Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Fakultät für Ingenieurwissenschaften

Master-Thesis

Homomorphic Post-Quantum Cryptography - Evaluation of Module Learning with Error in Homomorphic Cryptography

Date: October 29, 2024

Author: Pascal Stehling
E-Mail: pascal@stehl.ing

Abstract

This thesis investigates the conversion of Ring-LWE (R-LWE)-based homomorphic encryption schemes to Module-LWE (M-LWE) and analyses the resulting performance differences. The advantage of M-LWE is that a fixed-sized polynomial degree can be utilized and the security of the system can be changed by increasing the vector/matrix dimension. This is the same concept that is utilized in the CRYSTALS-Kyber encryption scheme. The feasibility of transferring R-LWE to M-LWE is demonstrated based on the BFV homomorphic encryption scheme, showing that a functioning homomorphic encryption can be maintained. While the addition is straightforward, the multiplication necessitates the generation of multiple relinearization (evaluation) keys. It is demonstrated that the practical performance is only slightly inferior to that of R-LWE, with the advantage of smaller ciphertext sizes. However, there is still considerable scope for improvement in theoretical aspects, such as the study of security benefits and in practice, in enhancing the general performance.

Contents

1 Introduction

In the early months of 1978, one of the most significant cryptographic systems, the RSA system [30], was published. With the advent of the Internet in the 1990s and the subsequent need for secure data transfer, it became one of the most widely used encryption schemes to date. In the subsequent period of slightly more than half a year after publishing the RSA algorithm, two of its authors published a new concept based on RSA which they called *privacy homomorphism* [29]. This concept would later be known as Homomorphic Encryption (HE). It's an encryption system whereby operations can be executed directly on encrypted data, eliminating the necessity of first decrypting it, running the operations, and then encrypting it again. Such a system would not only eliminate the necessity for decryption and encryption at the processing stage, it would also ensure that the plain text is not readable by the party undertaking this processing. However, at the system's inception, only one operation was feasible: multiplication. To develop a system capable of general computing, the addition operation was necessary as a second operation. With these two operations, all other operations can be constructed at the bit level, by creating a logical NAND gate. Unfortunately, the creation of a homomorphic encryption scheme with unlimited additions and multiplications, also known as full homomorphic encryption (FHE), proved to be a formidable challenge.

In 1994, Peter Shor published his algorithm [31], which describes how a quantum computer could factorize numbers in polynomial time. This is in contrast to classical computers, for which this problem is categorized as a hard problem, which means exponential time is necessary to solve the problem. Given that the RSA cryptosystem is based on the assumption that this particular mathematical problem is difficult to compute, it is theoretically possible to find the private key corresponding to any given public key in a reasonable amount of time. This would effectively compromise the security of the cryptosystem. Fortunately, no quantum computer capable of such an operation was anywhere near availability at the time, so this problem remained theoretical.

Approximately a decade later, in 2005, O. Regev devised a novel mathematical framework, termed Learning with Error (LWE) [28], which enables the construction

of new cryptosystems. This framework is based on an error term within a linear system of equations constructed on a lattice. The mathematical problem that he exploits for security is the hardness of the shortest vector problem (SVP). Over time, variants based on this problem were developed. Two such variants are Ring-LWE (R-LWE), which employs polynomials in place of the vectors and matrices, and Module-LWE (M-LWE), which combines R-LWE with (Plain-)LWE in a manner that results in multidimensional polynomials, vectors and matrices of polynomials. In 2009, Craig Gentry published the first full-homomorphic encryption scheme [13]. This development prompted renewed optimism regarding the advancement of FHE schemes, as it became evident that the concept was indeed feasible. However, the primary challenge that remained was the issue of performance. To enhance the efficiency of this scheme, the initial version, which was based on the ideal lattice, was adapted to the R-LWE scheme. Over time, significant advancements have been made in the development of these FHE schemes, which are constructed on basis of R-LWE [23]. However, the primary challenge persists, namely the performance, which is frequently 1000s of times slower than operations on the plain text.
In recent years, there has been a resurgence of interest in quantum computers as various companies compete to develop the first practical and useful quantum computer [14] [16]. Consequently, the performance of these computers has been steadily improving. If the promises made are accurate, it is possible that in 10 years, viable quantum computers will be available on the market. These computers could run Shor's Algorithm and thereby breach the security of RSA (and other) cryptosystems, potentially undermining the security of the internet as it currently stands. To circumvent such potential issues, the US National Institute of Standards and Technology (NIST) initiated an open competition in 2016, wherein individuals could submit novel cryptographic systems for analysis. Research teams from around the globe would then endeavor to identify vulnerabilities in these systems. In 2022, the NIST announced the first four winners [25], three of which were based on LWE. The two most recommended systems, CRYSTALS-Kyber [4] and CRYSTALS-Dilithium [9], are both based on M-LWE.

In light of these recent advancements in M-LWE-based encryption and the established R-LWE-based homomorphic encryption schemes, the main question of this thesis arises concerning the potential for integrating these two approaches: *Is it possible to port the R-LWE-based homomorphic encryption schemes to M-LWE and does this results in an improvement in performance?* The performance of an homomorphic encryption scheme is not merely a function of its processing speed, it is also determined by the reduction in memory usage or the increase in the depth

of operations that it is capable of performing. The depth of operations refers to the number of consecutive operations that can be performed without a significant increase in error, whereby the decryption produces erroneous results. Given that M-LWE employs matrices of polynomials in place of scalar polynomials in R-LWE, it is anticipated that the higher variance in error values will result in a greater depth of operations, as they could cancel out a bit. Should the advantages outweigh the disadvantages, this would facilitate new synergies between the current endeavour to enhance the security of a post-quantum internet and the construction of efficient and dependable homomorphic encryption algorithms. For instance, enhanced and high-performing implementations or even hardware accelerators could be reused, thereby enhancing the efficacy of homomorphic encryption while simultaneously reducing the cost of development.

The thesis is divided into five principal sections. The first section of the thesis provides an introduction to the mathematical background, wherein all necessary mathematical operations will be explained in sufficient detail. Subsequently, the LWE problems will be described in greater detail, and a basic LWE-based encryption scheme will be constructed. The scheme is capable of functioning on Plain-, Ring-, and Module-LWE. Following this, homomorphic encryption will be outlined, and the LWE-based encryption scheme will be expanded to become homomorphic for all three LWE types. These schemes will then be evaluated based on their memory usage, processing performance, and calculation depth. Ultimately, the main question will be addressed based on these findings.

2 Mathematical Background

In order to comprehend the mathematical principles underlying the encryption algorithms described in this thesis, it is first necessary to grasp a few fundamental concepts. However, it is assumed that the reader has a basic familiarity with linear algebra and polynomial calculus.

2.1 Preliminaries

The following notations are used throughout this thesis:

a ⇐Scalar: lowercase plain letter

$\mathbf{a}$ ⇐Vector: lowercase bold letter

$\mathbf{A}$ ⇐Matrix: uppercase bold letters

$a(x)$ ⇐Polynomial: lowercase plain letter with the variable in brackets

R ⇐Ring: uppercase plain

$x \leftarrow R$ ⇐choosing uniformly from all values in the ring: variable with leftarrow to ring

$x \leftarrow \chi_R$ ⇐choosing values from the ring around 0 (eg Discrete Gaussian): variable with leftarrow to an error distribution of the ring

$+$ ⇐Addition

$\cdot$ ⇐Multiplication between scalars and polynomials; dot product between vectors and/or matrices

$\odot$ ⇐Hadamard product (elementwise product)

$\otimes$ ⇐Outer product (Tensor product)

2.2 Lattice

All of the algorithms discussed in this thesis are based on lattices, which is why we will briefly focus on them in more detail. In general, lattices behave like any other vector space, but they only consist of discrete vectors. This means that the vectors only contain integers and not real numbers as in a vector space.

Let $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m\}$ be a set of linearly independent vectors of $\mathbb{R}^n$. The lattice L generated by $\mathbf{B}$ is the set of integer linear combinations of $\mathbf{B}$. $\mathbf{B}$ is called the basis of the lattice L. That is,

$$L(\mathbf{B}) = \{a_1\mathbf{b}_1 + \ldots + a_m\mathbf{b}_m | a_1, \ldots, a_m \in \mathbb{Z}\} \subset \mathbb{R}^n$$

Using a matrix $\mathbf{B}$, which contains the basis vectors as column vectors, we can generate L equivalently.

$$L(\mathbf{B}) = \{\mathbf{B} \cdot \mathbf{x} | \mathbf{x} \in \mathbb{Z}^m\} \subset \mathbb{R}^n$$

As in this definition, the integer n is the **dimension** of the lattice and m is its **rank**. If $m = n$, then L is a **full-rank** lattice, which is the usual case in this thesis.

An example of a lattice based on a basis $\mathbf{B}$ and all the points that can be created with it, also called the **span**, can be seen in the figure 1.

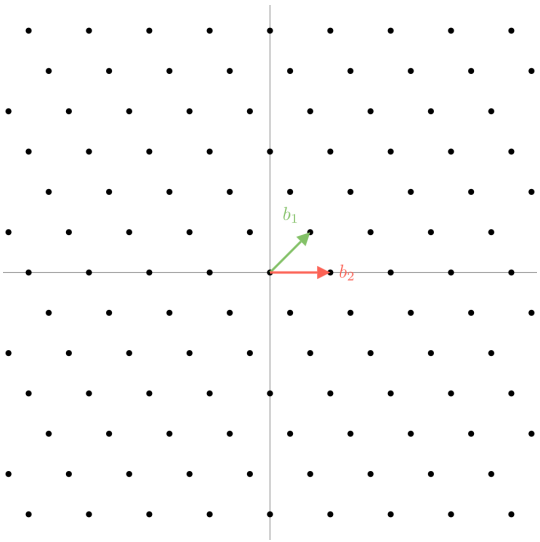

Figure 1: The span of an two-dimensional lattice with basis $B = \{b_1, b_2\}$.

2.3 Integer & Polynomial Rings with modulus

This section is based on the Book *Algebra* by *D. Plaumann*[27].

Rings

A ring is a set R on which addition ($+$) and multiplication ($\cdot$) can be performed and results in a new Element, which is also part of the set R.

$$+ : R + R \to R \text{ (Addition) and } \cdot : R \cdot R \to R \text{ (Multiplication)}$$

These calculations need to fulfill the following conditions:

for addition: R is an abelian group

- Associative property: $(a + b) + c = a + (b + c) | a, b, c \in R$
- Commutative property: $a + b = b + a | a, b \in R$
- Additive identity: There exists and element $0 \in R$ so that $a+0 = a | a \in R$
- Additive inverse: For each $a \in R$ there is an $-a \in R$ so that $a+(-a) = 0$

for multiplication: R is an monoid

- Associative property: $(a \cdot b) \cdot c = a \cdot (b \cdot c) | a, b, c \in R$
- Multiplicative identity: There exists and element $1 \in R$ so that $a \cdot 1 = 1 \cdot a = a | a \in R$

Addition and Multiplication are distributive

- $a \cdot (b + c) = a \cdot b + a \cdot c | a, b, c \in R$
- $(a + b) \cdot c = a \cdot c + b \cdot c | a, b, c \in R$

A ring is also called commutative if the multiplication is also commutative. For example, the ring over all integers $\mathbb{Z}$ is a commutative ring.

Modular arithmetic on Rings

Congruence arithmetic, or modular arithmetic, is the term used to describe arithmetic with remainders when dividing integers. In everyday life, this is mainly encountered in connection with clocks. After 60 minutes, the minute hand returns to the same position as before.

More generally, this can be described as $a \equiv b \mod q | a, b \in \mathbb{Z}, q \in \mathbb{N}$, where q is the module by which a and b are divided until the remainder of both is less than q. If a and b are then equal, they are congruent. Or in a more mathematical expression: If there is a k, such as $a - b = k \cdot q$, then a and b are congruent.

An congruence relation with module q on the set $\mathbb{Z}$, has the following properties: $k, q \in \mathbb{N}$ and $a, a', b, b', c \in \mathbb{Z}$

1. $a \equiv a \mod q$ (Reflexivity)
2. $a \equiv b \mod q$ **if** $b \equiv a \mod q$ (Symmetry)
3. **If** $a \equiv b \mod q$ **and** $b \equiv c \mod q$ **then** $a \equiv c \mod q$ (Transitivity)
4. **If** $a \equiv a' \mod q$ **and** $b \equiv b' \mod q$ **then** $a + b \equiv a' + b' \mod q$
5. **If** $a \equiv a' \mod q$ **and** $b \equiv b' \mod q$ **then** $a \cdot b \equiv a' \cdot b' \mod q$
6. If c and q are coprime and $c \cdot a \equiv c \cdot b \mod q$ **then** $a \equiv b \mod q$
7. **If** $a \equiv b \mod k \cdot q$ **then** $a \equiv b \mod q$

The congruence class is the set of all numbers for an integer $a \in \mathbb{Z}$ modulus q that produce the same remainder. It is defined as

$$[a]_q = \{b \in \mathbb{Z} | a \equiv b \mod q\}$$

It follows that two numbers are congruent if both congruence classes are equal:

$$a \equiv b \mod q \Leftrightarrow [a]_q = [b]_q$$

With this we can create a set of all congruence classes modulo q:

$$\mathbb{Z}_q = \mathbb{Z}/q = \mathbb{Z} \mod q = \{[a]_q | a = 0, 1, \cdots, q - 1\}$$

.

For example, $Z_3 = \{[0]_3, [1]_3, [2]_3\}$. With addition and multiplication it is possible to create a commutative ring from $\mathbb{Z}_q$.

$$[a]_q + [b]_q = [a + b]_q \text{ (addition) and } [a]_q \cdot [b]_q = [a \cdot b]_q \text{ (multiplication)}$$

This allows to create finite rings $\mathbb{Z}_q$ for every natural number q with q elements in each ring and to perform calculations inside these rings. For example, an ring with $n = 60$ can be created, which represents the minutes in every hour. If the minute hand shows now 48 and we want to know where it is after 3 times 13 minutes, we can calculate it like:

$$[48]_{60} + [3]_{60} \cdot [13]_{60} = [48 + 3 \cdot 13]_{60} = [87]_{60} = [27]_{60}$$

Polynomial Rings

A polynomial with coefficients in a ring R is expressed as

$$f(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{d-1} + a_d x^d | a_0, \cdots, a_d \in R$$

The variable d defines the degree $\deg(f(x))$ of the polynomial, which is the largest exponent in a polynomial.
Such polynomials can be added and multiplied like any other polynomial. Such a polynomial ring with one variable x and its coefficients in R is written as $R[x]$. This is a generalization of the rings we had before, because R is a subset of $R[x]$ ($R \subset R[x]$), since R is a polynomial with deg(0): $R = R[x] := a \cdot x^0 = a$.
Such a polynomial ring can also be defined over a finite ring, so that each coefficient is part of that finite ring. This is written as $R_q = \mathbb{Z}_q[x]$. The coefficients follow the same rules for addition and multiplication as described above. The following example takes place in the ring $R_5 = \mathbb{Z}_5[x]$ and $f, g \in R_5$ with $f(x) = 1 + 2x + 3x^2$ and $g(x) = 4 + 2x$:

$$
\begin{aligned}
f(x) \cdot 4 &= (1 + 2x + 3x^2) \cdot 4 \\
&= [4]_5 + [8]_5 x + [12]_5 x^2 \\
&= 4 + 3x + 2x^2 \\
f(x) + g(x) &= (1 + 2x + 3x^2) + (4 + 2x) \\
&= [5]_5 + [4]_5 x + [3]_5 x^2 \\
&= 4x + 3x^2 \\
f(x) \cdot g(x) &= (1 + 2x + 3x^2) \cdot (4 + 2x) \\
&= [4]_5 + [2]_5 x + [8]_5 x + [4]_5 x^2 + [12]_5 x^2 + [6]_5 x^3 \\
&= [4]_5 + [10]_5 x + [16]_5 x^2 + [6]_5 x^3 \\
&= 4 + 0x + 1x^2 + 1x^3
\end{aligned}
$$

As with any polynomial multiplication, the degree can increase as you multiply two polynomials, leading to increasingly larger polynomials with each multiplication. Since the modulo operation creates a finite ring, we can also create a modulo operation that creates a finite ring over a polynomial where the degree stays the same or is less than some upper bound. For this we have a ring R and $f(x), g(x), q(x), r(x) \in R[x], g \neq 0$, where $f(x)$ is a polynomial, $g(x)$ is the modulus, and $r(x)$ is the remainder:

$$f(x) = g(x) \cdot q(x) + r(x) \text{ and } \deg(r(x)) < \deg(g(x)).$$

After this calculation, $r(x)$ will be the remainder of $f(x)$ with a degree smaller than that of $g(x)$, which will be used for further calculations. With this, we can now define polynomial rings that have a module to generate finite coefficients and a polynomial function to generate finite degree. This is written as

$$R_q = \mathbb{Z}_q[x]/g(x)$$

In this thesis the modulus function $g(x)$ will be an polynomial with shape $x^d + 1$. In practice it should be an cyclotomic polynomial, where d needs to be a power of two $(2, 4, 8, \cdots)$. These polynomials are used as they simplify the calculation of the remainder, because the polynomial division can be simplified to addition and subtraction. When doing a polynomial division, one can subtract d from the

exponent and invert the coefficient if the exponent is greater than or equal to d. This must be repeated until the largest exponent is less than d. For example, $f \in \mathbb{Z}_5[x]/(x^3+1)$

$$\begin{aligned} f(x) &= 3 + 4x^2 + 2x^3 + x^5 + 3x^6 && \mod (x^3+1) \\ &= 3 + 4x^2 - 2 - x^2 - 3x^3 && \mod (x^3+1) \\ &= 3 + 4x^2 - 2 - x^2 + 3 \\ &= 4 + 3x^2 \end{aligned}$$

2.4 Polynomial Ring arithmetic using Vectors & Matrices

When calculating with polynomials, this can also be broken down into vector and matrix calculations. As we will see, this has advantages in performance, but also makes it easier to represent the polynomials when programming, as they can be handled as arrays of numbers. To create vectors from polynomials, each polynomial is separated into two vectors, the coefficient vector and the variable vector.

$$f(x) = a_0 + a_1 x + \cdots + a_{d-1}x^{d-1} + a_d x^d = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \\ a_d \end{bmatrix} \cdot \begin{bmatrix} 1 & x & \cdots & x^{d-1} & x^d \end{bmatrix}$$

The variable vector will always have the same shape as the coefficient vector. Because of the ring properties, the variable vector can always be factored out, since all polynomials have the same one in common. For this reason, only the coefficient vector will be written out in this thesis, to simplify the formulars.

