{x}) \in \mathbb{Z}_q^{\lceil \log q \rceil \times m}[x]$

**Lemma 1.** We have  $\langle \text{BitDecomp}(\mathbf{c}, q), \text{Powerof2}(\mathbf{s}, q) \rangle_{(\lceil \log q \rceil, t, n+d+k-1)}^{\odot_d} = \langle \mathbf{c}, \mathbf{s} \rangle_q^{\odot_d} \pmod q$  for vectors  $\mathbf{c} \in \mathbb{Z}_q^{d+t}$ ,  $\mathbf{s} \in \mathbb{Z}_q^{n+2d+k-1}$

*Proof.* By lemma, we know that we can add first and then compute the middle product, hence if we see the BitDecomp( $\mathbf{c}, q$ ) and Powerof2( $\mathbf{s}, q$ ) as extending  $\mathbf{c}, \mathbf{s}$  into bit compu-

tations then it is correct. Hence

$$\begin{aligned}
\langle \mathbf{c}, \mathbf{s} \rangle^{\odot_d} &= \left\langle \sum_{j=0}^{\lceil \log q \rceil} 2^j \cdot \mathbf{u}_j, \mathbf{s} \right\rangle^{\odot_d} \\
&= \sum_{j=0}^{\lceil \log q \rceil} \langle 2^j \cdot \mathbf{u}_j, \mathbf{s} \rangle^{\odot_d} \\
&= \sum_{j=0}^{\lceil \log q \rceil} \langle \mathbf{u}_j, 2^j \cdot \mathbf{s} \rangle^{\odot_d} \\
&= \langle \text{BitDecomp}(\mathbf{c}, q), \text{Powerof2}(\mathbf{s}, q) \rangle_{(d \lceil \log q \rceil, t, n+d+k-1)}^{\odot_d}
\end{aligned}$$



□

By the proof of our main lemma we can now introduce how to refresh the ciphertexts into fixed long.

SwitchKeyGen( $s_1 \in \mathbb{Z}_q^{n_1}[x], s_2 \in \mathbb{Z}_q^{n_2}[x]$ ):

1. Run  $\mathbf{A} \leftarrow \text{E.PublicKeyGen}(s_2, N)$  for  $N = n_1 \cdot \lceil \log q \rceil$ .
2. Set  $\mathbf{B} \leftarrow \mathbf{A} + \text{Powerof2}(s_1)$  (add it to  $\mathbf{A}$ 's first column). Output  $\tau_{s_1 \rightarrow s_2} = \mathbf{B}$

SwitchKey( $\tau_{s_1 \rightarrow s_2}, c_1$ ): Output  $c_2 = \text{BitDecomp}(c_1)^T \overline{\odot_d} \mathbf{B} \in \mathbb{Z}_q^{n_2}[x]$

**Lemma 2.** Let  $\mathbf{s}_1, \mathbf{s}_2, q, n_1, n_2, \mathbf{A}, \mathbf{B} = \tau_{s_1 \rightarrow s_2}$  be as in SwitchKeyGen( $\mathbf{s}_1, \mathbf{s}_2$ ), and let  $\mathbf{A} \overline{\odot_d} \mathbf{s}_2 = 2\mathbf{e}_2 \in \mathbb{Z}_q^{n_2}[x]$  (For MPLWR we have  $\mathbf{A} \overline{\odot_d} \mathbf{s}_2 = 2(2 \lceil a'_i \odot_d s'_i \rceil_2 - a'_i \odot_d s'_i)$ ). Let  $\mathbf{c}_1 \in \mathbb{Z}_q^{n_1}[x]$  and  $\mathbf{c}_2 \leftarrow \text{SwitchKey}(\tau_{s_1 \rightarrow s_2}, \mathbf{c}_1)$ . Then,

$$\langle \mathbf{c}_2, \mathbf{s}_2 \rangle^{\odot_d} = 2 \langle \text{BitDecomp}(\mathbf{c}_1), \mathbf{e}_2 \rangle_{(\lceil \log q \rceil, n_1-d, n_2-d)}^{\odot_d} + \langle \mathbf{c}_1, \mathbf{s}_1 \rangle^{\odot_d}$$





