SCIENCE ▪ PASSION ▪ TECHNOLOGY

Peter Julius Waldert

# Secure Classification as a Service

Levelled Homomorphic, Post-Quantum Secure Machine Learning Inference
based on the CKKS Encryption Scheme

**BACHELOR'S THESIS**

Bachelor's degree programmes:
*Physics* and *Information & Computer Engineering*

**Supervisors**

Dipl.-Ing. Roman Walch

Institute of Applied Information Processing and Communications (IAIK)
Graz University of Technology

Graz, March 2022

# Abstract

Abstract of your thesis (at most one page)

TODO: To be written..

**Keywords:** FHE, ML, image classification, neural network, Private AI, PPML, Confidential Computing

**Technologies:** Microsoft SEAL (C++, nodejs), Tensorflow Keras, Numpy, xtensor, Docker, msgpack, React, Materialize, Nginx

**Languages:** C++, Python, JavaScript

# Contents

# Chapter 1

# Introduction

The most well-known and widely used asymmetric ('public-key') cryptographic scheme, published by the trio Rivest-Shamir-Adleman in 1977 and known as RSA, is based on the hardness assumption of the integer factorisation problem, factorising a large 2-composite number into its two prime factors $p$ and $q$ is believed to be hard (Rivest, Shamir and Adleman 1983). As of today, this factorisation problem has not been proven to be in the Non-deterministic Polynomial time (NP) complexity class, yet it is suspected that it might indeed be NP-complete (i.e. NP-hard while still being in NP) when modelled using a traditional Turing machine. Since the advent of quantum computation, this situation changed as a whole with Peter Shor's algorithm (Shor 1997), threatening the security of many cryptosystems, for instance Rivest-Shamir-Adleman (RSA) which is still widely used today despite its known problems.

As it stands, lattice-based cryptography presents a solution to a politically and socially problematic situation in which few parties world-wide, with access to a sufficiently powerful quantum computer, may be able to decrypt most of today's digital communication. Lattice Cryptography is based on other mathematical problems, shown to be sufficiently hard on quantum computers and traditional ones alike, most notably LWE which this thesis will discuss in detail.

Many new cryptosystems have been developed on top of LWE, two of which this following thesis will focus on specifically: BFV and CKKS; whose security is still unaffected by efficient quantum algorithms. Yet, it is not only their security prospect that makes these encryption schemes attractive, but first and foremost their defining homomorphic property which allows for computations on the encrypted data. A *fully* homomorphic encryption scheme was first introduced by Craig Gentry in 2009, using a bootstrapping approach. The *levelled* homomorphic Brakerski-Gentry-Vaikuntanathan (BGV) encryption scheme is implemented in Microsoft SEAL and allows for integer arithmetic, up to a few multiplication 'levels' deep. The Brakerski-Fan-Vercauteren (BFV) scheme is very similar to it and described in a bit more detail in section 3.4. And finally, building upon concepts introduced in the former, the Cheon-Kim-Kim-Song (CKKS) scheme allows for approximative floating-point arithmetic that finally facilitates machine-learning applications.

Machine Learning allows a computer to 'learn' from specifically structured data using linear regression or similar methods, and to apply this 'knowledge' to new, unknown inputs. In its simplest form, or even using a multi-layered neural network, this only requires two different operations on numbers (or even better, vectors): addition and multiplication. Using an Homomorphic Encryption (HE) scheme such as the ones mentioned above and described in chapter 3, both are given and Privacy-Preserving Machine Learning (PPML) applications are born!

The present thesis not only focusses on theoretical remarks but also includes a publicly available implementation of an HE classification server written in C++ and a compact graphical user interface to interact with.