When doing addition with such vectors, we just need to make sure that the vectors have the same length, which means that the polynomials must have the same degree. If this is not the case, we can fill the shorter vector with 0 so that they have the same degree. As our polynomials are always defined in a commutative ring, the associative, commutative and distributive properties apply. So addition would look like the following:

$$f(x) + g(x) = (a_0 + a_1x + \cdots + a_{d-1}x^{d-1} + a_dx^d) + (b_0 + b_1x + \cdots + b_{d-1}x^{d-1} + b_dx^d)$$
$$= \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \\ a_d \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{d-1} \\ b_d \end{bmatrix} = \begin{bmatrix} a_0 + b_0 \\ a_1 + b_1 \\ \vdots \\ a_{d-1} + b_{d-1} \\ a_d + b_d \end{bmatrix}$$

The process of multiplication is somewhat more complex, as the degree typically increases when two polynomials are multiplied together. As previously demonstrated, this is not the case when calculations are conducted within a polynomial ring. Accordingly, the polynomial ring $R_q = \mathbb{Z}_q/(x^d + 1)$ ensures that the degree of the polynomial will never exceed d. Consequently, even when performing multiplication, the degree will remain less than d. In order to perform polynomial multiplication as a vector operation, convolutions serve as an effective tool. These can be employed for a variety of polynomial multiplications, with a particular prevalence in the Fast Fourier Transform algorithm. In this context, they are employed to reduce the polynomial-polynomial multiplication with subsequent polynomial division for the modulus into a representation of matrix-vector multiplication. The objective is to construct a matrix that encodes not only the polynomial-polynomial multiplication but also the subsequent division. To achieve this, it is essential to utilise the cyclotomic polynomial as a module and ensuring that the two polynomials are part of the same ring. This can be expressed as $f, g \in \mathbb{Z}_q/(x^d + 1)$. In order to construct the calculation, one of the polynomial coefficient vectors must be transformed into a circulant matrix, where the diagonal and the lower triangle are positive and the upper triangle (without diagonal) is negative. This matrix is then multiplied by the other coefficient vector, resulting in the output coefficient vector. This new coefficient vector is identical to the result of the aforementioned polynomial multiplication with subsequent division. The general case is illustrated below:

$$f(x) \cdot g(x) = \begin{bmatrix} a_0 & -a_d & -a_{d-1} & \cdots & -a_2 & -a_1 \\ a_1 & a_0 & -a_d & \cdots & -a_3 & -a_2 \\ a_2 & a_1 & a_0 & \cdots & -a_4 & -a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{d-1} & a_{d-2} & a_{n-3} & \cdots & a_0 & -a_d \\ a_d & a_{d-1} & a_{d-2} & \cdots & a_1 & a_0 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{d-1} \\ b_d \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{d-1} \\ c_d \end{bmatrix}$$

The following examples will show this for two polynomials $f, g \in \mathbb{Z}_q/(x^3+1)$ with $f(x) = 3x^2+4x+1$ and $g(x) = x^2+6x+3$, using a normal polynomial multiplication and subsequent modulo, and the same with an single matrix-vector multiplication.

$$\begin{aligned} f(x) \cdot g(x) &= (1 + 4x + 3x^2) \cdot (3 + 6x + x^2) && \mod x^3 + 1 \\ &= 3 + 6x + x^2 + 12x + 24x^2 + 4x^3 + 9x^2 + 18x^3 + 3x^4 && \mod x^3 + 1 \\ &= 3 + 18x + 34x^2 + 22x^3 + 3x^4 && \mod x^3 + 1 \\ &= 3 + 18x + 34x^2 - 22 - 3x \\ &= -19 + 15x + 34x^2 \end{aligned}$$

$$\begin{aligned} f(x) \cdot g(x) &= (1 + 4x + 3x^2) \cdot (3 + 6x + x^2) && \mod x^3 + 1 \\ &= \begin{bmatrix} 1 & -3 & -4 \\ 4 & 1 & -3 \\ 3 & 4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 6 \\ 1 \end{bmatrix} \\ &= 3 \cdot \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} + 6 \cdot \begin{bmatrix} -3 \\ 1 \\ 4 \end{bmatrix} + 1 \cdot \begin{bmatrix} -4 \\ -3 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} -19 \\ 15 \\ 34 \end{bmatrix} \\ &= -19 + 15x + 34x^2 \end{aligned}$$

After these calculations the modulo operation could be applied on the coefficients either in the polynomial or in the vector, to generate the finite coefficient ring.

2.5 Multidimensional Rings

Rings can be not only in one dimension, but also in higher dimensions. This is written, as usual, with the dimensions as exponents in the ring. So a $m \times n$ ring matrix with modulus q would be written as $\mathbb{Z}_q^{m \times n}$.

The same can be done with finite polynomial rings. To make it easier, we will first define the ring $R = \mathbb{Z}_q[x]/(x^d + 1)$ and based on this ring we will define the form of a variable like $\mathbf{A} \in R^{m \times n}$, which would result in a $m \times n$ matrix where all values are elements of the finite polynomial ring R.

In higher dimensions, calculations are carried out in accordance with the standard mathematical rules. The aforementioned method is employed for the purpose of multiplying higher-dimensional finite polynomial rings. This transformation entails the conversion of each polynomial into a vector, or a matrix in the case of a multiplication. The process results in the generation of matrices of matrices and vectors of matrices, which can be regarded as larger matrices due to the commutative law. Consequently, the dimensions of matrices and vectors are augmented by the polynomial degree, d. For a more comprehensive illustration of this concept, please refer to Appendix A.1.

3 Learning with Errors

This chapter will examine the Learning with Errors (LWE) algorithm (referred to as Plain-LWE in this thesis) and its variants, namely Ring LWE (R-LWE) and Module LWE (M-LWE). The objective is to develop a basic LWE-based encryption scheme that will serve as the foundation for a homomorphic encryption system.

3.1 The Learning with Errors Problem

In 2005, Regev initially described the LWE problem [28]. He also demonstrated its hardness, but a detailed examination of this proof is beyond the scope of this discussion. The fundamental concept is based on a discrete system of linear equations, where error values are incorporated. This system of linear equations is composed of four fundamental components: the coefficients $\mathbf{A}$, the variables $\mathbf{s}$, the additional error $\mathbf{e}$, and the results $\mathbf{b}$.

$$\begin{aligned}
\mathbf{A}_{11}\mathbf{s}_1 + \mathbf{A}_{12}\mathbf{s}_2 + \cdots + \mathbf{A}_{1m}\mathbf{s}_m + \mathbf{e}_1 &= \mathbf{b}_1 \\
\mathbf{A}_{21}\mathbf{s}_1 + \mathbf{A}_{22}\mathbf{s}_2 + \cdots + \mathbf{A}_{2m}\mathbf{s}_m + \mathbf{e}_2 &= \mathbf{b}_1 \\
\vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots &= \vdots \\
\mathbf{A}_{n1}\mathbf{s}_1 + \mathbf{A}_{n2}\mathbf{s}_2 + \cdots + \mathbf{A}_{nm}s_m + \mathbf{e}_n &= \mathbf{b}_n
\end{aligned}$$

This can be expressed as a matrix and vectors, as illustrated in equation 3.1 and is the fundamental equation that underlies all LWE problems. The majority of the observed differences can be attributed to the specific ring or matrix/vector dimensions employed in each case. To construct a Plain-LWE scheme, it is necessary to set the ring as $R = \mathbb{Z}_q = \mathbb{Z}/q$. In all LWE schemes presented in this thesis, $n = m$.

$$\begin{aligned}
&\mathbf{A} \cdot \mathbf{s} + \mathbf{e} = \mathbf{b} \\
&\mathbf{A} \in R_q^{n \times m}, \mathbf{s} \in R_q^m, \mathbf{e}, \mathbf{b} \in R_q^n
\end{aligned} \tag{3.1}$$

The construction of an asymmetric encryption scheme necessitates the presence of two distinct keys: a secret key, denoted by sk, and a public key, represented by pk. The public key is used to encrypt a message, generating a ciphertext, ct. However, it is not possible to decrypt the ciphertext with this key in order to recreate the original message. This can only be accomplished with the secret key. In order for this scheme to be operational, it is crucial to ensure that it is not possible to generate the secret key from ciphertexts and/or the private key. For further insights into the general structure of asymmetric encryption, please refer to [10].

In the context of LWE, $\mathbf{s}$ represents the secret key, while both $\mathbf{A}$ and $\mathbf{b}$ collectively serve as the private key. The error term, $\mathbf{e}$, will be discarded after the calculation is complete, thus preventing any individual from gaining knowledge of it. If one possesses both the secret and private key, the system of linear equations can be efficiently solved using the Gaussian algorithm. However, for an individual with access only to the private key, the number of unknowns is to high, rendering the system unsolvable. Despite its seemingly straightforward nature, this equation is, in fact, quite challenging to resolve. The difficulty can be reduced to variants of the Shortest Vector Problem (SVP), which describes the complexity of identifying the shortest vector in a lattice. While this is a relatively straightforward undertaking in smaller dimensions, it becomes increasingly challenging as the dimensions increase.

3.2 LWE based encryption scheme

In this section, an elementary asymmetric encryption scheme will be constructed based on the principal equation 3.1. This construction is loosely based on the CRYSTALS-Kyber encryption scheme [4], but with some simplifications. The objective is not to develop a practical and secure encryption scheme, but rather one that can serve as a foundation for subsequent investigations into the construction of homomorphic encryption. Additionally, the aim is to transform the scheme from Plain-LWE into R-LWE and M-LWE.

In order to construct a functional encryption scheme, it is necessary to develop three algorithms:

1. **KeyGen**: For generating the private and secret key
2. **Encryption**: For encrypting some message m with the private key pk creating a ciphertext ct
3. **Decryption**: For decrypting the ciphertext ct with the secret key sk retrieving the original message m

In order to construct a Plain-LWE encryption scheme, all calculations are done in the ring $R = \mathbb{Z}_q$, where q is the modulus. If values from R are chosen uniformly, this is denoted by $x \leftarrow R$. Otherwise, if small values are chosen from R, this is written as $x \leftarrow \chi_R$. This can be done by choosing uniformly from a set of small numbers all in R (e.g., $-4, \ldots, 4$ if q is big enougth), or by choosing from an error distribution, such as the discrete Gaussian, as described in [28].

The initial stage of the process is to generate the private key, pk, and the secret key, sk. This is detailed in Algorithm 1. It uses the LWE equation from 3.1 as described before. The secret key, which the owner should never share, is the vector $\mathbf{s}$. The public key pk, which can be shared, consists of the transformation matrix $\mathbf{A}$ and the transformed secret key plus the error $\mathbf{b}$. The error $\mathbf{e}$ is discarded after the computation of $\mathbf{b}$. The values of $\mathbf{e}$ and $\mathbf{s}$ should be rather small, and $\mathbf{A}$ is uniformly sampled from R.

Algorithm 1 Sample LWE: KeyGen

1. $\mathbf{s} \leftarrow \chi_R^n$
2. $\mathbf{A} \leftarrow R^{n \times n}$
3. $\mathbf{e} \leftarrow \chi_R^n$
4. $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$
5. **return** $(pk := (\mathbf{A}, \mathbf{b}), sk := \mathbf{s})$

The encryption algorithm, which describes how to generate the ciphertext ct from an message m with the public key pk is described in Algorithm 2. The errors $\mathbf{e}_1$ and e_2 are randomly sampled with small values and used to create more uncertainty around the message. The same message can therefore be encrypted with different errors, which yields different ciphertexts. This makes it harder for attackers to find patterns in the decryption. The value $\mathbf{r}$ is sampled randomly form 0 and 1. The objective is to select a subset of $\mathbf{A}$ and $\mathbf{b}$ given that approximately 50% of the values in $\mathbf{r}$ will be 0. Consequently, these columns in $\mathbf{A}$ and $\mathbf{b}$ are effectively irrelevant (multiplied by 0). This helps to create more entropy between different encryptions, as a different subset of $\mathbf{A}$ and $\mathbf{b}$ will be used to encrypt each time.
The newly created values and the public key are used in the calculation of two values: $\mathbf{u}$ and v. The first term, $\mathbf{u}$, is an transposed subset of $\mathbf{A}$ with some error: $\mathbf{u} = \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_1$. It can be regarded as the inverse of $\mathbf{b}$, with the secret term, $\mathbf{s}$, omitted. The term v represents the actual message value, which is calculated from a subset of the variable b with a small error value added and the scaled message. The complete calculation is as follows: $v = \mathbf{b}^T \cdot \mathbf{r} + e_2 + (m \cdot \lfloor q/2 \rfloor)$. The scaled message $m \cdot \lfloor q/2 \rfloor$ is obtained by multiplying the original message, m, by the rounded-down

version of half the modulus. This operation results in the values of the message in the ring, which are either 0 or 1, being scaled from the message space into the ciphertext space, (0 to $q-1$). The resulting values are approximately as distant from each other as possible, which is $q/2$.

Algorithm 2 Sample LWE: Encryption

Require: $m \in \mathbb{Z}_2 = \{0,1\}$, $pk = (\mathbf{A}, \mathbf{b})$

1. $\mathbf{r} \leftarrow \{0,1\}^n$
2. $\mathbf{e}_1 \leftarrow \chi_R^n$
3. $\mathbf{u} = \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_1$
4. $e_2 \leftarrow \chi_R$
5. $v = \mathbf{b}^T \cdot \mathbf{r} + e_2 + (m \cdot \lfloor q/2 \rfloor)$
6. **return** $ct := (\mathbf{u}, v)$

The decryption of the ciphertext ct back to the original message m using the secret key sk is described in Algorithm 3.

Algorithm 3 Sample LWE: Decryption

Require: $ct = (\mathbf{u}, v)$, $sk = \mathbf{s}$

1. **return** $\left\lfloor \frac{1}{\lfloor q/2 \rfloor} \cdot \left[v - \mathbf{s}^T \cdot \mathbf{u} \right]_q \right\rceil_2$

To get a better understanding of the equation, consider the following simplification of the term in Algorithm 3.

$$
\begin{aligned}
&\left\lfloor \frac{1}{\lfloor q/2 \rfloor} \cdot \left[v - \mathbf{s}^T \cdot \mathbf{u} \right]_q \right\rceil_2 \\
&= \left\lfloor \frac{1}{\lfloor q/2 \rfloor} \cdot \left[\mathbf{b}^T \cdot \mathbf{r} + e_2 + (m \cdot \lfloor q/2 \rfloor) - \mathbf{s}^T \cdot (\mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_1) \right]_q \right\rceil_2 \\
&= \left\lfloor \frac{1}{\lfloor q/2 \rfloor} \cdot \left[(\mathbf{As} + \mathbf{e})^T \cdot \mathbf{r} + e_2 + (m \cdot \lfloor q/2 \rfloor) - \mathbf{s}^T \mathbf{A}^T \cdot \mathbf{r} - \mathbf{s}^T \mathbf{e}_1 \right]_q \right\rceil_2 \\
&= \left\lfloor \frac{1}{\lfloor q/2 \rfloor} \cdot \left[(\mathbf{As})^T \cdot \mathbf{r} + \mathbf{e}^T \mathbf{r} + e_2 + (m \cdot \lfloor q/2 \rfloor) - (\mathbf{As})^T \cdot \mathbf{r} - \mathbf{s}^T \mathbf{e}_1 \right]_q \right\rceil_2 \\
&= \left\lfloor \frac{1}{\lfloor q/2 \rfloor} \cdot \left[\mathbf{e}^T \mathbf{r} + e_2 + (m \cdot \lfloor q/2 \rfloor) - \mathbf{s}^T \mathbf{e}_1 \right]_q \right\rceil_2 \\
&= \left\lfloor \frac{\mathbf{e}^T \mathbf{r}}{\lfloor q/2 \rfloor} + \frac{e_2}{\lfloor q/2 \rfloor} + m - \frac{\mathbf{s}^T \mathbf{e}_1}{\lfloor q/2 \rfloor} \right\rceil_2 \\
&= \lfloor m' \rceil_2 = \begin{cases} m \in \{0, 1\} & \text{if } \left| \frac{\mathbf{e}^T \mathbf{r}}{\lfloor q/2 \rfloor} + \frac{e_2}{\lfloor q/2 \rfloor} - \frac{\mathbf{s}^T \mathbf{e}_1}{\lfloor q/2 \rfloor} \right| < \frac{q}{4} \\ error & \text{otherwise} \end{cases}
\end{aligned}
$$

As demonstrated by the calculation, by multiplying the cancellation term $\mathbf{u}$ with the secret $\mathbf{s}$, the transformation $(\mathbf{As})^T \cdot \mathbf{r}$ in v can be canceled out. This results in the message with some error values being added to it. The erroneous message will then be rounded, which will result in the original message. This process will only be successful if the absolute value of all error terms together is smaller than $\frac{q}{4}$. This is due to the fact that the possible values in the message are separated by a distance of $\frac{q}{2}$ from each other. Consequently, all values between $-\frac{q}{4} \mod q = \frac{3q}{4}$ and $\frac{q}{4}$ are rounded back to 0, while all values between $\frac{q}{4}$ and $\frac{3q}{4}$ are rounded to 1. Consequently, provided that the message (either 0 or $\frac{q}{2}$) is not shifted by more than $\frac{q}{4}$, it will remain within the rounding area of the original message. In the event that the error terms exceed the value of $\frac{q}{4}$, the resulting message will be a erroneous bit, which consequently leads to a corrupted message.

The current definition of this algorithm allows only 1 bit to be encoded at the time. This could be improved with some tricks, but for simplicity reasons we wont do that here. To observe the functioning of this algorithm in practice, please refer to the example in Appendix A.2.

3.3 Transforming Plain-LWE to R-LWE and M-LWE

To transform the Plain-LWE encryption scheme described in the section before into Ring-LWE encryption scheme, only a few changes need to be made. Most importantly, a polynomial ring will be defined as $R = \mathbb{Z}[x]_q/(x^d + 1)$, with the matrix dimension $n = 1$ and the degree $d > 1$. Because n equals 1 all vectors and matrices only contain a single value, which is a polynomial, as d is now greater than 1. Consequently, instead of having a vector $\mathbf{r}$, it will now be a polynomial in the ring R, where all coefficients are either 0 or 1. The message to be encrypted is also transformed into a polynomial in R with the message bits being the coefficients of the polynomial. Because of this, d bits can now be encoded in one message. Instead of using matrix arithmetic, polynomial arithmetic is now utilized. However, as stated in section 2.4, the polynomial arithmetic in the ring can also be transformed into matrix arithmetic. All equations stay the same and the structure of the Algorithms does not change, only the values which are in- and outputted. An full example of the three-step process for RLWE can be found in Appendix A.3.