*Proof.*

$$\begin{aligned}
\langle \mathbf{c}_2, \mathbf{s}_2 \rangle &= \text{BitDecomp}(\mathbf{c}_1) \odot_d \mathbf{B} \odot_d \mathbf{s}_1 \\
&= \text{BitDecomp}(\mathbf{c}_1) \odot_d (2\mathbf{e}_2 + \text{Powerof2}(\mathbf{s}_1)) \\
&= 2 \langle \text{BitDecomp}(\mathbf{c}_1), \mathbf{e}_2 \rangle_{([\log q], n_1-d, n_2-d)}^{\odot_d} + \langle \text{BitDecomp}(\mathbf{c}_1), \text{Powerof2}(\mathbf{s}_1) \rangle_{([\log q], n_1-d, n_2-d)}^{\odot_d} \\
&= 2 \langle \text{BitDecomp}(\mathbf{c}_1), \mathbf{e}_2 \rangle_{([\log q], n_1-d, n_2-d)}^{\odot_d} + \langle \mathbf{c}_1, \mathbf{s}_1 \rangle^{\odot_d}
\end{aligned}$$

□

### 3.5 Key Switching for LWR(RLWR) based

In this section we will provide the LWR(RLWR) based key switching. Since it is an equivalent modification from LWE(RLWE) based, we will find it similar to it. We will use the notation denoted in [11].

- $\text{BitDecomp}(\mathbf{x} \in R_q^n, q)$  decomposes  $\mathbf{x}$  into its bit representation for each entries.

Namely, write  $\mathbf{x} = \sum_{j=0}^{[\log q]} 2^j \cdot \mathbf{u}_j$ , where all of the  $\mathbf{u}_j \in R_2^n$ , and output  $(\mathbf{u}_0, \dots, \mathbf{u}_{[\log q]}) \in R_2^{[\log q] \times n}$

- $\text{Powerof2}(\mathbf{x} \in R_q^m, q)$  Output  $(\mathbf{x}, 2 \cdot \mathbf{x}, \dots, 2^{[\log q]} \cdot \mathbf{x}) \in R_q^{[\log q] \times m}$

**Lemma 3.** For vectors  $\mathbf{c}, \mathbf{s}$  of equal length, we have  $\langle \text{BitDecomp}(\mathbf{c}, q), \text{Powerof2}(\mathbf{s}, q) \rangle = \langle \mathbf{c}, \mathbf{s} \rangle \pmod q$ .



*Proof.*

$$\begin{aligned}
\langle \mathbf{c}, \mathbf{s} \rangle &= \left\langle \sum_{j=0}^{\lceil \log q \rceil} 2^j \cdot \mathbf{u}_j, \mathbf{s} \right\rangle \\
&= \sum_{j=0}^{\lceil \log q \rceil} \langle 2^j \cdot \mathbf{u}_j, \mathbf{s} \rangle \\
&= \sum_{j=0}^{\lceil \log q \rceil} \langle \mathbf{u}_j, 2^j \cdot \mathbf{s} \rangle \\
&= \langle \mathbf{c}, \mathbf{s} \rangle \langle \text{BitDecomp}(\mathbf{c}, q), \text{Powerof2}(\mathbf{s}, q) \rangle
\end{aligned}$$

□

And hence we have

SwitchKeyGen( $s_1 \in R_q^{n_1}, s_2 \in R_q^{n_2}$ ):

1. Run  $\mathbf{A} \leftarrow \text{E.PublicKeyGen}(s_2, N)$  for  $N = n_1 \cdot \lceil \log q \rceil$ .
2. Set  $\mathbf{B} \leftarrow \mathbf{A} + \text{Powerof2}(s_1)$  (add it to  $\mathbf{A}$ 's first column). Output  $\tau_{s_1 \rightarrow s_2} = \mathbf{B}$

SwitchKey( $\tau_{s_1 \rightarrow s_2}, c_1$ ): Output  $c_2 = \text{BitDecomp}(c_1)^T \cdot \mathbf{B} \in R_q^{n_2}$