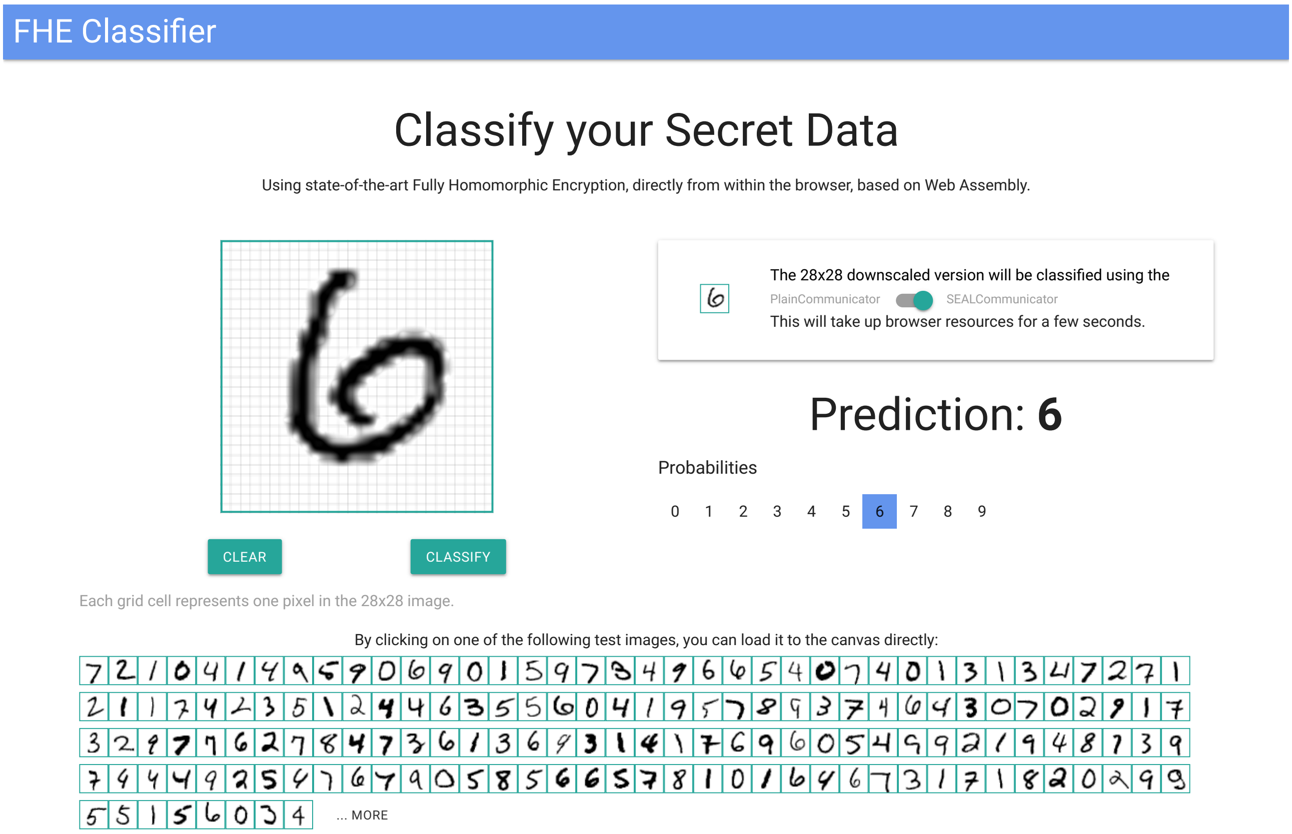


Figure 1.1: The user interface of the demonstrator, users can draw a digit by hand, select one of two communication means (plain or encrypted) and finally let the server handle the classification to obtain a prediction (including a visual of associated probabilities).

The following chapter 2 and chapter 3 aim to introduce most of the necessary theory to understand the HE schemes used in practice today, as well as the simple machine learning approaches involved in securely classifying images as a service.

Next, chapter 4 then focusses on the concrete system at hand, how the classification of handwritten digits (using the MNIST dataset) works in detail and what challenges arise when dealing with a system which acts not only on plain, but also encrypted data. The subsequent chapter 5 analyses the neural network performance in terms of its accuracy, digit-wise precision and recall, documents benchmarks of runtime, message size and accuracy and finally includes a visualisation of the ciphertext (containing all information about the original image).

# Chapter 2

# Background

The discussion of the HE schemes following in chapter 3 requires some mathematical background that will be introduced here, aiming for a consistent overview rather than full completeness. The last two sections introduce same background on Machine Learning and provide an outlook on Quantum Computation and why it affects cryptography today.

**Notational Conventions** Let $\mathbb{N}$ denote the natural numbers without 0, i.e. $\mathbb{N} = \{n \in \mathbb{Z} \,|\, n > 0\}$. For a probability distribution $\chi$ over a set $R$, let sampling a value $x \in R$ from the probability distribution be denoted by $x \leftarrow \chi$. For $a \in \mathbb{R}$ a real number, denote rounding down (floor) $a$ by $\lfloor a \rfloor \in \mathbb{Z}$, rounding up (ceil) by $\lceil a \rceil \in \mathbb{Z}$ and rounding to the nearest integer by $\lfloor a \rceil \in \mathbb{Z}$.

## 2.1 Polynomial Rings and Modular Arithmetic

As the algebraic structure underlying almost every single symbol following in the next chapters, we recall the definition of a ring:

**2.1.1 Definition (Ring)**

A tuple $(R, +, \cdot)$ consisting of a set $R$, an addition operation $+$ and a multiplication operation $\cdot$ is referred to as a ring, given that it satisfies the following *ring axioms*:

- Addition is closed: $a + b \in R \quad \forall a, b \in R$.
- Addition is commutative: $a + b = b + a \quad \forall a, b \in R$.
- Addition is associative: $(a + b) + c = a + (b + c) \quad \forall a, b, c \in R$.
- There exists an element $0 \in R$ such that $a + 0 = a \quad \forall a \in R$.
- An additive inverse $-a$ of each element $a$ in $R$ exists, such that $a + (-a) = 0$.
- Multiplication is associative: $(a \cdot b) \cdot c = a \cdot (b \cdot c) \quad \forall a, b, c \in R$.
- Multiplication is closed: $a \cdot b \in R \quad \forall a, b \in R$.
- There exists an element $1 \in R$, referred to as the identity element, or multiplicative identity of $R$, such that $a \cdot 1 = a \quad \forall a \in R$.
- Multiplication $\cdot$ is distributive w.r.t. addition $+$,
  i.e. $a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad \forall a, b, c \in R$ from the left and
  i.e. $(b + c) \cdot a = (b \cdot a) + (c \cdot a) \quad \forall a, b, c \in R$ from the right.