In a similar way, Plain-LWE was transformed into R-LWE, R-LWE can be transformed into Module-LWE. Todo this, only the matrix dimension needs to be increase, so that $n > 1$. So instead of working with polynomials as in R-LWE, matrices and vectors of polynomials will be used. Thus a combination of matrix and polynomial arithmetic is necessary for computation. An example of such an calculation can found in Appendix A.4.

Given the differing dimensions of the matrix n and polynomial degree d among the various types of LWE, the variables and keys that must be stored and shared, as well as the messages themselves, exhibit a corresponding variation in dimension. An summarized overview of the differences can be found in Table 1.

As illustrated in the aforementioned table, Plain-LWE and R-LWE are each dependent on a single dimension variable, either n or d, respectively. The internal and external variables used in Plain-LWE frequently have varying dimensionalities. These variables encompass matrices, such as A, which is a element of the private key, as well as scalar values, such as e_2 and elements of the ciphertext. However, the majority of values are in the form of vectors. In contrast, R-LWE is characterized by all values being polynomials (coefficient vectors) of size d. M-LWE, on the other hand, is dependent on both dimension variables, as it is a generalization of the other two. This results in all variables for M-LWE having the combined size of Plain- and

Table 1: Comparison between the shapes of the internal & external variables for the different LWE Types. The polynomial degree is treated as a vector for better comparison. For the message and the ciphertext ℓ refers to the length (number of bits) of the message.

	Plain-LWE	R-LWE	M-LWE
$\mathbf{A}$	$\mathbb{Z}_q^{n \times n}$	$\mathbb{Z}_q^d$	$\mathbb{Z}_q^{n \times n \times d}$
$\mathbf{s}, \mathbf{b}, \mathbf{e}, \mathbf{e}_1$	$\mathbb{Z}_q^n$	$\mathbb{Z}_q^d$	$\mathbb{Z}_q^{n \times d}$
e_2	$\mathbb{Z}_q$	$\mathbb{Z}_q^d$	$\mathbb{Z}_q^d$
$\mathbf{r}$	$\mathbb{Z}_2^n$	$\mathbb{Z}_2^d$	$\mathbb{Z}_2^{n \times d}$
sk	$\mathbb{Z}_q^n$	$\mathbb{Z}_q^d$	$\mathbb{Z}_q^{n \times d}$
pk	$\mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n$	$\mathbb{Z}_q^d \times \mathbb{Z}_q^d$	$\mathbb{Z}_q^{n \times n \times d} \times \mathbb{Z}_q^{n \times d}$
m	$\ell \times \mathbb{Z}_2$	$\lceil \ell/d \rceil \times \mathbb{Z}_2^d$	$\lceil \ell/d \rceil \times \mathbb{Z}_2^d$
ct	$\ell \times (\mathbb{Z}_q^n \times \mathbb{Z}_q)$	$\lceil \ell/d \rceil \times (\mathbb{Z}_q^d \times \mathbb{Z}_q^d)$	$\lceil \ell/d \rceil \times (\mathbb{Z}_q^{n \times d} \times \mathbb{Z}_q^d)$

R-LWE. Consequently, the smallest values are vectors of size d, while the largest ones are 3D tensors of size $n \times n \times d$.

One notable shortcoming of Plain-LWE is that only a single bit can be encoded at a time. Consequently, the resulting encrypted messages are of the form $\ell \times (\mathbb{Z}_q^n \times \mathbb{Z}_q)$, where ℓ is the number of bits that needs to be encoded. In contrast, R-LWE and M-LWE permit the encryption of ℓ bits in blocks of size d. If the number of bits is a multiple of the dimension d, it can be split into ℓ/d blocks. Otherwise, the last block must be padded with zeros to create a full block. Consequently, in the worst case, an additional encryption with a polynomial of size d is required to encrypt a single bit.

The security of Plain-LWE is dependent on the matrix dimension n, whereas R-LWE is reliant on the polynomial degree d. Conversely, M-LWE represents a combination of the matrix dimension n and the polynomial degree d. Furthermore, the security of all three schemes hinges on the modulus q. It can be inferred that the value of n can be smaller in M-LWE than in Plain-LWE, and that the value of d can be smaller then that in R-LWE. This can be verified by examining three distinct LWE-based encryption schemes and their recommended security parameters, as illustrated in Table 2. As can be observed in this table, the large value of n for Plain-LWE results in matrices that are quite large. In contrast, the polynomials used in R-LWE are of significantly smaller degree, and the dimensions in M-LWE are somewhere between those of Plain-LWE and R-LWE for n and d. A more detailed analysis of the size cost implications of these schemes will be presented in Chapter 5.1.

Table 2: Variable size comparison of different LWE based encryption schemes

	Source	n	d	q
Plain-LWE	Frodo [3]	752		32767
R-LWE	Practical Key Exchange [32]		512	25601
M-LWE	CRYSTALS-Kyber [4]	3	256	7681

3.4 Criteria for comparing LWE-based encryption schemes

To evaluate the efficacy of distinct cryptographic protocols, a multitude of quantitative and qualitative criteria can be employed, as outlined in reference [15]. First, one must consider the theoretical security of the algorithm. In the event that theoretical vulnerabilities are identified, the algorithms can be modified to prohibit certain assumptions that may have led to the vulnerability, or if the problem is fundamental, the algorithm may not be suitable for use. The theoretical security of LWE, R-LWE, and M-LWE has already been demonstrated (see [28], [22], and [19], respectively).

In addition to theoretical considerations, there are numerous practical factors to be taken into account. As is the case with any software, there are a number of security concerns that must be addressed, such as bugs and implementation vulnerabilities. Additionally, practical concerns, including the time and memory costs of the algorithms, must be considered. This thesis will focus on these latter two aspects, as they are relatively straightforward to test and are of significant relevance when considering the potential real-world applications of the algorithms. A comparative analysis of the security features of the various schemes will not be undertaken, as it would require a significant investment of time and resources that would exceed the scope of this thesis.

The *time costs* are derived from the amount of computation required to generate the keys, encrypt, or decrypt the data. The most precise number can be obtained by counting the elementary operations a CPU needs to perform in order to run the algorithms. To simplify this somewhat, the run time for different algorithms will be used to obtain a *time cost*, which can be used for comparisons. In order to obtain a meaningful value, multiple runs will be conducted for the same algorithm, and the median value will be used. This should reduce the noise and return comparable values. This time cost is also called the Performance of the algorithms.

The *size cost* refers to the runtime and memory requirements for computation and the storage necessary for the storage of obtained results. The storage size in question also holds significance not only for the system in which the algorithm is initially

executed, but also for other systems as it represents the data to be transmitted. In this thesis, the focus will be on the storage size, as its easy to compute and compare.

The three LWE schemes can be compared to each other based on the aforementioned criteria. The following chapter will present the construction of a homomorphic encryption scheme based on these three schemes. Subsequently, the criteria will be extended to facilitate a comparison between the homomorphic schemes.

4 Homomorphic Encryption

This chapter will present a general overview of the concept of homomorphic encryption (HE). Subsequently, the encryption scheme developed in the previous chapter will be extended to create a homomorphic encryption scheme based on an already existing HE scheme. This will result in an R-LWE-based homomorphic encryption scheme. The forthcoming and novel step will be to generalise the concepts of the R-LWE-based HE scheme into an M-LWE-based one. Finally, a brief overview will be provided of additional criteria for comparing HE schemes with one another.

4.1 Introduction to Homomorphic Encryption

Homomorphic encryption is a specialized cryptographic system that enables the execution of operations on encrypted data in a manner analogous to that of unencrypted data. Such operations may include, but are not limited to, addition and multiplication. This capability permits the outsourcing of data storage and computation to external services while maintaining the confidentiality of the data. This results in the formation of a zero-trust environment, wherein the necessity for trust in external providers is negated due to their inability to decrypt the data, while they remain capable of working with it. Moreover, the occurrence of data breaches would be effectively eliminated, as the data is always encrypted.

The concept was initially proposed by Rivest et al. in 1978 [29]. The authors proposed a homomorphic encryption (HE) scheme based on RSA, which is able to perform multiplication operations. This allows for the multiplication of two ciphertexts, each encrypted with the same RSA private key, with the result of the multiplication being retrievable after decryption. This result will be identical to that which would be obtained if the two ciphertexts were multiplied unencrypted. A system that is capable of performing a single operation (e.g., multiplication) for an unlimited amount of times is referred to as **Partially Homomorphic Encryption (PHE)** [1, 8]. Similarly, the same principle can also be observed with the El Gamal cryptosystem [11] for multiplication or the Paillier cryptosystem [26] for addition. If an encryption scheme with the capacity to perform addition and multiplication

on a single ciphertext can be constructed, then fully powered compute engines can be created using homomorphic circuits based on NAND gates in binary space.
But for a long time, it was not feasible to integrate addition and multiplication into a unified encryption scheme until the advent of the BGN scheme in 2005 [2]. The scheme allows for the realization of an arbitrary number of additions and a single multiplication. This novel scheme can be classified as a **Somewhat Homomorphic Encryption (SWHE)** scheme. Such schemes permit the execution of multiple types of operations, although only a limited number of times.

In 2009, C. Gentry introduced the first **Fully Homomorphic Encryption (FHE)** scheme [13]. The scheme was based on an ideal lattice and enabled the execution of addition and multiplication, which could be performed an unlimited number of times. But the primary factor contributing to the initial success of the first FHE scheme was the process of bootstrapping. The concept is to execute the decryption process, but not with the conventional secret key, but rather with an encrypted version of the secret key, frequently referred to as a "refreshing key." The decryption is performed in a homomorphic manner on the ciphertext. This results in the generation of a new ciphertext, wherein the internal error is reduced while still maintaining encryption. The new ciphertext can then be subjected to further operations, and if the error increases, the bootstrapping technique can be re-executed. This allows for the possibility of performing an unlimited number of computations, provided that the bootstrapping process is repeated at regular intervals. However, there are two significant drawbacks to this approach. Firstly, the bootstrapping process is highly resource-intensive, which can lead to suboptimal performance. Secondly, the new refresh key must be transmitted alongside the private key, increasing the amount of data that needs to be sent. Furthermore, if an attacker is able to decrypt the refresh key, they gain access to the secret key.

By employing this bootstrapping technique, SWHE schemes can be transformed into FHE schemes, as they are capable of reducing their own internal error. This innovation has led to the development of numerous new FHE encryption schemes in the subsequent years. In 2011, Brakerski and Vaikuntanathan published two new FHE schemes based on the bootstrapping technique. One was based on LWE [6] and the other on R-LWE [7]. Subsequently, numerous enhancements have been proposed (for further details, please see [1], [8] or [23]). One such enhanced scheme was the BFV scheme (also called just FV scheme) [12], which will be used to create the HE scheme based on the encryption scheme build in the last chapter.

Table 3: Comparison of the three different types of homomorphic encryption

HE Type	Operations	Number of consecutive operations	Examples
Partially Homomorphic Encryption (PHE)	addition **OR** multiplication	unlimited	ElGamal [11], RSA [30], Paillier [26]
Somewhat Homomorphic Encryption (SWHE)	addition **AND** multiplication	limited	BGN [2]
Fully Homomorphic Encryption (FHE)	addition **AND** multiplication	unlimited	BVF [12], LTV [20]

4.2 Creating an Somewhat Homomorphic Encryption scheme

In this section, the R-LWE based encryption scheme from section 3.3 will be extend to create an SWHE scheme. In order to implement a homomorphic encryption scheme based on LWE, it is necessary to define three additional functions in addition to the three functions defined in section 3.1:

1. **Addition**: This operation takes two ciphertexts as inputs and returns a new ciphertext by adding them together.
2. **Relinearization KeyGen**: This operation accepts a secret and a mapping value as input and generates a Relinearization Key (rlk). The rlk is necessary for creating a functional multiplication algorithm for LWE-based schemes. In other HE schemes, such a process is often called evaluation key. In this context we use the more specific term relinearization key.
3. **Multiplication**: This operation takes two ciphertexts and a rlk as inputs and performs a multiplication operation on the ciphertexts, with the help of the rlk, returning a new ciphertext.

The construction of the addition and multiplication operations will be based on a slightly modified version of the BFV scheme [12]. The objective is to develop a fully functional SWHE scheme that can subsequently be generalized to M-LWE. The step of creating a FHE scheme using bootstrapping will not be addressed in this thesis, as it was not feasible to incorporate it within the scope of this work. However, given that bootstrapping is based on homomorphic operations, as long as these are successfully implemented, it should be possible to also construct the bootstrapping process.

Addition

The objective is to develop a method for adding encrypted messages in such a way that the result is identical to that obtained by adding the plaintext messages. This can be achieved by simply adding the matching ciphertext variables together. The resulting error is thus increased linearly, as the errors from both ciphertexts are simply added together. Further details on this approach can be found in the BFV scheme.

Algorithm 4 R-LWE: Addition

Require: $ct_1 = (u_1, v_2)$, $ct_2 = (u_2, v_2)$

1. **return** $ct_{add} = ([u_1 + u_2]_q, [v_1 + v_2]_q)$

The newly created ct_{add} can then be employed, like any other ciphertext, for decryption or subsequent utilization in other operations. However, it is important to remember that the error in it has increased, which may potentially result in the generation of an incorrect decrypted message at some point.

Multiplication

Generating a functional multiplication operation in ciphertext space is a more complex process. To simplify the subsequent derivations and explanations, a simplification is made based on Algorithm 3 and can be seen in equation 4.1.

$$ct(s)_q = v - s \cdot u \tag{4.1}$$

Let ct_1 and ct_2 be two ciphertext that we want to use, with $ct_1(s) = v_1 - s \cdot u_1$ and $ct_2(s) = v_2 - s \cdot u_2$. The multiplication of these two values results in the equation 4.2

$$\begin{aligned}
[ct_1(s) \cdot ct_2(s)]_q &= [(v_1 - s \cdot u_1) \cdot (v_2 - s \cdot u_2)]_q \\
&= [v_1 \cdot v_2 - v_1 \cdot u_2 \cdot s - v_2 \cdot u_1 \cdot s + u_1 \cdot u_2 \cdot s^2]_q \\
&= [\underbrace{v_1 \cdot v_2}_{v_m} - \underbrace{(v_1 \cdot u_2 + v_2 \cdot u_1)}_{u_m} \cdot s + \underbrace{u_1 \cdot u_2}_{x_m} \cdot s^2]_q \\
&= [v_m - u_m \cdot s + x_m \cdot s^2]_q
\end{aligned} \tag{4.2}$$

Equation 4.2 results in the formation of three blocks, each dependent on a different

power of s ($s^0 = 1$, s^1 and s^2). In comparison to Equation 4.1, it can be observed that the current equation is similar, with the exception of the additional $x_m \cdot s^2$ factor. A method is required to approximate $x_m \cdot s^2$ and combine it with v_m and u_m, in order to reduce the degree of the equation from two to one. This process is known as relinearization. The formalization can be observed in Equation 4.3, where v'_m and u'_m are extended to include $x_m \cdot s^2$ and r represents the error that is created in this process, which should be minimized to ensure successful decryption.

$$[v_m - u_m \cdot s + x_m \cdot s^2]_q = [v'_m - u'_m \cdot s + r]_q \tag{4.3}$$

In order to resolve this issue, the *modulus switching* technique from BFV will be employed. The initial step is to define a Relinearization Key (rlk), which masks s^2. In this process, the value s^2 will be multiplied with a new constant, p. This constant is essential for reducing the error that is generated when "decrypting" the rlk (see equation 4.5). The form of the masked value is based on the public key (see algorithm 1), such that when A_{rlk} and b_{rlk} are "decrypted" with s, by putting them in equation 4.1, the original value $p \cdot s^2$ is obtained. The generation of this rlk is described in Algorithm 5.

Algorithm 5 R-LWE: rlk Generation

Require: s

1. $A \leftarrow R_{p \cdot q}$
2. $e \leftarrow \chi'_R$
3. $b = [A \cdot s + e + p \cdot s^2]_{p \cdot q}$
4. **return** $rlk := (A_{rlk}, b_{rlk})$

Utilizing the rlk, $x_m \cdot s^2$ is now decomposited into two distinct components. One component, designated xv_m, is added to v_m, while the other, xu_m, is added to u_m.

$$(xu_m, xv_m) = \left(\left[\left\lfloor \frac{x_m \cdot A_{rlk}}{p} \right\rceil \right]_q, \left[\left\lfloor \frac{x_m \cdot b_{rlk}}{p} \right\rceil \right]_q \right) \tag{4.4}$$

The process of "decryption", as illustrated in equation 4.5, reveals that x_m, a random element within R_q, is multiplied with the error e_{rlk}. This means, in worst case $x_m = q - 1$, which is then multiplied with an small error, for example 1. This would mean the error is nearly q, but as discussed in section 3.2, the error should not be bigger than $\frac{q}{4}$, otherwise the decryption does not work. To mitigate this, the error is

divided by p, thereby reducing its impact. In order to permit the creation of $x_m \cdot s^2$, s^2 was multiplied by p in the rlk to reverse the effect of dividing by p later on.

$$\begin{aligned} xv_m - xu_m \cdot s &= \left[\left\lfloor \frac{x_m \cdot b_{rlk}}{p} \right\rceil\right]_q - \left[\left\lfloor \frac{x_m \cdot A_{rlk}}{p} \right\rceil\right]_q \cdot s \\ &\approx \left[\frac{x_m \cdot b_{rlk}}{p} - \frac{x_m \cdot A_{rlk}}{p} \cdot s\right]_q \\ &\approx \left[\frac{x_m \cdot (A_{rlk} \cdot s + e_{rlk} + p \cdot s^2)}{p} - \frac{x_m \cdot A_{rlk} \cdot s}{p}\right]_q \\ &\approx \left[\frac{x_m \cdot A_{rlk} \cdot s}{p} + \frac{x_m \cdot e_{rlk}}{p} + \frac{x_m \cdot p \cdot s^2}{p} - \frac{x_m \cdot A_{rlk} \cdot s}{p}\right]_q \\ &\approx \left[\frac{x_m \cdot e_{rlk}}{p} + x_m \cdot s^2\right]_q \end{aligned} \tag{4.5}$$

The complete algorithm for multiplying can be found in Algorithm 6. In order for the algorithm to function correctly, it is necessary to multiply each of the factors by the value of $\frac{2}{q}$. Further details on this process can be found in the BFV paper [12].