**Lemma 4.** Let  $\mathbf{s}_1, \mathbf{s}_2, q, n_1, n_2, \mathbf{A}, \mathbf{B} = \tau_{s_1 \rightarrow s_2}$  be as in SwitchKeyGen( $\mathbf{s}_1, \mathbf{s}_2$ ), and let  $\mathbf{A} \cdot$

$\mathbf{s}_2 = 2^2(\lceil \frac{1}{2} \mathbf{A}' s' \rceil - \frac{1}{2} \mathbf{A}' s') \in R_q^N$ . Let  $\mathbf{c}_1 \in R_q^{n_1}$  and  $\mathbf{c}_2 \leftarrow \text{SwitchKey}(\tau_{s_1 \rightarrow s_2}, \mathbf{c}_1)$ . Then,

$$\langle \mathbf{c}_2, \mathbf{s}_2 \rangle = 2 \langle \text{BitDecomp}(\mathbf{c}_1), 2(\lceil \frac{1}{2} \mathbf{A}' s' \rceil - \frac{1}{2} \mathbf{A}' s') \rangle + \langle \mathbf{c}_1, \mathbf{s}_1 \rangle \pmod q$$

*Proof.*

$$\begin{aligned}
\langle \mathbf{c}_2, \mathbf{s}_2 \rangle &= \text{BitDecomp}(\mathbf{c}_1)^T \cdot \mathbf{B} \cdot \mathbf{s}_2 \\
&= \text{BitDecomp}(\mathbf{c}_1)^T \cdot (2^2(\lceil \frac{1}{2} \mathbf{A}' s' \rceil - \frac{1}{2} \mathbf{A}' s') + \text{Powerof2}(\mathbf{s}_1)) \\
&= 2 \langle \text{BitDecomp}(\mathbf{c}_1), 2(\lceil \frac{1}{2} \mathbf{A}' s' \rceil - \frac{1}{2} \mathbf{A}' s') \rangle + \langle \text{BitDecomp}(\mathbf{c}_1), \text{Powerof2}(\mathbf{s}_1) \rangle \\
&= 2 \langle \text{BitDecomp}(\mathbf{c}_1), 2(\lceil \frac{1}{2} \mathbf{A}' s' \rceil - \frac{1}{2} \mathbf{A}' s') \rangle + \langle \mathbf{c}_1, \mathbf{s}_1 \rangle
\end{aligned}$$



### 3.6 FHE scheme

The previous lemma provide us compatible to [11] scheme. In this section, we will present our scheme modify from [11].(Notation:  $L_c(\mathbf{x}) = \langle \mathbf{c}, \mathbf{x} \rangle^{\odot q}$  is a ciphertext-dependent linear equation over the coefficients of  $\mathbf{x}$ , and a linear equation  $L_{c_1, c_2}^{long}(\mathbf{x} \otimes \mathbf{x})$  is a linear equation over the coefficient over the coefficients of  $\mathbf{x} \otimes \mathbf{x}$  )

- FHE.KenGen. For  $j = L \rightarrow 0$  do
  1. Generation  $s_j$  and  $\mathbf{A}_j$  for each encryption scheme.
  2. Set  $s'_j = s_j \otimes s_j$  kronecker tensor in here.
  3. Set  $s''_j = \text{BitDecomp}(s'_j, q_j)$
  4. Run  $\tau_{s''_j \rightarrow s_j} = \text{SwitchKeyGen}(s''_j, s_{j-1})$  (Omit this step in the beginning i.e.  $j = L$ )
- FHE.Enc. Basic encryption scheme to encrypt messages.
- FHE.Dec. Suppose the ciphertext is under key  $s_j$ . Decrypt the message under key  $s_j$  in E.Decryption
- FHE.Add. Take two ciphertexts encrypt under key  $s_j$  (if not, do FHE.Refresh to make it encrypts under same key  $s_j$ ). Set  $\mathbf{c}_3 = \mathbf{c}_1 + \mathbf{c}_2 \pmod{q_j}$ . (In [11], it expands the ciphertexts size to make it indistinguishable to FHE.Mult) Hence we interpret  $\mathbf{c}_3$  under  $s'_j$  and output

$$\mathbf{c}_4 = \text{FHE.Refresh}(\mathbf{c}_3, \tau_{s''_j \rightarrow s_{j-1}}, q_j, q_{j-1})$$