Where the first 5 properties can be summarised as $(R, +)$ forming an Abelian group. If multiplication is additionally commutative, we refer to the ring as *commutative*:

- Multiplication is commutative: $a \cdot b = b \cdot a \quad \forall a, b \in R$.

Acting as the logical extension of a group, a ring can be considered the intermediary step towards a field (which also defines subtraction and division). An example of a ring would be the integers themselves, or the integers modulo $t$: $\mathbb{Z}/t\mathbb{Z}$, sometimes also denoted as $\mathbb{Z}_t$.

Given two groups $(G, +)$ and a subgroup $(N, +)$, we can construct another group $G/N$ as follows, referred to as a quotient group, or factor group:

**2.1.2 Definition (Quotient Group / Ring)**

A quotient group $(G/N, +)$ (pronounced '$G$ mod $N$') over the original group $G$ and a normal subgroup $N$ of $G$ with a standard element operation $+$ can be defined using the left cosets

$$g + N := \{g + n \,|\, n \in N\} \subseteq G$$

of $N$ in $G$. The corresponding set $G/N$ is defined as

$$G/N := \{g + N \,|\, g \in G\}$$

whereas the standard operation $+ : G/N \times G/N \mapsto G/N$ can be extended from the original group $G$ as follows:

$$(g + N) + (h + N) := (g + h)N$$

The quotient set $G/N$ can therefore be identified as the set of all possible left cosets $g + N$ that in union reconstruct the original group $G$.

As a highly relevant structure to cryptography and a great example of a quotient group, we would like to consider the ring of integers modulo a given modulus $q \in \mathbb{N}$.

**2.1.1 Lemma (Ring of Integers Modulo $q$: $\mathbb{Z}/q\mathbb{Z}$)**

Using equivalence classes $\overline{x}_q$ modulo $q$ referred to as congruence classes, define the commutative quotient ring of integers modulo $q$ as $(\mathbb{Z}/q\mathbb{Z}, +, \cdot)$ with two operations $+$ and $\cdot$ and

$$\mathbb{Z}/q\mathbb{Z} = \{\overline{x}_q \,|\, x \in \mathbb{Z}, 0 \leq x < q\}$$

where $q\mathbb{Z} = \{qx \,|\, x \in \mathbb{Z}\} \triangleleft \mathbb{Z}$ (where $\triangleleft$ refers to the left being a subgroup of the right) denotes the $q^{\text{th}}$ multiplicative coset[a] of the integers and

$$\overline{x}_q = \{y \equiv x \mod q \,|\, y \in \mathbb{Z}\}$$

is the set of all multiples of $q$ with remainder $x$. Note that many operations that resulting groups, rings or fields are commonly equipped with, such as addition or multiplication, propagate to an equivalent definition in the ring of integers modulo $q$ by considering their result as a congruence class instead of it, which in turn is again an element of $\mathbb{Z}/q\mathbb{Z}$.

[a] from the left and from the right, therefore $q\mathbb{Z}$ is called a normal subgroup of $\mathbb{Z}$

This ring is of specific importance in discrete mathematics and can be regarded as a formalisation of modular arithmetic, much of which we will require at a later point in this chapter.

As a first step towards the first central result, Corollary 2.1.1, we formally introduce polynomial rings and how to carry out addition and multiplication between them.

**2.1.3 Definition (Polynomial Ring over $\mathbb{Z}$)**

On the set of all complex-valued polynomials with integer coefficients (a function space)

$$\mathbb{Z}[X] = \Big\{p : \mathbb{C} \mapsto \mathbb{C}\,, p(x) = \sum_{k=0}^{\infty} a_k x^k, a_k \in \mathbb{Z}\ \forall k \geq 0\Big\},$$

we can define a commutative ring $(\mathbb{Z}[X], +, \cdot)$ equipped with the standard addition $+$ and multiplication $\cdot$ operations (as an extension over the field $\mathbb{C}$) of polynomials.

To further elaborate on the polynomial ring operations:

- In their coefficient representations $\boldsymbol{p} = (p_j)_{j\in\mathbb{N}} = (p_0, p_1, p_2, ...)$ (which are sequences) and $\boldsymbol{q} = (q_j)_{j\in\mathbb{N}} = (q_0, q_1, q_2, ...)$, an addition of two polynomials $p, q \in \mathbb{Z}[X]$ is equivalent to the element-wise addition of their coefficient sequences

$$\begin{aligned}(p+q)(X) &= \sum_{k=0}^{\infty} p_k X^k + \sum_{k=0}^{\infty} q_k X^k = \sum_{k=0}^{\infty}(p_k + q_k)X^k \\ &= \langle(\boldsymbol{p}+\boldsymbol{q}), \{X^0, X^1, X^2, ...\}^T\rangle\end{aligned}$$

  which indeed satisfies the additive ring axioms due to the existing structure of the underlying field $\mathbb{C}$.