Algorithm 6 R-LWE: Multiplication

Require: $rlk = (A_{rlk}, b_{rlk})$, $ct_1 = (v_1, u_1)$, $ct_2 = (v_2, u_2)$

1. $v_m = \left[\left\lfloor \frac{2}{q} \cdot (v_1 \cdot v_2) \right\rceil\right]_q$
2. $u_m = \left[\left\lfloor \frac{2}{q} \cdot (v_1 \cdot u_2 + v_2 \cdot u_1) \right\rceil\right]_q$
3. $x_m = \left[\left\lfloor \frac{2}{q} \cdot (u_1 \cdot u_2) \right\rceil\right]_q$
4. $xu_m = \left[\left\lfloor \frac{x_m \cdot A_{rlk}}{p} \right\rceil\right]_q$
5. $xv_m = \left[\left\lfloor \frac{x_m \cdot b_{rlk}}{p} \right\rceil\right]_q$
6. **return** $ct_m := \left(\left[u_m + xu_m\right]_q, \left[v_m + xv_m\right]_q\right)$

4.3 Generalizing from R-LWE to M-LWE

The objective in this section is to extend the algorithms defined in the previous section in order to make them applicable to M-LWE. As previously stated in section 3.3, M-LWE is a generalization of R-LWE, where matrices and vectors of polynomials are used. Consequently, the dimension n will be set to a value greater than 1. In

consequence, the dimensions of nearly all variables do change (see Table 1). Most importantly, $\mathbf{u}$ and $\mathbf{s}$ will now be vectors of length n, instead of single polynomials.

Addition

The addition operation has no impact on the shape of u and v, and thus the same algorithm can be used as before. The only difference is that the input ciphertexts and the newly created ciphertext are of shape $R_q^n \times R_q$.

Algorithm 7 M-LWE: Addition

Require: $ct_1 = (\mathbf{u}_1, v_2)$, $ct_2 = (\mathbf{u}_2, v_2)$

1. **return** $ct_{add} = ([\mathbf{u}_1 + \mathbf{u}_2]_q, [v_1 + v_2]_q)$

Multiplication

In contrast, the concept of multiplication is more complex due to the necessity of dealing with changing dimensions. When equation 4.1 is applied, the term $\mathbf{s}\cdot\mathbf{u}$ is now a dot product between two vectors, rather than a simple polynomial multiplication. The objective is, as before, to generate new v'_m and $\mathbf{u}'_m$ terms, which can be used for further operations or decryption. In contrast to the previous iteration, $\mathbf{u}'_m$ must now be represented as a vector rather than a polynomial.

The equation resulting from the multiplication of two ciphertexts is given by Equation 4.6. Given the complexity of this equation and process, a brief overview of the requisite steps will be provided in the subsequent paragraphs.

$$
\begin{aligned}
[ct_1(s) \cdot ct_2(s)]_q &= [(v_1 - \mathbf{s}\cdot\mathbf{u}_1)\cdot(v_2 - \mathbf{s}\cdot\mathbf{u}_2)]_q \\
&= [(v_1 - \sum_{i=0}^{n-1} s_i u_{1i})\cdot(v_2 - \sum_{i=0}^{n-1} s_i u_{2i})]_q \\
&= [v_1\cdot v_2 - v_1\cdot\sum_{i=0}^{n-1} s_i u_{2i} - v_2\cdot\sum_{i=0}^{n-1} s_i u_{1i} + \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} u_{1i}u_{2j}s_i s_j]_q \\
&= [v_1\cdot v_2 - v_1\cdot\mathbf{u}_2\cdot\mathbf{s} - v_2\cdot\mathbf{u}_1\cdot\mathbf{s} + \mathrm{sum}((\mathbf{u}_1\otimes\mathbf{u}_2)\odot(\mathbf{s}\otimes\mathbf{s}))]_q \\
&= [\underbrace{v_1\cdot v_2}_{v_m} - \underbrace{(v_1\cdot\mathbf{u}_2 + v_2\cdot\mathbf{u}_1)}_{\mathbf{u}_m}\cdot\mathbf{s} + \mathrm{sum}((\underbrace{\mathbf{u}_1\otimes\mathbf{u}_2}_{\mathbf{X}_m})\odot(\mathbf{s}\otimes\mathbf{s}))]_q \\
&= [v_m - \mathbf{u}_m\cdot\mathbf{s} + \mathrm{sum}(\mathbf{X}_m\odot(\mathbf{s}\otimes\mathbf{s}))]_q
\end{aligned}
\tag{4.6}
$$

The first technique employed was to convert the vector dot product into its sum form. As per the definition of the dot product between two vectors, it can be rewritten as a sum: $\mathbf{s} \cdot \mathbf{u} = \sum_{i=0}^{n-1} s_i u_i$. This step is derived from the calculations presented in [24].
The subsequent step is to transform the resulting sums once more. With the single sums, this is a relatively straightforward process, as they can simply be reformulated as dot products with an additional scalar (polynomial) multiplication. Essentially just reverting the previous step. With some bracketing and factoring out s, a new vector $\mathbf{u}_m$ can be created.
For the double sum, it is a bit more difficult process to extract a new x_m. The main Idea here is, that because of the double sum, an $n \times n$ matrix with all combinations of i and j is generated and all values are then added up. For example with $n = 3$ the following matrix will be created:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} u_{1i}u_{2j}s_i s_j = \text{sum}\left(\begin{bmatrix} u_{11}u_{21}s_1s_1 & u_{12}u_{21}s_2s_1 & u_{13}u_{21}s_3s_1 \\ u_{11}u_{22}s_1s_2 & u_{12}u_{22}s_2s_2 & u_{13}u_{22}s_3s_2 \\ u_{11}u_{23}s_1s_3 & u_{12}u_{23}s_2s_3 & u_{13}u_{23}s_3s_3 \end{bmatrix}\right)$$

The sum operation is simply a summation of all values, which is sometimes referred to as the *grand sum*. This is just a double dot product with a vector of length n, where all values are 1, which is denoted by the symbol 1-vector ($\mathbf{1}$): $\text{sum}(\mathbf{X}) := \mathbf{1} \cdot \mathbf{X} \cdot \mathbf{1} = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \mathbf{X}_{ij}$

The next step involves splitting the $n \times n$ matrix into two matrices, one for the u values and one for the s values. This can be seen in the equation below. Each term in the matrix is a product of four values, which can be split apart using the Associative Law. The $\mathbf{u}$ and $\mathbf{s}$ values are then multiplied separately, and the two matrices are multiplied together again using element-wise multiplication, also known as the Hadamard product, denoted by the $\odot$ symbol. Finally, the individual matrices can be decomposed into vector operations. This can be achieved through the use of the outer product, also referred to as the tensor product, which is represented by the symbol $\otimes$.

$$\begin{bmatrix} u_{11}u_{21}s_1s_1 & u_{12}u_{21}s_2s_1 & u_{13}u_{21}s_3s_1 \\ u_{11}u_{22}s_1s_2 & u_{12}u_{22}s_2s_2 & u_{13}u_{22}s_3s_2 \\ u_{11}u_{23}s_1s_3 & u_{12}u_{23}s_2s_3 & u_{13}u_{23}s_3s_3 \end{bmatrix} = \begin{bmatrix} u_{11}u_{21} & u_{12}u_{21} & u_{13}u_{21} \\ u_{11}u_{22} & u_{12}u_{22} & u_{13}u_{22} \\ u_{11}u_{23} & u_{12}u_{23} & u_{13}u_{23} \end{bmatrix} \odot \begin{bmatrix} s_1s_1 & s_2s_1 & s_3s_1 \\ s_1s_2 & s_2s_2 & s_3s_2 \\ s_1s_3 & s_2s_3 & s_3s_3 \end{bmatrix}$$
$$= (\mathbf{u}_1 \otimes \mathbf{u}_2) \odot (\mathbf{s} \otimes \mathbf{s})$$

When doing all these steps, a separation between $\mathbf{u}$ and $\mathbf{s}$ is achieved. The new factor $\mathbf{u}_1 \otimes \mathbf{u}_2$ will be the new matrix $\mathbf{X}_m$, as shown in equation 4.6.

The next step is to find a method for approximating the double sum in order to add it to v_m and $\mathbf{u}_m$. This needs to be done in a manner analogous to equation 4.3. Previously, a masking of s^2 was employed in order to eliminate this term. In the current context, an analogous issue arises with $\mathbf{s} \otimes \mathbf{s}$ and a second problem emerges, namely the shape of v_m as a polynomial and $\mathbf{u}_m$ as a vector.
As a first step, it is necessary to revisit the original rlk generation process, as outlined in Algorithm 5. In order to transform it into M-LWE, the same dimensions as those employed in the standard M-LWE key generation process are used: specifically, $\mathbf{A} \in R_{p \cdot q}^{n \times n}$ and $\mathbf{e}, \mathbf{s} \in \chi_R^{'n}$. The original equation was $b = [A \cdot s + e + p \cdot s^2]_{p \cdot q}$. When calculating the first part of $\mathbf{b}$ we get $\mathbf{A} \cdot \mathbf{s} + \mathbf{e}$, which is now an vector in R^n. As the second part needs to be added to this vector, the masked part (formerly s^2) needs to be a vector of the same dimension. As $\mathbf{s} \otimes \mathbf{s}$ is a matrix of dimension $n \times n$, it cannot be used directly. However, it can be split into n n-dimensional vectors, which can then be used instead. A similar approach must be taken with the $\mathbf{u}_1 \otimes \mathbf{u}_2$ matrix. The matching vectors of both matrices must be multiplied elementwise. This is a feasible approach, as all values within the matrix will be summed collectively at the final step, which can be done in any order (commutative law). This process is shown in the equation below and with it, the $\mathbf{X}_m$ matrix is split into n n-dimensional $\mathbf{x}_{mi}$ vectors and there corresponding secret vectors $\mathbf{s}'_i$ are created.

$$
\begin{aligned}
&\operatorname{sum}\left((\mathbf{u}_1 \otimes \mathbf{u}_2) \odot (\mathbf{s} \otimes \mathbf{s})\right) \\
&= \operatorname{sum}\left(\begin{bmatrix} u_{11}u_{21} & u_{12}u_{21} & u_{13}u_{21} \\ u_{11}u_{22} & u_{12}u_{22} & u_{13}u_{22} \\ u_{11}u_{23} & u_{12}u_{23} & u_{13}u_{23} \end{bmatrix} \odot \begin{bmatrix} s_1s_1 & s_2s_1 & s_3s_1 \\ s_1s_2 & s_2s_2 & s_3s_2 \\ s_1s_3 & s_2s_3 & s_3s_3 \end{bmatrix}\right) \\
&= \operatorname{sum}\left(\begin{bmatrix} u_{11}u_{21} \\ u_{11}u_{22} \\ u_{11}u_{23} \end{bmatrix} \odot \begin{bmatrix} s_1s_1 \\ s_1s_2 \\ s_1s_3 \end{bmatrix} + \begin{bmatrix} u_{12}u_{21} \\ u_{12}u_{22} \\ u_{12}u_{23} \end{bmatrix} \odot \begin{bmatrix} s_2s_1 \\ s_2s_2 \\ s_2s_3 \end{bmatrix} + \begin{bmatrix} u_{13}u_{21} \\ u_{13}u_{22} \\ u_{13}u_{23} \end{bmatrix} \odot \begin{bmatrix} s_3s_1 \\ s_3s_2 \\ s_3s_3 \end{bmatrix}\right) \\
&= \operatorname{sum}\left(u_{11} \cdot \begin{bmatrix} u_{21} \\ u_{22} \\ u_{23} \end{bmatrix} \odot s_1 \cdot \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} + u_{12} \cdot \begin{bmatrix} u_{21} \\ u_{22} \\ u_{23} \end{bmatrix} \odot s_2 \cdot \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} + u_{13} \cdot \begin{bmatrix} u_{21} \\ u_{22} \\ u_{23} \end{bmatrix} \odot s_3 \cdot \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}\right) \\
&= \operatorname{sum}\left(\sum_{i=0}^{n-1} (\underbrace{u_{1i} \cdot \mathbf{u}_2}_{\mathbf{x}_{mi}}) \odot (\underbrace{s_i \cdot \mathbf{s}}_{\mathbf{s}'_i})\right)
\end{aligned}
$$

In order to use $s_i \cdot \mathbf{s} = \mathbf{s}'_i$ in the *rlk* generation, the fixed s^2 in the original *rlk* generation algorithm 5 needs to be replaced with a variable, so the different $\mathbf{s}'_i$ can be encoded. The *rlk* generation process for a single $\mathbf{s}_i$ is illustrated in Algorithm 8, where $\mathbf{s}'$ represents the individual $s_i \cdot \mathbf{s}$ values. However, n *rlk* values, as one is required for each $\mathbf{s}'_i$, need to be created in order to encode the whole $\mathbf{s} \otimes \mathbf{s}$ matrix. The full *rlk* generation can be found in algorithm 10.

Algorithm 8 M-LWE: *rlk* Generation for a single $\mathbf{s}_i$

Require: $\mathbf{s}$, $\mathbf{s}' = s_i \cdot \mathbf{s}$

1. $\mathbf{A} \leftarrow R_{p \cdot q}^{n \times n}$
2. $\mathbf{e} \leftarrow \chi_R'^n$
3. $\mathbf{b} = [\mathbf{A} \cdot \mathbf{s} + \mathbf{e} + p \cdot \mathbf{s}']_{p \cdot q}$
4. **return** $rlk := (\mathbf{A}_{s'}, \mathbf{b}_{s'})$

As before, the *rlk* can be used to create two values, $\mathbf{xu}_m$ and xv_m. These values will be added to $\mathbf{u}_m \in R_q^n$ and $v_m \in R_q$, respectively. Therefore, it is necessary for these values to have the same shape. However, as the encryption process used near identical formulas, the correct shapes come by themselves. Thus, equation 4.4 can be translated into M-LWE, as shown in equation 4.7.

$$(\mathbf{xu}_m, xv_m) = \left(\sum_{i=0}^{n-1} \left[\left\lfloor \frac{\mathbf{A}_{\mathbf{s}'_i} \cdot \mathbf{x}_{mi}}{p} \right\rceil \right]_q, \sum_{i=0}^{n-1} \left[\left\lfloor \frac{b_{\mathbf{s}'_i} \cdot \mathbf{x}_{mi}}{p} \right\rceil \right]_q \right) \tag{4.7}$$

All this can be combined now into a single M-LWE multiplication algorithm, as seen in 9. And the new updated Key Generation, with the generation of the rlk can be seen in 10.

Algorithm 9 M-LWE: Multiplication

Require: $rlk = ((\mathbf{A}_{\mathbf{s}'_0}, \mathbf{b}_{\mathbf{s}'_0}), \ldots, (\mathbf{A}_{\mathbf{s}'_{n-1}}, \mathbf{b}_{\mathbf{s}'_{n-1}}))$, $ct_1 = (\mathbf{u}_1, v_1)$, $ct_2 = (\mathbf{u}_2, v_2)$

1. $v_m = \left[\left\lfloor \frac{2}{q} \cdot (v_1 \cdot v_2) \right\rceil \right]_q$
2. $\mathbf{u}_m = \left[\left\lfloor \frac{2}{q} \cdot (v_1 \cdot \mathbf{u}_2 + v_2 \cdot \mathbf{u}_1) \right\rceil \right]_q$
3. $\mathbf{x}_m = \left(\left[\left\lfloor \frac{2}{q} \cdot (u_{10} \cdot \mathbf{u}_2) \right\rceil \right]_q, \cdots, \left[\left\lfloor \frac{2}{q} \cdot (u_{1n-1} \cdot \mathbf{u}_2) \right\rceil \right]_q \right)$
4. $\mathbf{xu}_m = \sum_{i=0}^{n-1} \left[\left\lfloor \frac{\mathbf{A}_{\mathbf{s}'_i} \cdot \mathbf{x}_{mi}}{p} \right\rceil \right]_q$
5. $xv_m = \sum_{i=0}^{n-1} \left[\left\lfloor \frac{b_{\mathbf{s}'_i} \cdot \mathbf{x}_{mi}}{p} \right\rceil \right]_q$
6. **return** $ct_m := \left([\mathbf{u}_m + \mathbf{xu}_m]_q, [v_m + xv_m]_q \right)$

Algorithm 10 M-LWE: KeyGen

1. $\mathbf{s} \leftarrow \chi_R^n$
2. $\mathbf{A} \leftarrow R^{n \times n}$
3. $\mathbf{e} \leftarrow \chi_R^n$
4. $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$
5. $rlk = ()$
6. **for** $i = 0$ **to** $n - 1$ **do**
7. $\quad \mathbf{A}_i \leftarrow R_{p \cdot q}^{n \times n}$
8. $\quad \mathbf{e}_i \leftarrow \chi_R^{'n}$
9. $\quad \mathbf{b}_i = [\mathbf{A}_i \cdot \mathbf{s} + \mathbf{e}_i + p \cdot \mathbf{s} \cdot s_i]_{p \cdot q}$
10. $\quad rlk_i = (\mathbf{A}_i, \mathbf{b}_i)$
11. **end for**
12. **return** $(pk := (\mathbf{A}, \mathbf{b}), sk := \mathbf{s}, rlk := rlk)$

Generate R-LWE from M-LWE

One simple method for evaluating the efficacy of the generalization is to generate the R-LWE scheme from the M-LWE scheme with $n = 1$. The initial step is to calculate the rlk. With a dimension of 1, there is only a single rlk, which is calculated with

$s'_0 = s_0 \cdot \mathbf{s} = s_0 \cdot s_0 = s^2$. It can be seen that the M-LWE $rlk(s, s^2)$ (see algorithm 8) is identical to the R-LWE $rlk(s)$ (see algorithm 5), as both $A^{1\times1}$ and e^1 are single polynomials. As with s, the vector $\mathbf{u}$ is a single polynomial. Consequently, the calculation of u_m is identical between M-LWE (algorithm 9) and R-LWE (algorithm 6). Furthermore, only a single x_m value is required in M-LWE (as $n = 1$), which is then used to calculate both xu and xv, which are also single polynomials. Therefore, the entire calculation is identical to the R-LWE multiplication, which is a positive indication.

4.4 Criteria for comparing LWE-based homomorphic encryption schemes

In order to conduct a comparative analysis of homomorphic encryption schemes, the same methodology previously outlined in Section 3.4 can be employed to evaluate the underlying encryption and decryption mechanisms. As previously outlined, the model can be evaluated in terms of its theoretical robustness and practical viability in an operational context. As with the previous criteria, the size and time costs play a crucial role in the evaluation of HE schemes. However, to gain a comprehensive understanding of the relative efficiency of HE schemes, it is necessary to extend the analysis to encompass additional features. First, the relinearization key (RLK) must be included in the comparison of both costs. As an additional parameter that must be shared, the *size cost* of this parameter can be compared between the different LWE types. Furthermore, the additional time required for this calculation must be incorporated into the overall *time cost* of the key generation process, where the *rlk* generation takes place.
To assess the performance of the two new operations, addition and multiplication, additional *time cost* tests also need to be created. The performance differences between the various LWE types can be evaluated by simply running the operations multiple times. To minimize the impact of random variation, all tests should be conducted multiple times with identical parameter settings, and the median value should be used as the representative result. Given that the inputs and outputs are ciphertexts, it is not necessary to conduct an additional size cost test.