- FHE.Mult. Take two ciphertexts encrypt under key  $s_j$  (if not, do FHE.Refresh to make it encrypts under same key  $s_j$ ). The new ciphertexts is the kronecker tensor of two ciphertexts, with key  $s'_j = s_j \otimes s_j$ , and store it into a line, i.e.  $\mathbf{c}_3 = L_{c_1, c_2}^{long}(\mathbf{x} \otimes \mathbf{x})$  and output

$$\mathbf{c}_4 = \text{FHE.Refresh}(\mathbf{c}_3, \tau_{s''_j \rightarrow s_{j-1}}, \mathbf{q}_j, \mathbf{q}_{j-1})$$

- FHE.Refresh. Takes a ciphertext encrypted under  $s'_j$ , the auxiliary information  $\tau_{s''_j \rightarrow s_{j-1}}$  to facilitate key switching, and the current and next modulo  $q_j$  and  $q_{j-1}$ . Do the following:
  1. Expand: Set  $\mathbf{c}_1 = \text{Powerof2}(\mathbf{c}, q_j)$ .
  2. Switch Moduli: Set  $\mathbf{c}_2 = \text{Scale}(\mathbf{c}_1, q_j, q_{j-1}, 2)$ , a ciphertext under the key  $s''_j$  for modulus  $q_{j-1}$ .
  3. Switch Keys: Output  $\mathbf{c}_3 = \text{SwitchKey}(\tau_{s''_j \rightarrow s_{j-1}}, \mathbf{c}_2, q_{j-1})$ , a ciphertext under the key  $s_{j-1}$  for modulus  $q_{j-1}$ .

Since we have expressed our notation compatible to [11] it may look similar to its scheme. But, it actually does different things in MPLWE and MPLWR schemes. And for the LWR(RLWR) scheme, it can seem to be an equivalent modification to LWE(RLWE) scheme. Hence, we can use initial LWE(RLWE) scheme to make them into FHE. Since it looks almost the same, so we do not emphasis it particular.



## Chapter 4 Correctness

### 4.1 Correctness of LWR (RLWR) scheme

**Lemma 5** (Correctness). Let  $\mathbf{c}, \mathbf{A}, \mathbf{r}$  be described in the Encryption scheme of LWR (RLWR), then we can decrypt the message  $m \in \{0, 1\}$  using the secret key  $\mathbf{s}$

*Proof.*

$$\begin{aligned} m &= \langle \mathbf{c}, \mathbf{s} \rangle_q \pmod 2 = \langle \mathbf{m} + \mathbf{A}^T \mathbf{r}, \mathbf{s} \rangle = \langle \mathbf{m}, \mathbf{s} \rangle + \langle \mathbf{A}^T \mathbf{r}, \mathbf{s} \rangle \\ &= m + 2 \times 2 \times \left( \left\lceil \frac{1}{2} \mathbf{A}' \mathbf{s} \right\rceil - \frac{1}{2} \mathbf{A}' \mathbf{s} \right) r \pmod 2 \\ &= m \end{aligned}$$

□

**Lemma 6** (homomorphic properties). Let  $c_1, c_2$  be two different messages encrypt by  $\mathbf{s}$  and  $\mathbf{A}, \mathbf{r}$  be described in Encryption scheme of LWR (RLWR), then we can decrypt the message  $m_1 + m_2$  and  $m_1 \times m_2$  using the secret key  $\mathbf{s}$