- The multiplication operation can be defined using a discrete convolution of the coefficient vectors

$$r(X) = (p \cdot q)(X) = (\sum_{k=0}^{\infty} p_k X^k) \cdot (\sum_{l=0}^{\infty} q_l X^l) = \sum_{k=0}^{\infty}\sum_{l=0}^{\infty} p_k q_l X^{k+l} = \sum_{k=0}^{\infty} r_k X^k$$

  with the arising coefficients $(r_k)_{k\in\mathbb{N}}$ determined by the discrete convolution

$$r_k = \sum_{l=0}^{k} p_l q_{k-l} \iff \boldsymbol{r} = \boldsymbol{p} * \boldsymbol{q}$$

  in this context also referred to as the Cauchy-product. Therefore,

$$(p \cdot q)(X) = \langle(\boldsymbol{p} * \boldsymbol{q}), \{X^0, X^1, X^2, ...\}^T\rangle.$$

  Again, this generally applicable approach satisfies the multiplicative ring axioms and even satisfies commutativity due to the existing structure of the underlying field $\mathbb{C}$ and the symmetry of convolutions.

Where $\langle\cdot,\cdot\rangle$ denotes the dot (scalar) product between two vectors.

Polynomials with degree $\geq 1$ over the complex numbers can always be factorised using their roots due to the fundamental theorem of algebra. Polynomials over the integers however, cannot always be factorised further, yielding the definition of an irreducible polynomial.

**2.1.4 Definition (Irreducible Polynomials)**

A polynomial is called irreducible if and only if (iff) it cannot be written as a product of other polynomials *while staying in the same coefficient space*.

### 2.1.1 Cyclotomic Polynomials

Due to their interesting structure and efficient computability, in the schemes introduced in the following chapter, certain polynomials (Corollary 2.1.1) are chosen as representations of plaintexts and ciphertexts. An important concept is that of cyclotomic ('circle-cutting') polynomials, which we will discuss in a bit more detail here.

An important polynomial is

$$p : \mathbb{C} \mapsto \mathbb{C},\ p(x) = x^n - 1\,.$$

Its roots, found by solving $p(x) = 0$ for $x$, yielding $x^n = 1 \leftrightarrow x_k = \sqrt[n]{1}$ are referred to as the $n^{\text{th}}$ roots of unity, of which there are multiple for each $n \in \mathbb{N}$.

**2.1.2 Lemma (The $n^{\text{th}}$ roots of unity)**

For some integer $n \in \mathbb{N}$, the $n$ complex roots $x_1, x_2, ..., x_n \in \mathbb{C}$ of unity can be found as

$$x_k = e^{2\pi i \frac{k}{n}} \quad k \in \{1, 2, ..., n\}$$

with $i$ the imaginary unit. Confer Figure 2.1. Using Euler's identity, their real and imaginary components can be explicitly found as $x_k = \cos\left(2\pi \frac{k}{n}\right) + i \sin\left(2\pi \frac{k}{n}\right)$.
An $n^{\text{th}}$ root of unity $y$ is referred to as *primitive*, iff there exists no $m < n$ for which that root $y$ is also an $m^{\text{th}}$ root of unity, i.e. $y^m \neq 1$. An equivalent indicator of a primitive root is $\gcd(m, n) = 1$, referring to the greatest common divisor between $m$ and $n$ which is 1 iff they are mutually prime.

Due to the fact that for any $k, l \in \mathbb{Z}$, their product $x_k \cdot x_l$ is also a root of unity, and $x_{k+jn} = x_k\ \forall j \in \mathbb{Z}$, they clearly comprise a cyclic Abelian group over the complex numbers $\mathbb{C}$ under multiplication with (for instance) the first root $x_1 = e^{2\pi i \frac{1}{n}}$ as its generator.

**2.1.5 Definition (Cyclotomic Polynomial)**

Given the $n^{\text{th}}$ roots of unity $\{x_k\}$, we can define the $n^{\text{th}}$ cyclotomic polynomial $\Phi_n \in \mathbb{Z}[X]$ as the product over all primitive roots of unity

$$\Phi_n(x) = \prod_{\substack{k=1 \\ x_k \text{primitive}}}^{n} (x - x_k)\,.$$

It is unique for each given $n \in \mathbb{N}$.

The number of primitive roots of unity is given by $\varphi(n)$, denoting Euler's totient function which counts the natural numbers $m$ less than $n$ who do not share a common divisor $\neq 1$, Placeholder

Figure 2.1: The $5^{\text{th}}$ roots of unity visualised on the complex plane. Obviously, they all lie on the unit circle $|z| = 1$, motivating the name of cyclotomic, 'circle-cutting', polynomials, whose roots cut the unit circle into multiple sectors.

i.e. $\gcd(m, n) = 1$. $\varphi(n)$ therefore also counts the number of primitive roots of unity for $n$, consequently also yielding the degree of the $n^{\text{th}}$ cyclotomic polynomial.

An important aspect of cyclotomic polynomials is that they are irreducible over their coefficient space, the integers $\mathbb{Z}$.

**2.1.1 Remark (Irreducibility of Cyclotomic Polynomials)**

Cyclotomic polynomials are always irreducible.