Furthermore, it is essential to ascertain the additive and multiplicative depth. This is the number of consecutive calculations that can be performed while maintaining the ability to successfully decrypt the ciphertext. To improve accuracy in prediction, it is necessary to examine the evolution of the errors associated with addition and multiplication. This allows for a more precise evaluation of the influence of the

variables on the outcome depth. This process may be accomplished by sequentially executing the desired operation and then verifying that the resulting decryption is consistent with the anticipated outcome.

5 Comparison of the SWHE scheme for Plain-, R- and M-LWE

In this chapter the SWHE scheme described in the previous chapter will be applied to the three LWE types, namely Plain-LWE, R-LWE and M-LWE, to compare them based on the criteria already described in section 3.4 and 4.4

When comparing the three LWE types, the primary distinctions is related to the two dimension variables, the matrix dimension n and the polynomial degree d. As outlined in previous sections, when $n > 1$ and $d = 1$, a Plain-LWE scheme is obtained. Conversely, when $n = 1$ and $d > 1$, the resulting scheme is R-LWE and when $n > 1$ and $d > 1$, the scheme is M-LWE. The first comparison will be based the size cost of the output variables, that are created when running the schemes. The size cost of the output variables can be compared based on the two dimension variables n and d and the moduli q and p. These output variables are the secret key sk, the private key pk, the relinearization key rlk and the ciphertext ct.
The second comparison will concentrate on the time cost of the algorithms for the different schemes. These algorithms are the *KeyGen*, the *Encryption*, the *Decryption*, the *Addition* and the *Multiplication*. To create a comparative analysis of the models performance, an example implementation in Python was constructed and the resulting data is used to evaluate the different schemes. As the same algorithm, with varying dimensions (n and d), can be utilized to describe all three schemes, the relative performance between them can be effectively assessed. However, given that Python is a relatively slow-performing language and the code is not optimized, the absolute values may not be entirely meaningful. Nevertheless, the relative performance between the schemes should remain comparable.
As a third comparison the additive and multiplicative depth will be compared to each other. These values show how often the operation can be redone, until decrypting the ciphertext is not longer working correctly, as the error has grown to big.

5.1 Size cost comparison

The initial comparison is the size in bits of the various output variables generated when working with the different LWE based HE schemes. The output is dependent on four variables: the matrix dimension n, the polynomial degree d and the number of bits of the modulus values, written as q_b and p_b.
In the case of Plain-LWE, the polynomial degree is equal to one, while in R-LWE, the matrix dimension is equal to one. For M-LWE, the polynomial degree and matrix dimension are greater than one. The size of the different variables can be calculated based on the dimensions defined in Table 1. As the rlk is essentially a modified private key, it has the same dimension; it is simply required n times. Additionally the modulus is bigger, as it incorporates the modulus p. All equations for computing the number of bits needed for the different variables for the schemes can be found in Table 4.

Table 4: LWE output variable size in bits based on n, d, q_b and ℓ

	Plain-LWE	R-LWE	M-LWE
sk	$n \cdot q_b$	$d \cdot q_b$	$n \cdot d \cdot q_b$
pk	$(n^2+n) \cdot q_b$	$2 \cdot d \cdot q_b$	$(n^2+n) \cdot d \cdot q_b$
rlk	$n \cdot ((n^2+n) \cdot (q_b+p_b))$	$2 \cdot d \cdot (q_b+p_b)$	$n \cdot ((n^2+n) \cdot d \cdot (q_b+p_b))$
ct	$(n+1) \cdot q_b$	$2 \cdot d \cdot q_b$	$(n+1) \cdot d \cdot q_b$

To provide a more intuitive understanding of the differences between the various schemes, a simulation of the values can be observed in Figure 2. The development of the number of factors is plotted for the different schemas based on n and d. The number of factors represents the amount of distinct variables that need to be stored for the output variable. It is equivalent to the equations in Table 4 if $q_b = 1$ or $q_b + p_b = 1$. When calculating the total storage size in bits, the number of factors can be multiplied with the number of bits needed for the moduli.
In the case of Plain-LWE, the matrix dimension, n, is plotted against the number of factors. In contrast, for R-LWE, the polynomial degree, d, is plotted against the number of factors. Given that M-LWE relies on two variables, namely n and d, it was decided that the best approach would be to plot n against the number of factors, with multiple lines representing different values of d. This was done to simplify the visualization of the differences, as a three-dimensional plot is more challenging to comprehend, particularly when printed in two dimensions. Additionally, the value of d is always linear in relation to the number of factors, indicating that it solely affects the slope of the change in n, as evidenced in the plot.

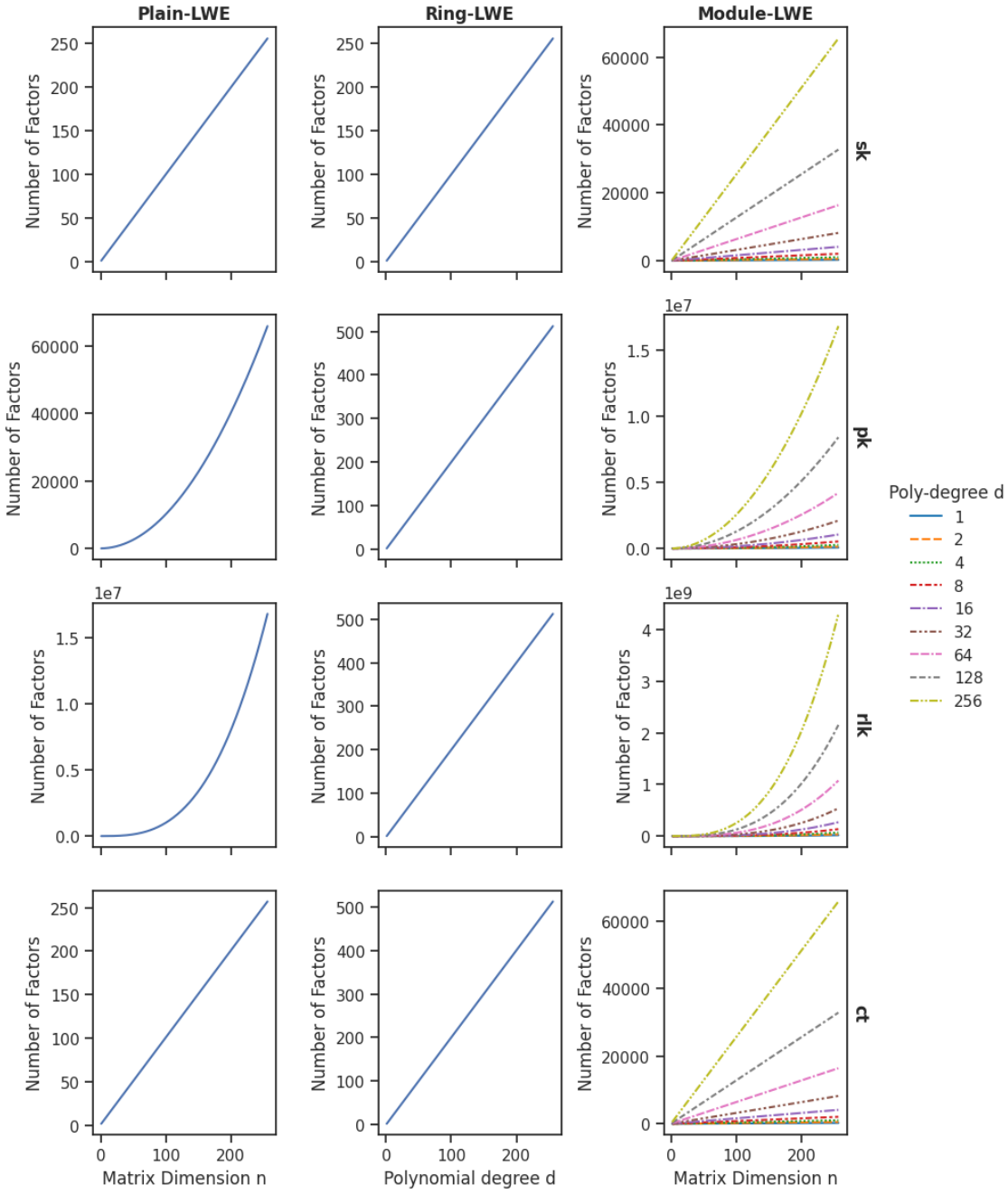

Figure 2: The graph illustrates the growth in the number of factors (y-axis) against the dimensions (x-axis). Each of the three columns of plots represents a distinct scheme, as indicated at the top of the figure. The rows represent the output variables, which are indicated on the right-hand side of the right plot. In Plain-LWE and M-LWE, the x-axis is represented by the variable n. In contrast, R-LWE employs the variable d. In the case of M-LWE, the diverse line types serve to differentiate between distinct values of d.

The growth rates observed for the secret key sk are linear with regard to the variables n and d for Plain-LWE and R-LWE, respectively. M-LWE exhibits a linear relationship with either n or d. When both variables grow simultaneously, the growth rate becomes quadratic. As a consequence, the growth rate for the number of factors in this case exhibits a significantly higher rate of increase than that observed for the other two schemes.

The linear growth rate observed for the sk in R-LWE remains for both the private key pk and the relinearization key rlk, but with an steeper incline. In contrast, the growth rates for the other two schemes exhibit a significant increase, particularly with regard to the matrix dimension n. For Plain-LWE the growth rate of the private key pk is quadratic, while for the relinearization key rlk, it is even cubic. When increasing the polynomial degree d simultaneously, the growth rate for M-LWE becomes cubic or quartic, respectively. It is also noteworthy that the rlk does not employ the conventional modulus q, as all the other parameters do, but rather the larger modulus $q \cdot p$. As both moduli are represented in the computer as binary numbers, the maximum space they require can be expressed as $2^{q_b} \cdot 2^{p_b} = 2^{q_b+p_b}$. Therefore, rather than requiring q_b bits, it is necessary to use q_b+p_b bits to represent each factor for the rlk. This contributes to the unfavorable growth rates observed in Plain-LWE and M-LWE, as more and significantly larger numbers are needed.

The ciphertext ct exhibits a similar growth pattern to that of the secret key sk. For Plain-LWE and M-LWE, the growth behavior is nearly identical, with the sole distinction being that, rather than n, $n + 1$ is utilized. This is negatable when n is increasing. As $n = 1$ for R-LWE, the increment of 1 results in a doubling of the requisite number of parameters, thereby causing the incline to double, but the growth rate remains linear.

One important note that need to be done, is the difference between word-wise and bit-wise FHE schemes [21]. The difference is, that bit-wise FHE operates at the single bit level and word-wise FHE operates on the whole word (eg, 64 bit). This means that each polynomial can be seen as a word and operations can be done all at once and the polynomial dimension d defines the word size. For a bit wise scheme, the logic needs to be done bit wise, which means that a lot more operations need to be done, which makes bit-wise slower. In this thesis, BFV was chosen as backbone for the created M-LWE HE scheme, which is a word-wise scheme, but by choosing $d = 1$ it is essentially turned into a bit-wise scheme. In practical terms, this implies that R-LWE and M-LWE are capable of encrypting an entire word at once, for example, 32- or 64-bit numbers. In contrast, Plain-LWE requires that each bit

be encrypted individually. This necessitates the decomposition of words into their constituent bits, followed by their encryption and the construction of operations based on these single-bit ciphertexts. This approach not only demands greater computational resources but also requires more space for storing the additional data. Further advantages and disadvantages are detailed in reference [21].

In general, R-LWE requires the fewest number of factors and the smallest amount of physical space to store its output variables. In comparison, Plain and especially M-LWE require a greater number of factors and a greater amount of physical space. Furthermore, the growth rate for R-LWE is significantly lower than that of Plain and M-LWE. In defense of M-LWE, it should be noted that in practice the matrix dimension n is quite small. In the CRYSTALS-Kyber scheme [4], the values are set between 2, which corresponds to low security, and 4, which corresponds to high security. This can be achieved because the security of the system is not solely dependent on the matrix, but also on the polynomials. In contrast to Plain-LWE, significantly larger values of n are required for security. In the Frodo encryption scheme [3], which utilize Plain-LWE, values between 352 for low security and 864 for high security and a value of $n = 752$ is recommended. This results in the number of factors exceeding 400 million just for the rlk.

A comparison of the physical space required for the various encryption schemes, with variables based on the regular/recommended security level of published encryption schemes that employ the underlying method, can be found in Table 5. It is important to note that the results should be interpreted with caution and that direct comparisons between the numbers are not possible. However, the data provides a general indication of the differences between the schemes. Notably, the variables for Plain-LWE are considerably larger than those for the other two, requiring more than 3 gigabytes, solely for the rlk. Even tho the ciphertext ct is the smallest for Plain-LWE, this amount of disk space is necessary for each bit, whereas R-LWE and M-LWE can store 512 or 256 bits, respectively, with a slightly larger ct. With the exception of the ciphertext, R-LWE has the smallest overall size for every variable. However, as it can store twice the information of M-LWE and, as previously mentioned, 512 times more than Plain-LWE, it requires the smallest amount per bit stored.

In conclusion, the comparison based on the size of the output values indicates that R-LWE has the slowest growth of physical space needed based on its dimension variables. Although M-LWE has the fastest growth rate, as it is based on multiple dimension variables, this makes it far more secure than Plain-LWE. Consequently,

Table 5: Theoretical LWE Output Variable size in Kilobyte (KB), based on variables for the regular/recommended security level of published encryption schemes. The modulus $p = q^3; p_b = q_b * 3$ as described in BFV

	Source	n	d	q	q_b	sk	pk	rlk	ct
Plain-LWE	[3]	752		32767	15	1.41	1061.73	3193684	1.41
R-LWE	[32]		512	25601	15	0.96	1.92	7.68	1.92
M-LWE	[4]	3	256	7681	13	1.25	4.992	59.904	1.66

the dimensions can be smaller in general to achieve a similar level of security, which results in a smaller physical size of its output variables. In terms of storage requirements for the output variables, Plain-LWE is by far the most demanding, with M-LWE requiring more physical space than R-LWE, except for the ciphertext. However, the difference in these requirements is not as significant as that observed in the comparison with Plain-LWE.

5.2 Time cost comparison

The objective of this section is to perform a comparative analysis of the runtime performance of the algorithms with respect to a selected set of input parameters. To minimize the impact of noise, multiple iterations of each algorithm were executed with the same inputs, and the median time value in seconds was calculated. In total, there are four parameters that can be modified: the dimensions n and d, as well as the modulus values q and p. To simplify the analysis of multiple variables and enhance the visualization of the differences, two performance tests were conducted. One test was conducted with varying dimension variables n and d, while the modulus values q and p were constant. The second test was performed with the variables n and d fixed while the modulus values q and p were varied. This should provide a more straightforward means of illustrating the impact of these variables on the performance of the algorithms. The algorithms that are evaluated are the *KeyGen* (Algorithm 10), the *Encryption* (Algorithm 2), the *Decryption* (Algorithm 3), the *Addition* (Algorithm 7) and the *Multiplication* (Algorithm 9).

The initial comparison is that of processing time, based on the two size dimensions, namely, n and d. The results of the comparative analysis of the algorithms' performance are presented in Figure 3.

In the figure, Plain-LWE is illustrated in the left column, in the line representing $d = 1$, and R-LWE is shown in the right column with $n = 1$. All other lines represent distinct versions of M-LWE. It can be observed that the *KeyGen* and *Multiplication*

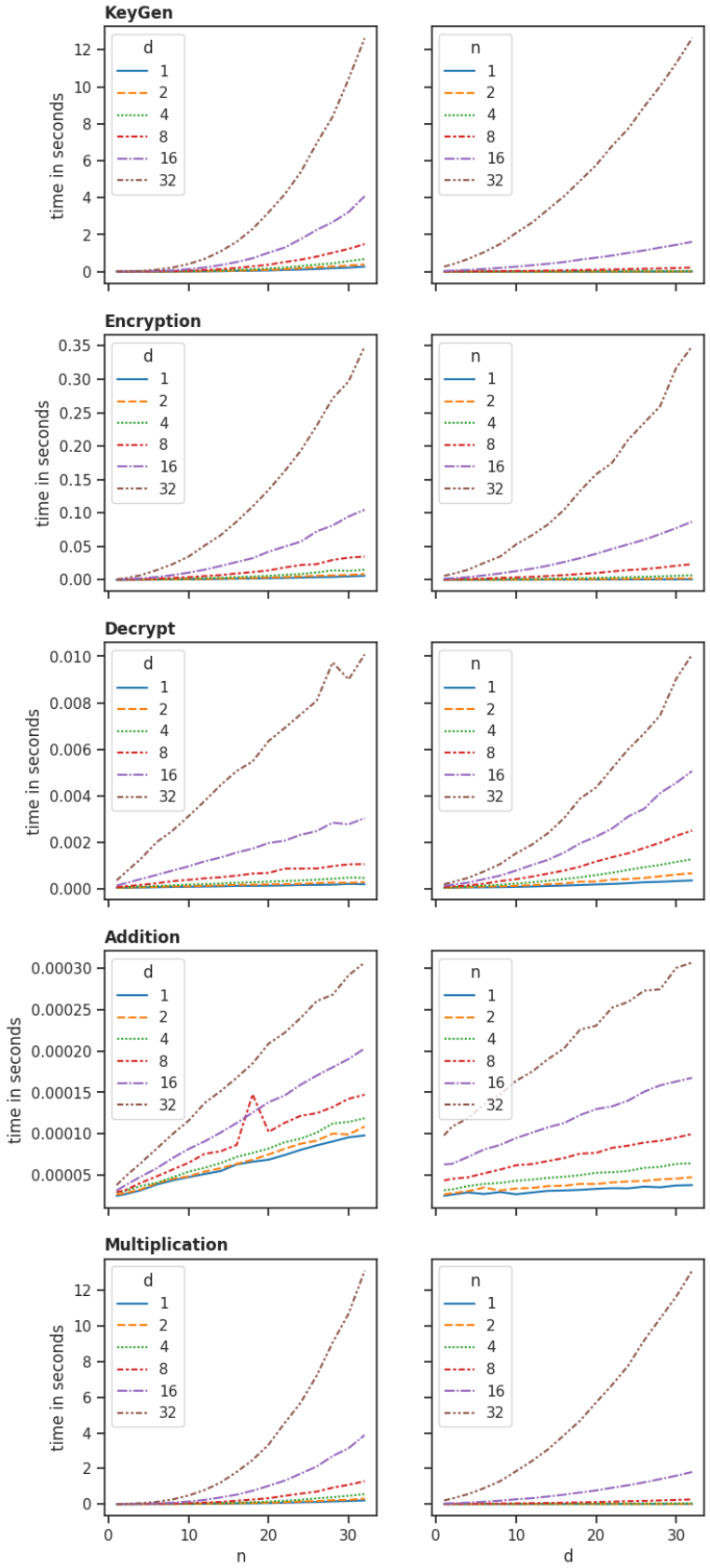

Figure 3: Performance of the main algorithms in regard to dimensions n and d and the time in seconds. Each row is a different algorithm, as stated always above the left plot. The left column plots the time against n for some d values. The right column plots the time against d for some n values.