*Proof.*

$$\begin{aligned} m_1 + m_2 &= \langle \mathbf{c}_1 + \mathbf{c}_2, \mathbf{s} \rangle_q \pmod 2 = \langle \mathbf{c}_1, \mathbf{s} \rangle_q + \langle \mathbf{c}_2, \mathbf{s} \rangle_q \pmod 2 \\ &= m_1 + m_2 \end{aligned}$$

$$\begin{aligned}
m_1 \times m_2 &= \langle \mathbf{c}_1 \otimes \mathbf{c}_2, \mathbf{s} \otimes \mathbf{s} \rangle = \langle (\mathbf{m}_1 + \mathbf{A}^T r_1) \otimes (\mathbf{m}_2 + \mathbf{A}^T r_2), \mathbf{s} \otimes \mathbf{s} \rangle \\
&= \langle \mathbf{m}_1 \otimes \mathbf{m}_2, \mathbf{s} \otimes \mathbf{s} \rangle + \langle \mathbf{A}^T r_1 \otimes \mathbf{m}_2, \mathbf{s} \otimes \mathbf{s} \rangle \\
&\quad + \langle m_1 \otimes \mathbf{A}^T r_2, \mathbf{s} \otimes \mathbf{s} \rangle + \langle \mathbf{A}^T r_1 \otimes \mathbf{A}^T r_2, \mathbf{s} \otimes \mathbf{s} \rangle \\
&= m_1 \times m_2
\end{aligned}$$



□

## 4.2 Correctness of MPLWE(MPLWR) scheme

**Lemma 7** (Correctness). Assume that  $\alpha < 1/(16\sqrt{\lambda tk})$  and  $q \geq 16t(k+1)$ . With probability  $\geq 1 - d \cdot 2^{-\Omega(\lambda)}$  over the randomness of  $(sk, pk) \leftarrow KeyGen$ , for all plaintext  $\mu$  and with probability 1 over the randomness of  $Encrypt$ , we have  $Decrypt(sk, Encrypt(pk, \mu)) = \mu$

*Proof.* In [9] **Lemma 4.1**.

**Lemma 8** (homomorphic properties). Let  $c_1, c_2$  be two different messages encrypt by  $\mathbf{s}$  and  $\mathbf{A}, \mathbf{r}$  be described in Encryption scheme of MPLWE (MPLWR), then we can decrypt the message  $m_1 + m_2$  and  $m_1 \times m_2$  using the secrete key  $\mathbf{s}$

*Proof.*

$$\begin{aligned}
m_1 + m_2 &= \langle \mathbf{c}_1 + \mathbf{c}_2, \mathbf{s} \rangle_q^{\odot d} = \langle \mathbf{c}_1, \mathbf{s} \rangle_q^{\odot d} + \langle \mathbf{c}_2, \mathbf{s} \rangle_q^{\odot d} \\
&= m_1 + m_2
\end{aligned}$$

$$\begin{aligned}
m_1 \times m_2 &= \langle \mathbf{c}_1 \otimes \mathbf{c}_2, \mathbf{s} \otimes \mathbf{s} \rangle_q^{\odot d} = \langle (\mathbf{m}_1 + \mathbf{A}^T r_1)_q^{\odot d} \otimes (\mathbf{m}_2 + \mathbf{A}^T r_2)_q^{\odot d}, \mathbf{s} \otimes \mathbf{s} \rangle_q^{\odot d} \\
&= \langle \mathbf{m}_1 \otimes \mathbf{m}_2, \mathbf{s} \otimes \mathbf{s} \rangle_q^{\odot d} + \langle \mathbf{A}^T r_1 \otimes \mathbf{m}_2, \mathbf{s} \otimes \mathbf{s} \rangle_q^{\odot d} \\
&\quad + \langle \mathbf{m}_1 \otimes \mathbf{A}^T r_2, \mathbf{s} \otimes \mathbf{s} \rangle_q^{\odot d} + \langle \mathbf{A}^T r_1 \otimes \mathbf{A}^T r_2, \mathbf{s} \otimes \mathbf{s} \rangle_q^{\odot d} \\
&= m_1 \times m_2
\end{aligned}$$



□







# Chapter 5 Optimization

## 5.1 Bootstrapping and Batching

A somewhat homomorphic encryption scheme is a scheme which contains addition and multiple properties at the same time (Roughly speaking). In [4], Gentry has proved that there exist an efficient transformation that given a description of a bootstrappable scheme  $\varepsilon$  and a parameter  $d = d(\lambda)$  outputs a description of another encryption scheme such that  $\varepsilon^{(d)}$  is compact (which means the size of the ciphertext is bound) and  $\varepsilon^{(d)}$  is homomorphic for all circuits of depth up to  $d$ .