This enables us to *uniquely* define a quotient ring with cyclotomic polynomials as moduli, later. In theory, there are multiple equivalent definitions of said ring, but by convention we choose the cyclotomic polynomial because it cannot be simplified further. The proof for Remark 2.1.1 is quite cumbersome, but can be found in Serge 2002.

**2.1.1 Theorem ($2^{k\text{th}}$ cyclotomic polynomial)**

The $m^{\text{th}}$ cyclotomic polynomial, where $m = 2n = 2^k$ $(k \in \mathbb{N})$ is a power of 2, can be identified as

$$\Phi_m(x) = x^n + 1\,.$$

Its degree is $n$, consistent with $\varphi(2^k) = 2^{k-1}\ \forall k \in \mathbb{N}$.

Find a short but illustrative proof of Theorem 2.1.1 in the Appendix.

**2.1.6 Definition (Ring of Polynomials of highest degree $N - 1$)**

One can construct the quotient ring $(R, +, \cdot)$ as

$$R = \mathbb{Z}[X]/(X^N + 1)$$

where $(X^N + 1)$ denotes the set of all polynomial multiples of the polynomial $p \in \mathbb{Z}[X], p(x) = x^N + 1$, so

$$(X^N + 1) = \{q : \mathbb{C} \mapsto \mathbb{C},\ q(x) = r(x) \cdot (x^N + 1) \mid r \in \mathbb{Z}[X]\}\,.$$

> The elements of $R$ are then polynomials with integer coefficients of maximum degree $N-1$.

If $N$ is a power of 2, according to Theorem 2.1.1, $R$ is the set of integer-coefficient polynomials reduced modulo $\Phi_d(X)$ the $d^{\text{th}}$ cyclotomic polynomial with $N = \varphi(d) = \frac{d}{2}$, where $R$ is

$$R = \mathbb{Z}[X]/\Phi_d(X) = \mathbb{Z}[X]/(X^N + 1)\,.$$

Since every cyclotomic polynomial is irreducible, this is a unique representation of $R$ without any possible further simplifications.

As promised above, we will require Lemma 2.1.1 for the fundamental structure underlying the HE schemes described in the next chapter, defining ourselves a ring with coefficients in said quotient ring $\mathbb{Z}/q\mathbb{Z}$.

**2.1.1 Corollary (Polynomial Ring modulo $q$)**

Further modifying $R = \mathbb{Z}[X]/(X^N + 1)$ to only take coefficients mod $q$, we obtain two equivalent definitions for the same ring:

$$R_q = R/qR = (\mathbb{Z}/q\mathbb{Z})[X]/(X^N + 1)$$

which contains polynomials with integer coefficients modulo $q$ of degree $N-1$. Explicitly stated, the set can be written as:

$$R/qR = \{p : \mathbb{C} \mapsto \mathbb{C},\ p(x) = \sum_{k=0}^{N-1} a_k x^k \mid a_k \in \mathbb{Z}/q\mathbb{Z}\}$$

This bounded polynomial ring is central to understanding objects in the next chapter and Corollary 2.1.1 can be regarded as the central result of this section.

## 2.2 Lattice Cryptography

Lattice-based cryptography takes a different approach to encryption than classical factorisation or the discrete logarithm problem, as it is based on different problems, namely ones on lattices. The goal of any mathematical encryption scheme is to leave a potential attacker with a computationally hard, at best infeasible, problem to solve when attempting to decrypt messages without a secret key. This section will start with three basic problems, SVP, GapSVP and SIS and move on to Learning With Errors (LWE) and Learning With Errors on Rings (RLWE). To illustrate the connection of these problems to lattices, we take a closer look at them before considering further details of LWE. Most notably, lattice problems are conjectured to be secure against quantum computers (Corrigan-Gibbs, S. Kim and Wu 2018).

**2.2.1 Definition (Lattice)**

A lattice $(\mathcal{L}, +, \cdot)$ is a vector field over the integers $(\mathbb{Z}, +, \cdot)$, defined using a set of $n$ basis vectors $\boldsymbol{b_1}, \boldsymbol{b_2}, ..., \boldsymbol{b_n} \in \mathbb{R}^n$, that can be introduced as a set

$$\mathcal{L} = \left\{ \sum_{i=1}^{m} c_i \boldsymbol{b}_i \,\middle|\, c \in \mathbb{Z} \right\} \subseteq \mathbb{R}^n$$

equipped with at least vector addition $+ : \mathcal{L} \times \mathcal{L} \mapsto \mathcal{L}$ and scalar multiplication $\cdot : \mathbb{Z} \times \mathcal{L} \mapsto \mathcal{L}$. As an extension of $\mathbb{R}^n$, the Euclidean norm $||\cdot||$ is also defined and the standard Euclidean metric $d : \mathcal{L} \times \mathcal{L} \mapsto \mathbb{R}$, yielding a metric space $(\mathcal{L}, d)$, can be obtained by the norm of a vector difference, denoted $||(\cdot) - (\cdot)||$.