algorithms are the most time-consuming. Regardless of whether n or d is increased, these algorithms exhibit an exponential growth in time cost. However, the rate of increase for d is relatively modest, in contrast to the significant rise observed in the case of n. This exponential growth can be attributed to the fact that, in order to perform matrix multiplication, the number of scalar multiplications increases quadratically. One potential solution to this issue is the use of dedicated matrix multiplication hardware, which performs the entire operation in a single step, or the incorporation of other optimized libraries. But, as it was necessary to work with numbers larger than 64 bits, it was not possible to use such libraries in the implemented system. These optimized libraries (such as numpy) only support 64-bit integers in python. Therefore, the arbitrary size integers in pure Python were used and a simple matrix multiplication algorithm was created, which is not optimized. This provides an explanation as to why the increase of n results in exponential growth, but also demonstrates that this phenomenon occurs at a slower rate when d is increased and n stays fixed. The rationale behind this is that, in order to perform polynomial multiplication, the methodology outlined in Section 2.4 was employed, which translates polynomial multiplication into matrix-vector multiplication. As the value of d increases, the dimensions of the matrix also increase, resulting in a quadratic growth in the number of multiplications required, following the same pattern previously described. The combination of these two facts also explains why the growth for n is more pronounced. In addition to the increase in the number of polynomial multiplications due to the larger dimensions, each new polynomial multiplication also exhibits a quadratic runtime. As the number of matrix multiplications increases, the algorithms become less efficient, exhibiting a decline in speed. This phenomenon is most evident in the *KeyGen* and *Multiplication*, where a single matrix multiplication is required for each rlk, which increases linearly with n. Consequently, these algorithms are quite slow. The *Encryption* algorithm requires only a single matrix-vector and one vector-vector multiplication, resulting in a quadratic growth rate. This makes it significantly faster than the previously mentioned algorithms. In contrast, *Addition* does not require any multiplication, making it the fastest algorithm by far. As only vectors need to be added, the time growth is also linear, as the number of additions scales linearly with the size of the vectors. With the *Decryption*, there is a very interesting pattern. While the runtime scales linearly with the increase of n, it has a quadratic growth with the increase of d. This is due to the quadratic scaling of the matrix required for the polynomial multiplication, as previously explained.

In consideration of the overall performance, the impact of the dimensions can re-

sult in a significant decline in performance, potentially by a factor of multiple and even orders of magnitude. Therefore, n and d exert a considerable influence on the performance.

The second cost comparison is based on the development of the modulus q and p, as illustrated in Figure 4. The quantity represented by the variable q_b in the graph is the number of bits for the modulus $q = 2^{q_b}$, while the quantity represented by the variable p_f is the factor by which the q bits are multiplied to retrieve the modulus $p = 2^{q_b \cdot p_f}$. The rationale for this is that the value of p should be a multiple of several orders of magnitude greater than that of q, as outlined in the BFV paper. Since the x-axis plots the q_b or p_f factors, the axis uses a logarithmic scale with respect to the actual modulus values.

Upon examination of the data, it becomes evident that there are two distinct groups of algorithms: KeyGen+Multiplication and Encryption+Decryption+Addition. The former exhibits a linear increase in time consumption, while the latter displays a pattern of steps and a constant time consumption between steps, as q_b increases. The primary distinction between these two groups is that the former requires the use of p for computations, whereas the latter employs solely q. This is also the reason why, for the right graphs associated with group one, there is a linear increase with the increase of p, whereas for group two, this value remains constant. Group two, on the left side, exhibits performance jumps around 16 and 32 bits. It is plausible that Python has internal improvements for 16 and 32 bit numbers. However, this is not observed in the first group, as they calculate with numbers greater than 32 bits due to the usage of p.

In general, it can be observed that the variables q and p exert a relatively minor influence on performance, particularly in comparison to the dimension variables. As evidenced by the data, even a significant discrepancy in q and p only results in a 40% increase in the time required for multiplication. Especially, when considering the range of p values, from $1 \cdot 4 = 4$ bits to $64 \cdot 8 = 512$ bits, which is an increase by factor 128.

To provide some real-world examples, the same parameters as in the previous section (see Table 5) were used to calculate the performance in seconds, which can be seen in Table 6. The Plain-LWE version was excluded from the comparison due to the discrepancy between the required memory and the available memory, which prevented its execution. Even if the limitation could be overcome, the runtime would be significantly slower than that observed for the other two versions. A

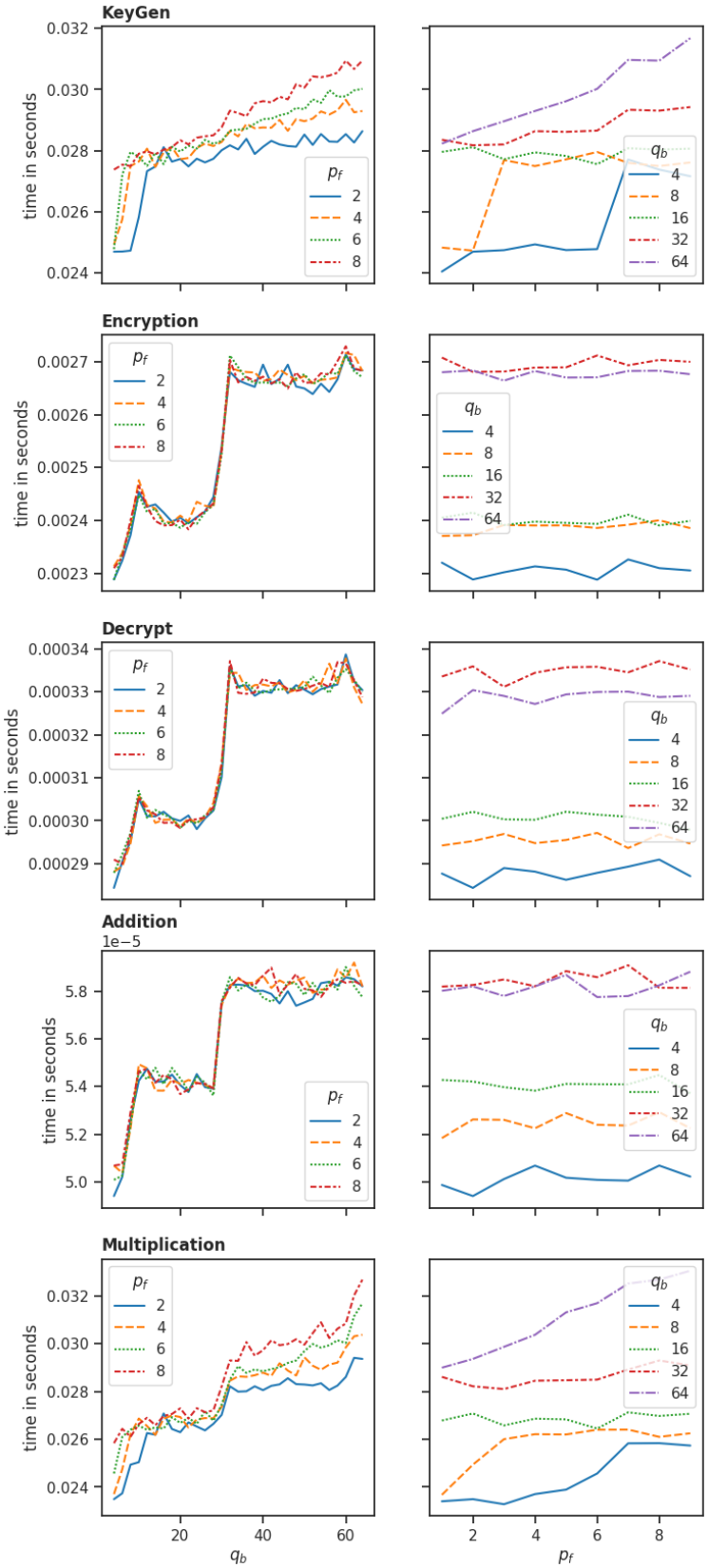

Figure 4: Performance of the main algorithms in regard to modulus with q_b bit and the modulus factor p_f, which represents $q_b \cdot p_f$ bit in regards to the time in seconds. Each row is a different algorithms, as stated always above the left plot. The left column plots the time against q_b for some p_f values. The right column plots the time against p_f for some q_b values.

comparison of the two other two schemes reveals that they both operate within the same order of magnitude. With the exception of the decryption algorithm, the R-LWE algorithms are consistently faster than their M-LWE counterparts. This discrepancy in decryption time may be attributed to the fact that R-LWE has twice the polynomial degree d, compared to M-LWE. But this also means, that R-LWE works with the double word size at the same time, which allows for bigger numbers to be operated on. Additionally, it is not unexpected that the M-LWE version is slower than R-LWE, as the higher matrix dimension, n, necessitates more calculations.

Table 6: M-LWE and R-LWE Performance in seconds, based on variables for the regular/recommended security level of published encryption schemes

	Source	Addition	Decrypt	Encryption	KeyGen	Multiplication
R-LWE	[32]	0.000163	0.061521	0.125041	0.182526	0.473052
M-LWE	[4]	0.000176	0.041224	0.174293	0.696122	1.039662

As previously demonstrated, the majority of algorithms exhibit a quadratic growth in runtime with respect to the dimension variables, namely n and d. This is growth rate is especially problematic for KeyGen, which has the advantage of being executed only once per session, and for multiplication, which is a significant challenge for homomorphic encryption schemes. In order to maintain satisfactory performance, it is advisable to minimise the dimensions, with particular attention paid to the matrix dimension n, which exhibits a higher growth rate and therefore a greater influence. In contrast, the influence of the modulus q and p on performance is comparatively minor, particularly in comparison to the dimension variables. Thus, increasing these variables results in a slight decline in performance, but not a drastic one. When examining real-world parameters, it becomes evident that R-LWE remains the best choice, closely followed by its M-LWE counterpart. Plain-LWE exhibits a considerable memory footprint, which would also result in a significant decline in performance, rendering it impractical to run this version at all in practice.

5.3 Comparison of the additive and multiplicative depth

As a last test, the additive and multiplicative depth of the LWE based HE schemes will be compared. As the number of operations performed in a homomorphic manner increases, the resulting error grows in magnitude. After a certain number of operations, the accumulated error becomes so significant that the decryption process yields incorrect numerical values. The source of the error is the ciphertext itself,

which is the foundation of LWE's security. This section will examine the maximum depth and the extent of the error growth for addition and multiplication operations, as well as the impact of dimension variables n and d, and modulus variables q and p.

The following statistical data were derived from a process whereby an operation was performed repeatedly with on a random ciphertext. The initial step involved encrypting a randomly generated message, thereby creating the original ciphertext. In each iteration, a new, randomly generated message was encrypted, resulting in a new, second ciphertext. Subsequently, the new, second ciphertext was added or multiplied, depending on the specific test, to the original message, thereby updating it. After that, the updated ciphertext was evaluated to ascertain whether the resulting message, obtained through decryption, exhibited the same outcome as if the calculations had been performed in plaintext space. The discrepancy between the present ciphertext and the correct plain text, the error, was recorded for each iteration. This process is then repeated until either the decrypted message does not match the expected result or until a pre-defined maximum limit is reached, as otherwise the procedure would take an excessive amount of time. For each parameter set, this process was repeated multiple times to reduce noise. The median values for each round were then calculated. The rounds are referred to as depth, which denotes the number of consecutive operations that can be performed.

First, the additive depth and error will be subjected to examination. The graphs illustrating this can be seen in Figure 5. The addition is dependent on three variables: the two dimensional variables and the modulus, q. The modulus p is not a required component of the addition process and therefore does not exert any influence on it. The figure illustrates the evolution of the error with depth for various configurations of the dimensions and for different modulus values. Once more, the value of q_b represents the exponent, and the utilized modulus is defined as $q = 2^{q_b}$. The error axis is logarithmic in order to facilitate the visualization of the comparative magnitude of the errors. A logarithmic curve is observed in nearly all of the graphs, indicating that the error grows in a linear fashion with depth. In the event that the value of q is insufficient, the curve will be terminated prematurely, as the resulting error is too high for successful decryption. As the value of q_b increases, the minimum error decreases, thereby enabling a greater number of operations. However, when n or d are increased, the minimum error rises due to the number of total factors and with that more error in the system. Accordingly, as the dimensions increase, a larger modulus must be selected to facilitate subsequent additions. The linear

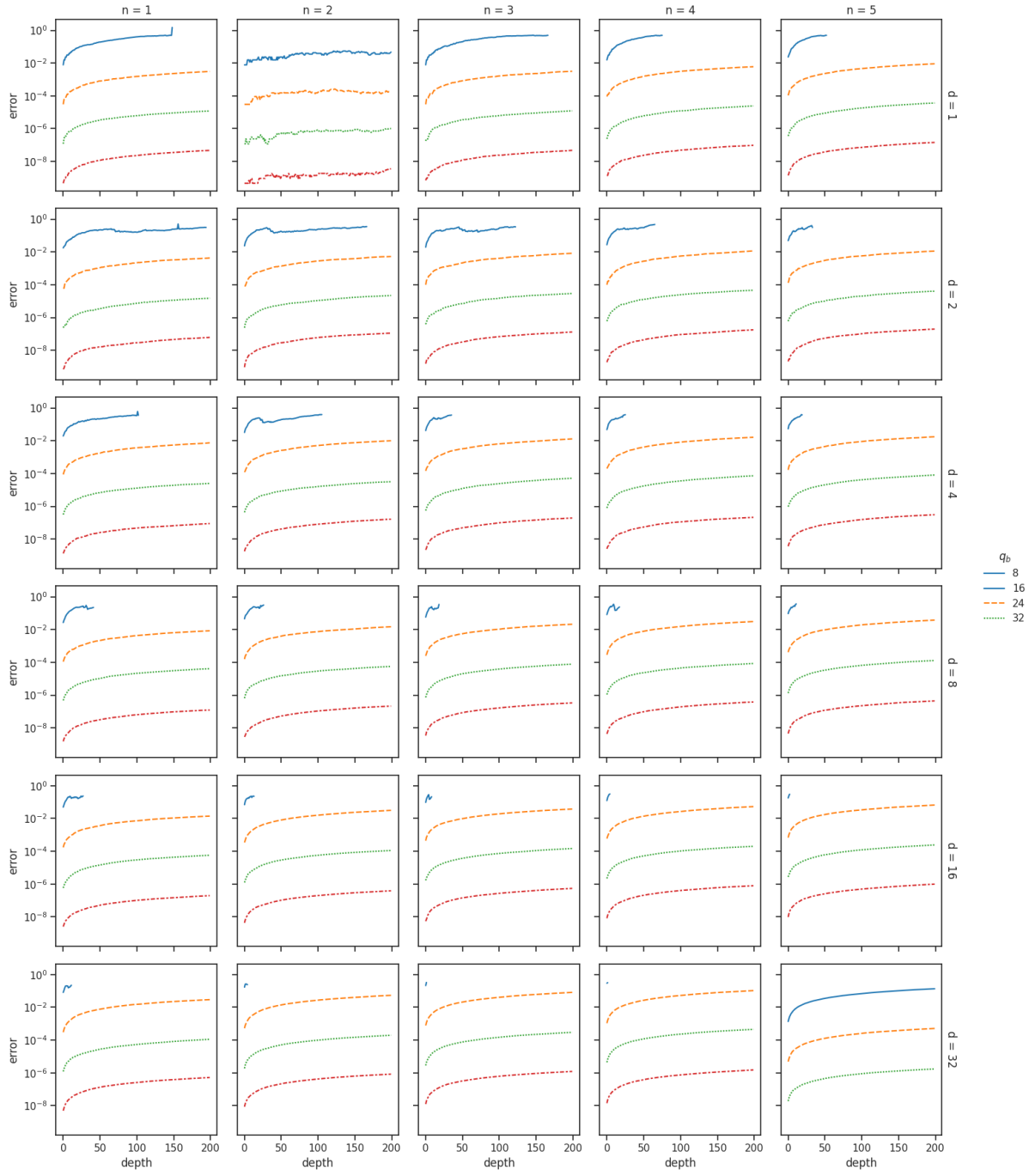

Figure 5: Additive Error Development

growth can be attributed to the fact that the errors resulting from the addition are simply added together, as outlined in algorithm 7. Therefore, the resulting error is equal to the sum of the two errors from the ciphertexts. In this experiment, only freshly encrypted data with a minimal error was added to one ciphertext. In practice, however, ciphertexts with larger errors will also be added, as they will have undergone previous operations. When the errors are combined, the growth rate will be faster than that shown in the graph. Nevertheless, the growth rate remains linear and thus manageable with a sufficiently large modulus. This makes it possible to run more then 200 additions easily, as can be seen by the error in the graphs. Also the maximum used modulus with $q = 2^{32}$, is not that high and bigger ones could be used without any problem making it possible to run many more additions in a row.

A comparison of the various LWE schemes reveals that the difference in computational depth for addition is not significant when evaluated based on depth alone. A comparison of the associated data indicates that the variables n and d exert a comparable influence on the reduction of computation depth. Consequently, an evaluation of the graphs suggests that Plain-LWE (first row) and R-LWE (first column) exhibit comparable performance for small dimensions. M-LWE, however, which incorporates both n and d into its formulation,results in a lower computational depth for addition.

The next step is to analyse the multiplication process, which is illustrated in Figure 6. The multiplication function makes use of the modulus p for the modulus switching component. The impact of this variable is illustrated by the colored bands displayed alongside each graph. It displays the minimum to maximum error values for all values of p at that depth. To obtain a comprehensive set of values, multiple runs were conducted with identical dimension variables and modulus and varying p_f values between 1 and 15. These p_f values were used to calculate the modulus $p = 2^{q_b \cdot p_f}$. The primary line represents the median values for a given depth across all p values.