Our scheme is obviously a somewhat homomorphic scheme, i.e. we still can make our scheme into bootstrappable. The advantage to make our scheme into bootstrappable is that in [5] it has described a way to batch the bootstrapping scheme and have high efficient in specific problems.

## 5.2 Public Key Compression for LWR

In [6], introduce a way to compress the public key size with a pseudo-random number generator  $f$ . However, it is not compatible to polynomial ring based encryption, hence we only introduce the public key compression for LWR.

- $\text{KeyGen}(1^\lambda)$  Generate a random prime integer  $p$  of size  $\eta$  bits. And randomly generate  $\mathbf{a}_i$ 's, compute  $b_i = 4\lceil \frac{1}{2}a_i s \rceil - a_i s$ . Initialize a pseudo-random number generator  $f$  with a random seed  $se$ . Use  $f(se)$  to generate a set of integers  $\chi_i \in [0, 2^\gamma)$  for  $1 \leq i \leq \tau$ . For all  $1 \leq i \leq \tau$  compute:

$$\delta_i = \langle \chi_i \rangle_p + \xi \cdot p - r_i$$

where  $r_i \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$  and  $\xi_i \leftarrow \mathbb{Z} \cap [0, 2^{\lambda+\eta}/p]$ . For all  $1 \leq i \leq \tau$  compute:

$$b_i = \chi_i - \delta_i$$

Let  $pk = (a_0, \dots, a_\tau, se, \delta_0, \dots, \delta_\tau)$  and  $sk = p$

We should store all of  $a_i$  and about a one dimension terms  $b_i$ 's, i.e. we store about  $\tau \cdot \eta + \gamma + \eta$  bits public key instead of  $2\tau\eta$  public key, which is about halves the initial public key, but notice that we should calculate the public key each time we need to use.



## Chapter 6 Zero knowledge proof

In [3], it introduces a way to do Zero knowledge proof via fully homomorphic encryption. It is still compatible to our scheme. The generic protocol, between a prover  $P$  and a verifier  $V$ , is as follows.

$P_1$ . Choose an encryption  $c' = b' + r'$  of zero and send  $c'$  to the verifier.

$V_1$ . Select  $e \leftarrow \{0, 1\}$  and send  $e$  to the prover.

$P_2$ . If  $e = 0$ , set  $d = b'$ , or if  $e=1$ , set  $d = b + b'$ . Transmit  $d$ .

$V_2$ . Verify that  $d$  is a lattice point, and check that the noise  $ec + c' - d$  is well-formed and sufficiently small.

This is also our advantage to choose lattice based encryption. It is easily to compatible our scheme to other lattice based protocol.





## Chapter 7 Application

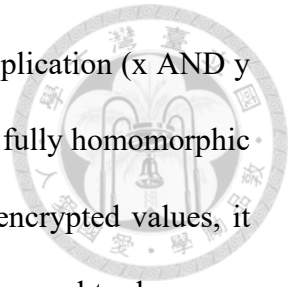
There are many situations we will likely to use fully homomorphic encryption. For example, machine learning may need a huge amounts of computing. However, it may be a hard time for a start-up company to buy high-performance computers. The solutions to this situation are to rend computing power via cloud computing. However, how to save data security ? Hence, we can save our security via fully homomorphic encryption. Since we have addition properties and multiplication properties. We can do all kinds of computing in encrypted state.

Although we may spend more computing resource to keep the data security, it is still pay-off if the total spending time is fewer than using personal computer. Hence fully homomorphic encryption may be a good choice to keep data safe and save more times than usual.