Its *minimum distance* $\lambda_{min}$ is defined as the smallest Euclidean distance between two points $\boldsymbol{p_1}$ and $\boldsymbol{p_2} \in \mathcal{L}$

$$\lambda_{min} = \min_{\boldsymbol{p_1}, \boldsymbol{p_2} \in \mathcal{L}} d(\boldsymbol{p_1}, \boldsymbol{p_2}) = \min_{\boldsymbol{p_1}, \boldsymbol{p_2} \in \mathcal{L}} ||\boldsymbol{p_1} - \boldsymbol{p_2}|| \,,$$

which can be equivalently thought of as the minimal length of any non-zero vector in the lattice $\mathcal{L}$, because of $\mathbf{0}$ always being an element of the lattice which can be chosen as $\boldsymbol{p_1}$ and the translational symmetry between fundamental lattice volumes (or regions).

The three problems frequently showing up in cryptography are stated below, each taking a different approach in their own interesting way.

**2.2.2 Definition (Shortest Vector Problem (SVP))**

Given a lattice $\mathcal{L}$ constructed from $n$ basis vectors, find the shortest non-zero lattice vector $\boldsymbol{x} \in \mathcal{L} \backslash \{\mathbf{0}\}$, i.e. find $\boldsymbol{x}$ such that $||\boldsymbol{x}|| = \lambda_{min}$ (Peikert 2016).

Based on SVP, one can construct GapSVP, an approximative version with advantages for usage in practical problems.

Placeholder

Figure 2.2: Illustration of a standard lattice $\mathcal{L}$ over the integers $\mathbb{Z}$ with two basis vectors $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$, cf. Definition 2.2.1. The shortest vector problem in this case is solved by $\boldsymbol{x} = 0\boldsymbol{b}_1 \pm 1\boldsymbol{b}_2$.

**2.2.3 Definition (Decisional Approximate SVP (GapSVP))**

Given a lattice $\mathcal{L}$ and some pre-defined function $\gamma : \mathbb{N} \mapsto \mathbb{R}$ depending on the lattice dimension $n$ (constant for a given $\mathcal{L}$) with $\gamma(n) \geq 1$, the decisional approximate shortest vector problem is distinguishing between $\lambda_{min} \leq 1$ and $\lambda_{min} > \gamma(n)$. For other cases, it is up to the algorithm what to return.

**2.2.4 Definition (Short Integer Solution (SIS) Problem)**

For $m$ given vectors $(\boldsymbol{a}_i)_{0<i\leq m} \in (\mathbb{Z}/q\mathbb{Z})^n$ that comprise the columns of a matrix $A \in (\mathbb{Z}/q\mathbb{Z})^{n\times n}$ and an upper bound $\beta$, find a solution vector $\boldsymbol{z} \in \mathbb{Z}^n \backslash \{\boldsymbol{0}\}$ such that

$$A\boldsymbol{z} = \boldsymbol{0} \quad \text{with} \quad ||\boldsymbol{z}|| \leq \beta \,.$$

Note that without the last requirement $||\boldsymbol{z}|| \leq \beta$, the SIS problem can be easily solved through Gaussian elimination or similar algorithms, however they rarely yield a short (or *the* shortest) solution. It can be shown that solving SIS is at least as hard as solving GapSVP with appropriate parameters (Ajtai 1996).

Using the above problems, multiple cryptographic primitives can be constructed due to the proven hardness that also propagates to quantum computers. Examples include collision resistant hash functions, signatures, pseudorandom functions or even Regev's public-key cryptosystem (Peikert 2016).

### 2.2.1 Learning with Errors (LWE)

Next, we would like to consider LWE, a computing problem that is believed to be sufficiently hard to be used in cryptography and, most notably, is not yet solvable in linear time by a quantum algorithm (cf. section 2.4). Its hardness assumptions are related to GapSVP and were first formally proven by Regev, for which he received the 2018 Gödel price.

**2.2.5 Definition (LWE-Distribution $A_{\boldsymbol{s},\chi_{error}}$)**

Given a prime $p \in \mathbb{N}$ and $n \in \mathbb{N}$, we choose some secret $\boldsymbol{s} \in (\mathbb{Z}/p\mathbb{Z})^n$. In order to sample a value from the LWE distribution $A_{\boldsymbol{s},\chi_{error}}$:

- Draw a random vector $a \in (\mathbb{Z}/p\mathbb{Z})^n$ from the multivariate uniform distribution with its domain in the integers up to $p$.
- Given another probability distribution $\chi_{error}$ over the integers modulo $p$, sample a scalar 'error term' $\mu \in \mathbb{Z}/p\mathbb{Z}$ from it.
- Set $b = \boldsymbol{s} \cdot \boldsymbol{a} + \mu$, with $\cdot$ denoting the standard vector product.
- Output the pair $(\boldsymbol{a}, b) \in (\mathbb{Z}/p\mathbb{Z})^n \times (\mathbb{Z}/p\mathbb{Z})$.

The general approach useful to cryptography is to sample an element from the LWE-distribution and construct two problems out of it, *search*-LWE and *decision*-LWE.