In contrast to the behavior observed in the addition function, the error line in these graphs is, in general, more linear. As the error axis employs a logarithmic scale, the error grows exponentially with each multiplication. This results in a relatively shallow depth, particularly when the dimensions are increasing. To illustrate, the maximum depth attained in the bottom rightmost graph is 6, with an modulus of $q = 2^{80}$. The aforementioned error growth is also elucidated in other studies regarding LWE-based FHE schemes. For instance, in reference [5], it is detailed

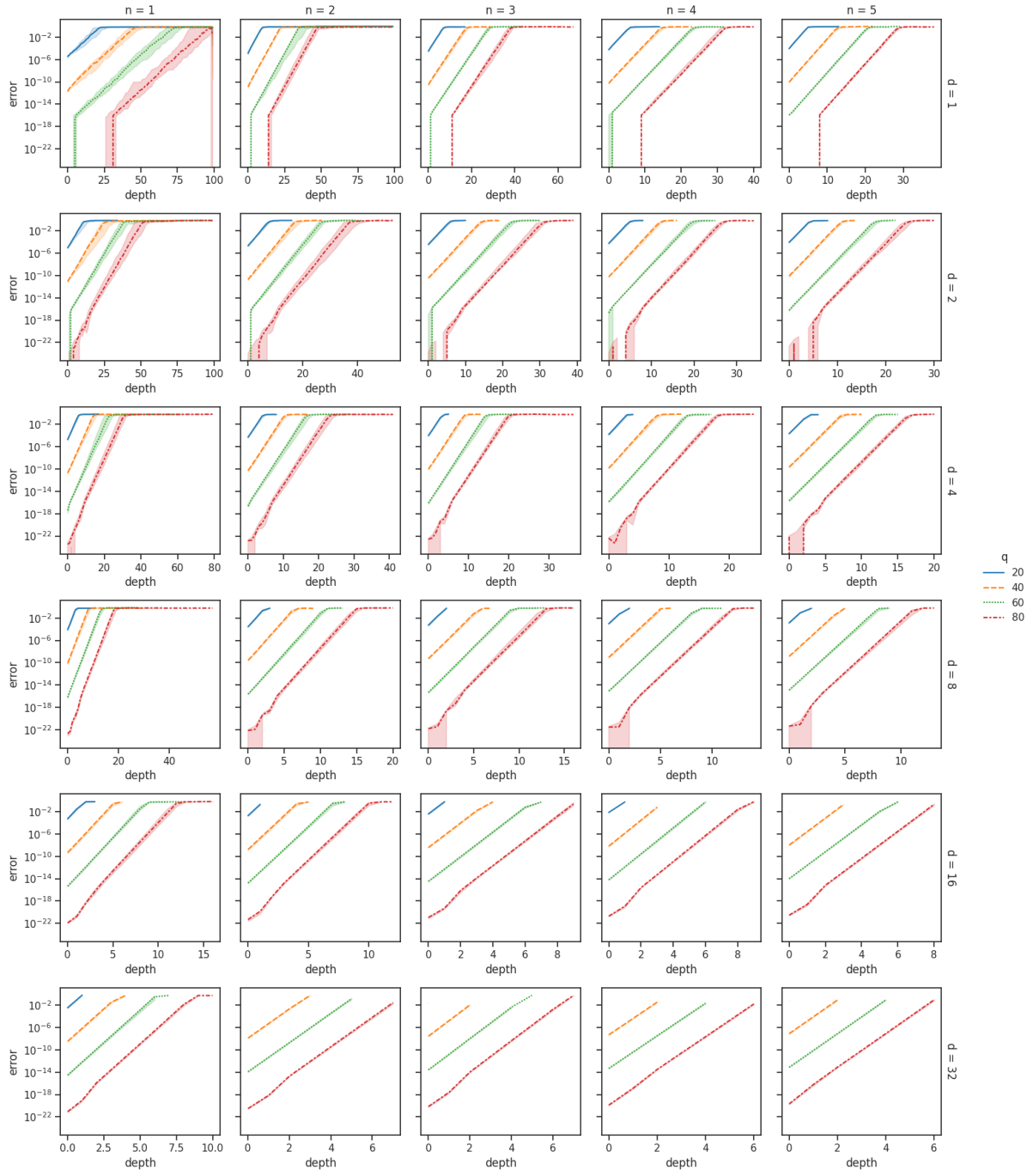

Figure 6: Multiplicative Error Development

that the error for addition increases by a maximum of two, while for multiplication, the error is quadratic in the worst case. These analogous growth rates are observable in the graphs for addition and multiplication.

The value of p is utilized for the modulus switching process, with the objective of enabling multiplication and concurrently reducing the error, as detailed in the BFV paper. As evidenced by the graphs, this objective was not achieved for any value of p. Furthermore, an additional trial was conducted, wherein the value of p was increased to a considerably higher amount, specifically 1000. This resulted in the generation of a modulus with a size of 2^{q*1000}. However, this approach did not yield any apparent improvement in the outcomes observed for the higher-dimensional M-LWE schemes, such as the one depicted in the bottom right region of the graph. The precise reason why p has no tangible impact is yet to be determined. One potential explanation for this discrepancy is the presence of numerical issues and minor rounding errors associated with the manipulation of these large numbers in Python. In particular, the process of dividing by p, as illustrated in steps 4 and 5 of algorithm 9, may introduce rounding errors due to the conversion from an arbitrary-size integer to a 64-bit float. This could potentially contribute to the observed issue.

When comparing the influence of the variables n and d and, consequently, the various schemes, on the multiplication error and depth, it becomes evident that, as was the case with addition, both variables exert a roughly equivalent influence. Therefore, irrespective of the variable that is increased, the computational depth rate will decline to a similar extent. As observed in the case of multiplication and addition, by enhancing the modulus q, the minimum error can be lowered, consequently leading to an increase in the computational depth.

Table 7: A comparison of the computation depth of M-LWE and R-LWE for variable modules, with $q = 2^{q_b}$ and $p = 2^{q_b \cdot 3}$. The remaining parameters are based on variables corresponding to the regular or recommended security level of published encryption schemes. The resulting numbers are the median values when an error occurred and thus no further calculations were possible.

	Addition						Multiplication					
q_b	13	15	20	32	64	128	13	15	20	32	64	128
R-LWE [4]	-	56	200	200	200	200	-	0	0	1	3	7
M-LWE [32]	7	-	200	200	200	200	0	-	0	1	3	7

As a final comparative analysis, the R-LWE and M-LWE algorithms were once again evaluated with the aid of a number of recommended parameters. The outcome of this

comparison is presented in Table 7. In order to evaluate the computational feasibility of the proposed approach, the experiment was conducted with increased modulus values. Both the M-LWE and R-LWE versions are capable of performing addition in their base configurations, however, the depth for the R-LWE is considerably higher than that for the M-LWE. This comes from the fact that in their base versions, the R-LWE version has a bigger modulus q then the M-LWE version. However, it was not possible to perform multiplication for any scheme with the original modulus values. As the value of q increased, the depth for addition could be easily raised to the maximum depth for this comparison (200) for both schemes in an equal manner. Achieving the desired increase in depth for multiplication proved more challenging. With a modulus of 128-bits, only an average of seven computations in a row were feasible. However, the values observed between the models are identical, indicating that the error increase is approximately equivalent across both R-LWE and M-LWE. In general it can be seen, that the R-LWE and M-LWE schemes are quite similar in additive and multiplicative depth for the same modulus q.

As discussed before, with a polynomial dimension d for R-LWE that is twice that of M-LWE, R-LWE is capable of processing significantly larger numbers.

6 Conclusion

The principal objective of this thesis was to investigate the viability of transferring R-LWE homomorphic encryption schemes to M-LWE and to evaluate the performance of these novel schemes. The viability of this approach was evaluated by developing an M-LWE HE scheme using the BFV scheme [12] as basis.

As detailed in Section 4.3, it is possible to extend R-LWE-based HE schemes to M-LWE, thereby creating novel HE schemes where not only the polynomial degree d but also the dimension size n of the matrix can be modified. This also permits the creation of Plain-LWE HE schemes, as these are a special case of M-LWE, where $d = 1$. To accomplish this, an M-LWE encryption scheme was extended to accommodate homomorphic addition and multiplication operations. In the case of the addition operation, this was a trivial process, as the values could be added together in the same manner as was done previously with R-LWE. The process for multiplication is somewhat more cumbersome and comprises a greater number of steps. The primary challenge was the generation of an $n \times n$ matrix during the multiplication of the ciphertexts. Given that the resulting ciphertext must retain its original structure, comprising a polynomial and a vector of polynomials, this matrix must be decomposed and incorporated into the other values. By decomposing this matrix and multiplying it with n relinearization keys, rather than a single one, the requisite output structure can be created, which, given a sufficiently large modulus q, will decrypt to the desired results. Given the general nature of the decomposition process, it seems reasonable to hypothesize that the same method could be applied with minor modifications to other R-LWE-based HE schemes.

In examining the findings of the evaluation, it becomes evident that the M-LWE scheme exhibits both advantageous and disadvantageous characteristics. Prior to an in-depth examination of R-LWE and M-LWE, it is worthwhile to briefly consider the characteristics of Plain-LWE. It is evident that a homomorphic version based on BFV cannot be used to construct a practical encryption scheme with Plain-LWE. The primary challenge lies in the substantial memory requirements for the variables (see Table 5), particularly the relinearization key, which is significantly higher than

what a practical implementation would allow. Even with future advancements in memory, networking for transmission, and computing power, the size will remain impractically large. Exchanging the key and performing operations with it will be computationally expensive, making it infeasible. Given that the other two methods are more effective, this finding is particularly noteworthy.

A comprehensive assessment of R-LWE and M-LWE reveals that it is not a straightforward task to reach a definitive conclusion. When comparing M-LWE and R-LWE based on their parameter sizes, R-LWE appears to exhibit superior characteristics. In contrast to M-LWE, the increase in parameters for R-LWE is always linear to the polynomial degree d. Conversely, when the matrix dimension, n, is increased in M-LWE, the private key, pk, and the relinearization key, rlk, increase exponentially, resulting in a significant increase in memory consumption. In practice, smaller polynomial degrees are employed for M-LWE, however, this does not offset the growth in parameters resulting from higher matrix dimensions. This is evident from Table 5. The issue with the accelerated growth rate for the private key and relinearization key is, that these keys must be exchanged prior to the initiation of encrypted communication. Given a fixed speed and capacity of networks, the establishment of an encrypted channel is more time-consuming with M-LWE, as a greater volume of data must be transmitted. However, the ciphertext is smaller for M-LWE than for R-LWE, at least in practical analysis. This implies that, with sufficient data, the initial higher cost can be offset and potentially reduced over the course of communication, allowing for the storage of more data with the same amount of disk space. In an enterprise setting, this could enhance the viability of M-LWE over R-LWE.

Another characteristic that seems superior in R-LWE in contrast to M-LWE is the processing time. As the calculations in R-LWE only depend on scalar polynomials, less operation need to be done compared to the vector and matrix processing of M-LWE. The big advantage of M-LWE is that the polynomial degree can stay fixed and improving the security can be done purely by increasing the matrix dimension. Therefore optimized multiplications can be implemented based on these fixed size polynomials, instead of re-implementing optimized versions for the different R-LWE security levels based on different polynomial degrees. Such matrix multiplication optimization are also currently being developed for the CRYSTALS-Kyber encryption scheme [4], where hardware solutions have been constructed already ([18], [17]). If these can be reused this would create new synergies between different M-LWE based encryption schemes. The value of these optimizations is debatable, as the optimized code can be written once and reused widely. Consequently, optimized

versions of R-LWE and M-LWE will be developed, which should result in faster R-LWE performance due to the lower number of required calculations.

The depth of operations for both the tested R-LWE and M-LWE scheme appear to be essentially equivalent, as shown in Table 7. It appears that the polynomial degree and matrix dimension exert a comparable influence on the outcome, suggesting that augmenting one while reducing the other results in outcomes that remain within a similar range. The most pronounced impact arises from the increase in modulus q, which is consistent across both versions.

Another significant distinguishing factor between these two that merits consideration, particularly when evaluating practical encryption protocols, is the word size. This refers to the number of bits that each of the schemes is capable of encrypting and working with in a single operation. In the practical examples, R-LWE utilises a polynomial degree, that is equal to the word size, of 512, while M-LWE employs a word size of 256. As the majority of computations and data structures operate within a 32- or 64-bit system, these word sizes are the most relevant in practical applications. Even when larger numbers are employed, 256- or 512-bit numbers are uncommon. One potential methodology for enhancing the M-LWE HE scheme would be to optimize it by reducing the polynomial degree (thereby reducing the word size) and increasing the matrix dimension in a manner that minimizes the ciphertext size while maintaining the same security level and avoiding an excessive increase in memory size of the private key and relinearization key. This increases the number of matrix multiplications, but because of the smaller polynomial degree, these matrices are smaller. If the total number of multiplications needed does not increase significantly, it would result in an even smaller ciphertext size, which would bring all the earlier-discussed benefits. Such an scheme could be more useful and efficient in a practical setting, where lots of ciphertext is created with the same keys.

The aforementioned points raise significant questions regarding the potential for superior performance of M-LWE over R-LWE. While it is evident that M-LWE can achieve comparable performance to R-LWE in practical scenarios, a definitive conclusion awaits the resolution of numerous open questions.
The SWHE scheme that was created could be extended to a full FHE scheme by applying bootstrapping. However, this process is already resource intensive, and it may become even more expensive in M-LWE, potentially limiting its usability. As an alternative, different already existing FHE schemes based on Plain or R-LWE could be considered as basis instead of BFV.

Furthermore, a more thorough examination of the security aspects could be conducted, which would enable the creation of a more comprehensive performance analysis, given that practical values for the various dimensions are then known. Additionally, it would be valuable to investigate the impact of increasing the modulus on the security. Given the substantial size of the modules, there is a possibility of reducing the dimensions.
A more practical study would be an improved implementation of this concept. One of the current problems is the difficulty of implementing this concept in Python, particularly when dividing large modulus values, due to rounding errors. Additionally, the matrix multiplication process is quite slow and requires optimization. A more detailed investigation into the potential for hardware acceleration using the research conducted for CRYSTALS-Kyber (as previously mentioned) and comparisons to enhanced R-LWE implementations could also yield valuable insights.

This leads to the conclusion that it is possible to change Plain and R-LWE encryption schemes to M-LWE. However, more research is needed to make statements about practical usability.

Bibliography

[1] Abbas Acar et al. "A Survey on Homomorphic Encryption Schemes: Theory and Implementation." In: *ACM Comput. Surv.* 51.4 (2018). ISSN: 0360-0300. DOI: 10.1145/3214303. URL: https://doi.org/10.1145/3214303.

[2] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. "Evaluating 2-DNF Formulas on Ciphertexts." In: *Theory of Cryptography Conference*. 2005. URL: https://api.semanticscholar.org/CorpusID:785423.

[3] Joppe Bos et al. *Frodo: Take off the ring! Practical, Quantum-Secure Key Exchange from LWE*. Cryptology ePrint Archive, Paper 2016/659. https://eprint.iacr.org/2016/659. 2016. DOI: 10.1145/2976749.2978425. URL: https://eprint.iacr.org/2016/659.

[4] Joppe Bos et al. *CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM*. Cryptology ePrint Archive, Paper 2017/634. https://eprint.iacr.org/2017/634. 2017. DOI: 10.1109/EuroSP.2018.00032. URL: https://eprint.iacr.org/2017/634.

[5] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. *Fully Homomorphic Encryption without Bootstrapping*. Cryptology ePrint Archive, Paper 2011/277. https://eprint.iacr.org/2011/277. 2011. URL: https://eprint.iacr.org/2011/277.

[6] Zvika Brakerski and Vinod Vaikuntanathan. *Efficient Fully Homomorphic Encryption from (Standard) LWE*. Cryptology ePrint Archive, Paper 2011/344. 2011. URL: https://eprint.iacr.org/2011/344.

[7] Zvika Brakerski and Vinod Vaikuntanathan. "Fully homomorphic encryption from ring-LWE and security for key dependent messages." In: *Proceedings of the 31st Annual Conference on Advances in Cryptology*. CRYPTO'11. Santa Barbara, CA: Springer-Verlag, 2011, pp. 505–524. ISBN: 9783642227912.

[8] Thi Van Thao Doan et al. "A survey on implementations of homomorphic encryption schemes." In: *The Journal of Supercomputing* 79.13 (Apr. 2023), pp. 15098–15139. ISSN: 1573-0484. DOI: 10.1007/s11227-023-05233-z. URL: http://dx.doi.org/10.1007/s11227-023-05233-z.

[9] Leo Ducas et al. *CRYSTALS – Dilithium: Digital Signatures from Module Lattices*. Cryptology ePrint Archive, Paper 2017/633. https://eprint.iacr.org/2017/633. 2017. URL: https://eprint.iacr.org/2017/633.

[10] Claudia Eckert. *It-Sicherheit*. de. 10th ed. de Gruyter Studium. de Gruyter, Aug. 2018.

[11] T. Elgamal. "A public key cryptosystem and a signature scheme based on discrete logarithms." In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472. DOI: 10.1109/TIT.1985.1057074.

[12] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2012/144. https://eprint.iacr.org/2012/144. 2012. URL: https://eprint.iacr.org/2012/144.

[13] Craig Gentry. "A fully homomorphic encryption scheme." In: 2009. URL: https://api.semanticscholar.org/CorpusID:53903759.

[14] *Google Quantum AI — quantumai.google*. https://quantumai.google. [Accessed 21-09-2024].

[15] Kimmo Halunen et al. "A Taxonomy of Metrics for Cryptographic Systems." English. In: Thirteenth International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2019, SECURWARE 2019 ; Conference date: 27-10-2019 Through 31-10-2019. 2019. URL: https://www.iaria.org/conferences2019/SECURWARE19.html.

[16] *IBM Quantum Computing — ibm.com*. https://www.ibm.com/quantum. [Accessed 21-09-2024].

[17] Arpan Jati et al. "A Configurable CRYSTALS-Kyber Hardware Implementation with Side-Channel Protection." In: *ACM Trans. Embed. Comput. Syst.* 23.2 (Mar. 2024). ISSN: 1539-9087. DOI: 10.1145/3587037. URL: https://doi.org/10.1145/3587037.

[18] Tendayi Kamucheka et al. *A Masked Pure-Hardware Implementation of Kyber Cryptographic Algorithm*. Cryptology ePrint Archive, Paper 2022/1547. https://eprint.iacr.org/2022/1547. 2022. URL: https://eprint.iacr.org/2022/1547.

[19] Adeline Langlois and Damien Stehle. *Worst-Case to Average-Case Reductions for Module Lattices*. Cryptology ePrint Archive, Paper 2012/090. https://eprint.iacr.org/2012/090. 2012. URL: https://eprint.iacr.org/2012/090.