Otherwise, fully homomorphic encryption may also be a good choice to do multi-party computation. Since we have addition properties and multiplication properties we can easily construct a scheme.

If you want to do secure multi-party computation, you can express the computation as a boolean circuit  $C$ , and you can easily transform any circuit so that it uses only AND gates and NOT gates. Then, it turns out that you can compute  $C$  on encrypted data, if the data was encrypted using a fully homomorphic encryption scheme, using the follow-

ing relationship: when working with 0,1, AND can be done by multiplication ( $x \text{ AND } y = xy$ ), and NOT can be done with addition ( $\text{NOT}(x) = 1-x$ ). Since the fully homomorphic encryption lets you do addition, subtraction, and multiplication on encrypted values, it also lets you do NOT and AND on encrypted values, which is all you need to do secure multi-party computation.





## Chapter 8 Summary

We introduce a way to do FHE on LWR, MPLWE and MPLWR which have higher security than basic LWE scheme. And it seems that the variants of LWE have the similar properties. And we also introduce the public key compression, which is useful when the transfer costs may be high in certain case.

The table below show the different between these protocol. The estimate time showed in the table is calculated by the numbers of multiplication. We consider that we calculate 1 GB messages AES-128 with 3.60GHz. And the multiplication in AES-128 contains 7568 multiplications over finite field. The finite field multiplication algorithm we use is the Montgomery modular multiplication with each piece 32 bits. The estimated performance times is about 484 times AES-128 computing times (about 14.48 (s) for 1GB AES-128).

	LWE	LWR	MPLWE	MPLWR
Public key	$\lceil (2n + 3) \log q \rceil$	$\lceil (2n + 3) \log q \rceil$	$(n + d + k - 1)t$	$(n + d + k - 1)t$
Secret key	$(n + 1) \log q$	$(n + 1) \log q$	$(n + d + k - 1) \log q$	$(n + d + k - 1) \log q$
Ciphertext	$(n + 1) \log q$	$(n + 1) \log q$	$(n + k) \log q + d \log q$	$(n + k) \log q + d + k$
Estimate time	1.953 (h)	1.954 (h)	2.198 (h)	2.210 (h)
Hardness	$LWE \leq LWR, LWE \leq MPLWE \leq MPLWR$			







## Chapter 9 Future Work


In 2018, the most simple fully homomorphic encryption scheme DGHV is said to be broken via quantum computer in quantum polynomial times. For our scheme, we still do not have a security proof to withstand quantum computer. We hope that we can prove the quantum security to our scheme. Otherwise, it seems that we have to transmit more information on computing, if there is some transmitting error the message will be to break. We hope that we can solve this problem for example changing the scheme for code based encryption.





## References

- [1] C. P. Abhishek Banerjee and A. Rosen. Pseudorandom functions and lattices. 26, 2011.
- [2] E. T. Adriana Lopez-Alt and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. 70, 2013.
- [3] G. T. D. K. G. Christopher Carr, Anamaria Costache and M. Strand. Zero-knowledge proof of decryption for the ciphertexts. 16:16, 2018.
- [4] C. Gentry. Fully homomorphic encryption using ideal lattices. 28:169–178, 2009.
- [5] T. L. Jean-Sebastien Coron and M. Tibouchi. Batch fully homomorphic encryption over the integers. 27, 2013.
- [6] D. N. Jean-Sébastien Coron and M. Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. 27, 2011.
- [7] K. P. Joel Alwen, Stephan Krenn and D. Wichs. Learning with rounding, revisited. Annual Cryptology Conference, 18:57–74, 2013.
- [8] Z. Z. Long Chen and Z. Zhang. On the hardness of the computational ring-lwr problem and its applications. 33, 2018.

- 
- [9] D. S. Miruna Rosca, Amin Sakzad and R. Steinfeld. Middle-product learning with errors. 17, 2017.
- [10] D. D. A. R.-L. W. W. Shi Bai, Katharina Boudgoust and Z. Zhang. Middle-product learning with rounding problem and its applications. 32, 2019.
- [11] C. G. Zvika Brakerski and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. 26, 2011.