**2.2.6 Definition (LWE-Problem - Search Version)**

Given $m$ independent samples $(\boldsymbol{a}_i, b_i)_{0<i\leq m}$ from $A_{\boldsymbol{s},\chi_{error}}$, find the secret $\boldsymbol{s}$.

**2.2.7 Definition (LWE-Problem - Decision Version)**

Given $m$ samples $(\boldsymbol{a}_i, b_i)_{0<i\leq m}$, distinguish (with non-negligible advantage) whether they were drawn from $A_{\boldsymbol{s},\chi_{error}}$ or from the uniform distribution $u$ over $(\mathbb{Z}/p\mathbb{Z})^n \times (\mathbb{Z}/p\mathbb{Z})$.

In their above definitions, Regev showed that the two problems are equivalent.

**2.2.1 Theorem (Hardness of LWE)**

If there exists an efficient algorithm that solves either search-LWE or decision-LWE then there exists an efficient quantum algorithm that approximates the decision version of the shortest vector problem (GapSVP) in the worst case (Regev 2010).

He also provided a construction of a public-key cryptosystem based on them, i.e. an asymmetric cryptographic system for at least two parties that includes a public and corresponding private key.

Public-key cryptosystems are fundamentally different from symmetric systems, which only require one single key for encryption and decryption at the same time, known by all involved parties. Often times, public-key schemes (rather slow) are used to exchange keys for subsequent symmetric encryption (rather fast) of large plaintexts, for instance in the Transport Layer Security (TLS) protocol (Rescorla 2018).

### 2.2.2 Learning with Errors on Rings (RLWE)

Very similar to Definition 2.2.5, the Ring-LWE distribution is derived as follows (Lyubashevsky, Peikert and Regev 2010):

**2.2.1 Corollary (RLWE-Distribution $B_{\boldsymbol{s},\chi_{error}}$)**

Given a quotient ring $(R/qR, +, \cdot)$, we choose some secret $s \in R/qR$. In order to sample a value from the RLWE distribution $B_{s,\chi_{error}}$:

- Uniformly randomly draw an element $a \in R/qR$
- Given another probability distribution $\chi_{error}$ over the ring elements, sample an 'error term' $\mu \in R/qR$ from it.
- Set $b = s \cdot a + \mu$, with $\cdot$ denoting the ring multiplication operation.
- Output the pair $(a, b) \in R/qR \times R/qR$.

In the exact same manner as above, the search and decision problems can be constructed.

**2.2.2 Corollary (RLWE-Search Problem)**

Given $m$ independent samples $(a_i, b_i)_{0<i\leq m}$ from $B_{s,\chi_{error}}$, find the secret $s$.

**2.2.3 Corollary (RLWE-Decision Problem)**

Given $m$ samples $(a_i, b_i)_{0<i\leq m}$, distinguish (with non-negligible advantage) whether they were drawn from $B_{s,\chi_{error}}$ or from the uniform distribution $u$ over $R/qR \times R/qR$.

The main advantage of RLWE over LWE is that is conceptually similar and yet simple to formalise over an arbitrarily chosen ring $(R, +, \cdot)$ which allows for a vast amount of applications and interesting constructions.

## 2.3 Machine Learning

Undoubtedly one of the most prevalent concepts in todays computing world, Machine Learning (ML) has shaped how computers think and how we interact with them significantly. As Shafi Goldwasser puts it, '*Machine Learning is somewhere in the intersection of Artificial Intelligence, Statistics and Theoretical Computer Science*' (Goldwasser 2018).

Within the scope of this thesis, the basics of neural networks and associated learning methods shall be covered, limited to the category of supervised learning problems (as opposed to unsupervised learning problems). Supervised learning refers to the machine *training* an algorithm to match some input data (features) with corresponding output data (targets), often related to pattern recognition. The trained algorithm can then be utilised to match fresh input data with a prediction of the targets.

A popular subset of applications to ML are classification problems, predominantly image classification, which was not as easily possible before without a human eye due to the lack of computing power. Classification problems can be formulated quickly, the goal is to computationally categorize input data (for instance, images) into a predefined set of classes (for instance, cats and dogs). The primary concept behind Machine Learning is not at all new, linear regression was already employed by Gauss and Legendre in the early $19^{\text{th}}$ century; the term 'Neural Network' was coined by McCulloch and Pitts in 1943. Much media attention was earned in the 2000-2010 decade when larger image classification problems became feasible with the increasing computational power of modern computers, up until the advent of Deep Learning (Bishop and Nasrabadi 2007). Keep in mind however, that Machine Learning in this form is nothing more than glorified statistical regression.

**2.3.1 Definition (Linear Regression)**

Given an input vector $\boldsymbol{x} \in \mathbb{R}^n$, the goal of linear regression is to predict the value of a target $t \in \mathbb{R}$, according to some model $M$.