[20] Adriana Lopez-Alt, Eran Tromer, and Vinod Vaikuntanathan. *On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2013/094. 2013. URL: https://eprint.iacr.org/2013/094.

[21] Wen-jie Lu et al. "PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption." In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, pp. 1057–1073. DOI: 10.1109/SP40001.2021.00043.

[22] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. *On Ideal Lattices and Learning with Errors Over Rings*. Cryptology ePrint Archive, Paper 2012/230. https://eprint.iacr.org/2012/230. 2012. URL: https://eprint.iacr.org/2012/230.

[23] Chiara Marcolla et al. *Survey on Fully Homomorphic Encryption, Theory, and Applications*. Cryptology ePrint Archive, Paper 2022/1602. https://eprint.iacr.org/2022/1602. 2022. DOI: 10.1109/JPROC.2022.3205665. URL: https://eprint.iacr.org/2022/1602.

[24] Anisha Mukherjee et al. *ModHE: Modular Homomorphic Encryption Using Module Lattices: Potentials and Limitations*. Cryptology ePrint Archive, Paper 2023/895. https://eprint.iacr.org/2023/895. 2023. URL: https://eprint.iacr.org/2023/895.

[25] *NIST Announces First Four Quantum-Resistant Cryptographic Algorithms*. July 2022. URL: https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms (visited on 01/03/2024).

[26] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes." In: *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[27] Daniel Plaumann. *Algebra*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2023.

[28] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography." In: *Symposium on the Theory of Computing*. 2005. URL: https://api.semanticscholar.org/CorpusID:53223958.

[29] R L Rivest, L Adleman, and M L Dertouzos. "On Data Banks and Privacy Homomorphisms." In: (1978).

[30] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems." In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: https://doi.org/10.1145/359340.359342.

[31] P.W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring." In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[32] Vikram Singh. *A Practical Key Exchange for the Internet using Lattice Cryptography*. Cryptology ePrint Archive, Paper 2015/138. https://eprint.iacr.org/2015/138. 2015. URL: https://eprint.iacr.org/2015/138.

List of Figures

List of Tables

List of Algorithms

A Example Calculations

A.1 Example Multidimensional Ring Calculation

Consider the ring $R = \mathbb{Z}_5[x]/(x^3+1)$ and

$$f(x) = \begin{bmatrix} 1+2x+3x^2 & 2+3x+4x^2 \\ 3+4x+x^2 & 1+3x+4x^2 \end{bmatrix} \in R^{2\times 2}$$

$$g(x) = \begin{bmatrix} 1+x+x^2 \\ 2+2x+2x^2 \end{bmatrix} \in R^2$$

$$\begin{aligned}
f(x)\cdot g(x) &= \begin{bmatrix} 1+2x+3x^2 & 2+3x+4x^2 \\ 3+4x+x^2 & 1+3x+4x^2 \end{bmatrix} \cdot \begin{bmatrix} 1+x+x^2 \\ 2+2x+2x^2 \end{bmatrix} \mod 5 \\
&= \left[\begin{bmatrix} 1 & -3 & -2 \\ 2 & 1 & -3 \\ 3 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & -4 & -3 \\ 3 & 2 & -4 \\ 4 & 3 & 2 \end{bmatrix} \right] \cdot \left[\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right] \mod 5 \\
&= \begin{bmatrix} 1 & -3 & -2 & 2 & -4 & -3 \\ 2 & 1 & -3 & 3 & 2 & -4 \\ 3 & 2 & 1 & 4 & 3 & 2 \\ 3 & -1 & -4 & 1 & -4 & -3 \\ 4 & 3 & -1 & 3 & 1 & -4 \\ 1 & 4 & 3 & 4 & 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \end{bmatrix} \mod 5 \\
&= \begin{bmatrix} -14 \\ 2 \\ 24 \\ -14 \\ 6 \\ 24 \end{bmatrix} \mod 5 = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 1 \\ 1 \\ 4 \end{bmatrix} = \left[\begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \\ 4 \end{bmatrix}\right] \\
&= \begin{bmatrix} 1+2x+4x^2 \\ 1+1x+4x^2 \end{bmatrix}
\end{aligned}$$

A.2 Example encryption with Plain-LWE

The following calculations show the working of the Plain-LWE encryption for the algorithms 1 to 3. The ring used for this calculations is defined as $R = \mathbb{Z}_{100}$ and $n = 2$ for the matrix dimension. Starting first with the key generation:

$$
\begin{aligned}
\mathbf{s} &= \begin{bmatrix} 1 \\ 2 \end{bmatrix} \mathbf{A} = \begin{bmatrix} 56 & 77 \\ 29 & 59 \end{bmatrix} \mathbf{e} = \begin{bmatrix} 99 \\ 1 \end{bmatrix} \\
\mathbf{b} &= \mathbf{A}\mathbf{s} + \mathbf{e} \\
&= \begin{bmatrix} 56 & 77 \\ 29 & 59 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 99 \\ 1 \end{bmatrix} \\
&= 1 \cdot \begin{bmatrix} 56 \\ 29 \end{bmatrix} + 2 \cdot \begin{bmatrix} 77 \\ 59 \end{bmatrix} + \begin{bmatrix} 99 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 309 \\ 148 \end{bmatrix}_{100} \\
&= \begin{bmatrix} 9 \\ 48 \end{bmatrix} \\
sk &= \mathbf{s} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\
pk &= (\mathbf{A}, \mathbf{b}) = \left(\begin{bmatrix} 56 & 77 \\ 29 & 59 \end{bmatrix}, \begin{bmatrix} 9 \\ 48 \end{bmatrix} \right)
\end{aligned}
$$

With the secret and public key generated, the next step is to encrypt the message $m = 1$ with the public key pk

$$\mathbf{r} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathbf{e}_1 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} e_2 = 99$$

$$\begin{aligned}
\mathbf{u} &= \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_1 \\
&= \begin{bmatrix} 56 & 77 \\ 29 & 59 \end{bmatrix}^T \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 56 & 29 \\ 77 & 59 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} \\
&= 0 \cdot \begin{bmatrix} 56 \\ 77 \end{bmatrix} + 1 \cdot \begin{bmatrix} 29 \\ 59 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 31 \\ 61 \end{bmatrix}_{100} = \begin{bmatrix} 31 \\ 61 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
v &= \mathbf{b}^T \cdot \mathbf{r} + e_2 + (m * \lfloor q/2 \rfloor) \\
&= \begin{bmatrix} 9 \\ 48 \end{bmatrix}^T \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 99 + 1 \cdot \lfloor 100/2 \rfloor \\
&= \begin{bmatrix} 9 & 48 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 99 + 50 \\
&= 9 \cdot 0 + 48 \cdot 1 + 99 + 50 \\
&= 197_{100} \\
&= 97 \\
ct &= (\mathbf{u}, v) = \left(\begin{bmatrix} 31 \\ 61 \end{bmatrix}, 97 \right)
\end{aligned}$$

Now the cipher text ct can be decrypted again, using the secret key sk:

$$\begin{aligned}
m &= \left\lfloor \frac{1}{\lfloor q/2 \rfloor} \cdot \left[v - \mathbf{s}^T \cdot \mathbf{u} \right]_q \right\rceil_2 \\
&= \left\lfloor \frac{1}{\lfloor 100/2 \rfloor} \cdot \left[97 - \begin{bmatrix} 1 \\ 2 \end{bmatrix}^T \cdot \begin{bmatrix} 31 \\ 61 \end{bmatrix} \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{1}{50} \cdot \left[97 - \begin{bmatrix} 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 31 \\ 61 \end{bmatrix} \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{1}{50} \cdot \left[97 - (31 \cdot 1 + 61 \cdot 2) \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{1}{50} \cdot \left[-56 \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{1}{50} \cdot 44 \right\rceil_2 \\
&= \left\lfloor \frac{44}{50} \right\rceil_2 = \lfloor 0.88 \rceil_2 \\
&= 1
\end{aligned}$$

A.3 Example encryption with R-LWE

The following calculations show the working of the Ring LWE encryption for the algorithms 1 to 3. The ring used for this calculations is defined as $R = \mathbb{Z}[x]_{100}/(x^3 + 1)$. Starting first with the key generation:

$$\begin{aligned}
s &= 1 + 0x + 1x^2 \\
A &= 28 + 56x + 1x^2 \\
e &= 1 + -1x + 2x^2 = 1 + 99x + 2x^2 \\
b &= As + e \\
&= (28 + 56x + 1x^2) \cdot (1 + 0x + 1x^2) + (1 + 99x + 2x^2) \\
&= (28 + 28x^2) + (56x + 56x^3) + (1x^2 + 1x^4) + (1 + 99x + 2x^2) \\
&= 29 + 155x + 31x^2 + 56x^3 + 1x^4 \mod x^3 + 1 \\
&= 29 + 155x + 31x^2 - 56 - 1x \\
&= -27 + 154x + 31x^2 \mod 100 \\
&= 73 + 54x + 31x^2 \\
sk &= s = 1 + 0x + 1x^2 \\
pk &= (A, b) = (28 + 56x + 1x^2, 73 + 54x + 31x^2)
\end{aligned}$$

Encryption of message $m = (1, 1, 0)$:

$$
\begin{aligned}
r &= 0 + 1 + 1x^2 \\
e_1 &= 98 + 0x + 98x^2 \\
e_2 &= 1 + 0x + 0x^2 \\
m &= (1, 1, 0) = 1 + 1x + 0x^2 \\
u &= A \cdot r + e_1 \\
&= (28 + 56x + 1x^2) \cdot (0 + 1 + 1x^2) + (98 + 0x + 98x^2) \\
&= (28x + 84x^2 + 57x^3 + x^4) + (98 + 0x + 98x^2) \\
&= 98 + 28x + 182x^2 + 57x^3 + x^4 \mod (x^3 + 1) \\
&= 98 + 28x + 182x^2 - 57 - x \\
&= 41 + 27x + 182x^2 \mod 100 \\
&= 41 + 27x + 82x^2 \\
v &= b \cdot r + e_2 + (m \cdot \lfloor q/2 \rfloor) \\
&= (73 + 54x + 31x^2) \cdot (0 + 1 + 1x^2) + (1 + 0x + 0x^2) + (1 + 1x + 0x^2) \cdot (100/2) \\
&= (73x + 127x^2 + 85x^3 + 31x^4) + (1 + 0x + 0x^2) + (50 + 50x + 0x^2) \\
&= 51 + 123x + 127x^2 + 85x^3 + 31x^4 \mod (x^3 + 1) \\
&= 51 + 123x + 127^2 - 85 - 31x \\
&= -34 + 92x + 127x^2 \mod 100 \\
&= 66 + 92x + 27x^2 \\
ct &= (u, v) = (41 + 27x + 82x^2, 66 + 92x + 27x^2)
\end{aligned}
$$

Decryption:

$$
\begin{aligned}
m &= \left\lfloor \frac{1}{\lfloor q/2 \rfloor} \cdot \left[v - s^T \cdot u \right]_q \right\rceil_2 \\
&= \left\lfloor \frac{1}{\lfloor 100/2 \rfloor} \cdot \left[(66 + 92x + 27x^2) - (1 + 0x + 1x^2) \cdot (41 + 27x + 82x^2) \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{1}{50} \cdot \left[(66 + 92x + 27x^2) - (41 + 27x + 123x^2 + 27x^3 + 82x^4) \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{1}{50} \cdot \left[25 + 65x - 96x^2 - 27x^3 - 82x^4 \right]_{100} \right\rceil_2 \mod (x^3 + 1) \\
&= \left\lfloor \frac{1}{50} \cdot \left[25 + 65x - 96x^2 + 27 + 82x \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{1}{50} \cdot (52 + 147x - 96x^2) \right\rceil_2 \mod 100 \\
&= \left\lfloor \frac{1}{50} \cdot (52 + 47x + 4x^2) \right\rceil_2 \\
&= \left\lfloor 1.04 + 0.64x + 0.08x^2 \right\rceil_2 \\
&= 1 + 1x + 0x^2 \\
&= (1, 1, 0)
\end{aligned}
$$

A.4 Example encryption with M-LWE

The following calculations show the working of the Module LWE encryption for the algorithms 1 to 3. The ring used for this calculations is defined as $R = \mathbb{Z}[x]_{100}/(x^3 + 1)$ and the matrix dimension $n = 2$.

Starting first with the key generation:

$$
\begin{aligned}
\mathbf{s} &= \begin{bmatrix} 2+1x+0x^2 \\ 3+1x+1x^2 \end{bmatrix} \\
\mathbf{A} &= \begin{bmatrix} 27+2x+43x^2 & 30+10x+35x^2 \\ 91+34x+50x^2 & 82+21x+94x^2 \end{bmatrix} \\
\mathbf{e} &= \begin{bmatrix} 1+1x+2x^2 \\ -3+3x+3x^2 = 97+3x+3x^2 \end{bmatrix} \\
\mathbf{b} &= \mathbf{A}\mathbf{s}+\mathbf{e} \\
&= \begin{bmatrix} 27+2x+43x^2 & 30+10x+35x^2 \\ 91+34x+50x^2 & 82+21x+94x^2 \end{bmatrix} \cdot \begin{bmatrix} 2+1x+0x^2 \\ 3+1x+1x^2 \end{bmatrix} + \begin{bmatrix} 1+1x+2x^2 \\ 97+3x+3x^2 \end{bmatrix} \\
&= \begin{bmatrix} 56+56x+233x^2 \\ 263+210x+519x^2 \end{bmatrix} + \begin{bmatrix} 1+1x+2x^2 \\ 97+3x+3x^2 \end{bmatrix} \\
&= \begin{bmatrix} 57+57x+235x^2 \\ 360+213x+522x^2 \end{bmatrix}_{100} \\
&= \begin{bmatrix} 57+57x+35x^2 \\ 60+13x+22x^2 \end{bmatrix} \\
sk = \mathbf{s} &= \begin{bmatrix} 2+1x+0x^2 \\ 3+1x+1x^2 \end{bmatrix} \\
pk = (\mathbf{A}, \mathbf{b}) &= \left(\begin{bmatrix} 27+2x+43x^2 & 30+10x+35x^2 \\ 91+34x+50x^2 & 82+21x+94x^2 \end{bmatrix}, \begin{bmatrix} 57+57x+35x^2 \\ 60+13x+22x^2 \end{bmatrix} \right)
\end{aligned}
$$

Encryption of message $m = (1, 1, 0)$:

$$\mathbf{r} = \begin{bmatrix} 1 + 1x + 1x^2 \\ 1 + 0x + 0x^2 \end{bmatrix}$$
$$\mathbf{e}_1 = \begin{bmatrix} 2 + 98x + 3x^2 \\ 97 + 3x + 3x^2 \end{bmatrix}$$
$$e_2 = 2 + 97x + 97x^2$$
$$m = (1, 1, 0) = 1 + 1x + 0x^2$$

$$\begin{aligned}
\mathbf{u} &= \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_1 \\
&= \begin{bmatrix} 27 + 2x + 43x^2 & 30 + 10x + 35x^2 \\ 91 + 34x + 50x^2 & 82 + 21x + 94x^2 \end{bmatrix}^T \cdot \begin{bmatrix} 1 + 1x + 1x^2 \\ 1 + 0x + 0x^2 \end{bmatrix} + \begin{bmatrix} 2 + 98x + 3x^2 \\ 97 + 3x + 3x^2 \end{bmatrix} \\
&= \begin{bmatrix} 27 + 2x + 43x^2 & 91 + 34x + 50x^2 \\ 30 + 10x + 35x^2 & 82 + 21x + 94x^2 \end{bmatrix} \cdot \begin{bmatrix} 1 + 1x + 1x^2 \\ 1 + 0x + 0x^2 \end{bmatrix} + \begin{bmatrix} 2 + 98x + 3x^2 \\ 97 + 3x + 3x^2 \end{bmatrix} \\
&= \begin{bmatrix} 73 + 20x + 122x^2 \\ 67 + 26x + 169x^2 \end{bmatrix} + \begin{bmatrix} 2 + 98x + 3x^2 \\ 97 + 3x + 3x^2 \end{bmatrix} \\
&= \begin{bmatrix} 75 + 118x + 125x^2 \\ 164 + 23x + 172x^2 \end{bmatrix}_{100} = \begin{bmatrix} 75 + 18x + 25x^2 \\ 64 + 23x + 72x^2 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
v &= \mathbf{b}^T \cdot \mathbf{r} + e_2 + (m \cdot \lfloor q/2 \rfloor) \\
&= \begin{bmatrix} 57 + 57x + 35x^2 \\ 60 + 13x + 22x^2 \end{bmatrix}^T \cdot \begin{bmatrix} 1 + 1x + 1x^2 \\ 1 + 0x + 0x^2 \end{bmatrix} + (2 + 97x + 97x^2) + (1 + 1x + 0x^2) \cdot \lfloor 100/2 \rfloor \\
&= (25 + 92x + 171x^2) + (2 + 97x + 97x^2) + (50 + 50x + 0x^2) \\
&= (77 + 239x + 268x^2)_{100} \\
&= 77 + 39x + 68x^2
\end{aligned}$$
$$ct = (\mathbf{u}, v) = \left(\begin{bmatrix} 75 + 18x + 25x^2 \\ 64 + 23x + 72x^2 \end{bmatrix}, 77 + 39x + 68x^2 \right)$$

Decryption:

$$
\begin{aligned}
m &= \left\lfloor \frac{1}{\lfloor q/2 \rfloor} \cdot \left[v - \mathbf{s}^T \cdot \mathbf{u} \right]_q \right\rceil_2 \\
&= \left\lfloor \frac{1}{\lfloor 100/2 \rfloor} \cdot \left[77 + 39x + 68x^2 - \begin{bmatrix} 2 + 1x + 0x^2 \\ 3 + 1x + 1x^2 \end{bmatrix}^T \cdot \begin{bmatrix} 75 + 18x + 25x^2 \\ 64 + 23x + 72x^2 \end{bmatrix} \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{1}{50} \cdot \left[77 + 39x + 68x^2 - (216 + 190x + 377x^2) \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{1}{50} \cdot \left[-139 - 151x - 309x^2 \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{1}{50} \cdot \left[61 + 49x + 91x^2 \right]_{100} \right\rceil_2 \\
&= \left\lfloor \frac{61}{50} + \frac{49}{50}x + \frac{91}{50}x^2 \right\rceil_2 = \left\lfloor 1.22 + 0.98x + 1.82x^2 \right\rceil_2 \\
&= \left\lfloor 1 + 1x + 2x^2 \right\rceil_2 = 1 + 1x + 0x^2 \\
&= (1, 1, 0)
\end{aligned}
$$