To illustrate the concept, we will focus on a simple learning method, namely that of gradient descent. In supervised learning problems, this technique first requires us to introduce a loss (error) function $L : \mathbb{R}^n \mapsto \mathbb{R}$, usually Mean-Squared-Error (MSE), which has comparably nice convergence properties due to its parabolic shape:

$$L(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(t_i - \boldsymbol{w}^T\boldsymbol{x}_i) = \frac{1}{2}(\boldsymbol{t} - \Phi\boldsymbol{w})^T(\boldsymbol{t} - \Phi\boldsymbol{w})$$

where $\boldsymbol{w} \in \mathbb{R}^n$ represents the weights and $\Phi \in \mathbb{R}^{N\times(n+1)}$ is an auxiliary matrix introduced for sleeker notation, consisting of .., referred to as the design matrix. When $L(\boldsymbol{w}^*) = 0$, this means we have found the perfect weights, since our predictions exactly match the targets (labels) $t_i$. This is not always possible, so we aim for the minimum error between predictions and targets. In other words, our goal is to find

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}\in\mathbb{R}^n}{\operatorname{argmin}}\, L(\boldsymbol{w})$$

given a dataset $\{\boldsymbol{x}_i, t_i\}$.

### 2.3.1 Gradient Descent

Placeholder

Figure 2.3: Illustration of Gradient Descent, adapted from (User 2020).

TODO: Describe going in the direction of $-\nabla L$ to find the minimum and add more details

To make up for numerical problems and potentially slow convergence, Gradient Descent (GD) can be replaced by more sophisticated methods such as Conjugate Gradient (with convergence guarantees within a certain boundary) or by adding in momentum to the distance travelled in each GD iteration (Bishop and Nasrabadi 2007).

### 2.3.2 Multi-Layered Neural Networks

TODO: Woher kommt diese Struktur?

Placeholder

Figure 2.4: A neural network TODO: Symmetrie zu unserem Netzwerk erklären

To better understand the implications and possibilities of a large neural network, consider the following universal approximation theorem:

**2.3.1 Theorem (Universal Approximation)**

If the neural network has at least one hidden layer, proper nonlinear activation functions and enough data and hidden units, it can approximate any continuous function $y(x, w) : \mathbb{R}^n \mapsto \mathbb{R}$ arbitrarily well on a compact domain (Hornik, Stinchcombe and White 1989).

TODO: Quickly describe common types of layers in NNs?

- Matrix Multiplication (Dense Layer)

- Convolutional Layer
- Sigmoid Activation
- Max Pooling

## 2.4 Post-Quantum Security

Placeholder

Figure 2.5: Illustration of a wave function $\tilde{\psi} : \mathbb{R}^2 \mapsto \mathbb{R}$ as commonly used in quantum mechanics.

In quantum mechanics, we seek a mathematical description of quantum phenomena, commonly building upon SCHRÖDINGER's formalisms based on wave functions and the basic postulates of quantum mechanics.

The mathematical foundation of quantum mechanics is deeply rooted in functional analysis; An important concept is that of function spaces and, especially, Hilbert spaces. Wave functions $\psi : \mathbb{C}^3 \mapsto \mathbb{R}$ are usually chosen as elements of the $\mathcal{L}^2$-space, the space of square-integrable functions:

$$\mathcal{L}^2 = \left\{ \psi : \mathbb{C}^3 \mapsto \mathbb{C} \,\middle|\, ||\psi|| < \infty \right\} \quad \text{with } ||\psi|| = \int_{-\infty}^{\infty} \psi^*(\boldsymbol{x})\psi(\boldsymbol{x})\, dx\,.$$

The most popular interpretation / notion of a wave function is that it relates to the probability that a particle is at the current position $\boldsymbol{r}$ at time $t$ at the given time. Namely, this probability is given by $|\psi(\boldsymbol{r}, t)|^2$.

TODO: Explain quantum entanglement on a high level, Qubits

TODO: Mention Shor's (and Grover's) algorithm and why they are fast, intuitively

### 2.4.1 Definition (NP-Hardness)

A problem is referred to as *NP-hard* iff it is at least as hard as the hardest problems in the complexity class NP (nondeterministic polynomial time). Formally written,

$$\mathrm{NP} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}(n^k)$$

the union of all decision problems with runtime bounded by $\mathcal{O}(n^k)$.

TODO: This is needed in the introduction and other places too, where to put it?

# Chapter 3

# Homomorphic Encryption

## 3.1 Basics of Fully Homomorphic Encryption

HE makes it possible to operate on data without knowing it. One can distinguish three flavors of it, Partial-, Somewhat- and Fully Homomorphic Encryption (FHE).

For FHE, there exist a few schemes in use today with existing implementations.

- BFV scheme for integer arithmetic (Fan and Vercauteren 2012; Brakerski 2012).
- BGV scheme for integer arithmetic (Brakerski, Gentry and Vaikuntanathan 2012).
- CKKS scheme for (complex) floating point arithmetic (Cheon et al. 2017).
- Fastest Homomorphic Encryption in the West (FHEW) scheme for Boolean circuit evaluation (Ducas and Micciancio 2015).
- Torus Fully Homomorphic Encryption (TFHE) scheme for Boolean circuit evaluation (Chillotti et al. 2019).

We will first introduce the BFV scheme (integer arithmetic) as it represents a fundamental building block behind CKKS. Due to the inherent applications, this thesis will focus on the CKKS scheme to perform homomorphic operations on (complex-valued) floating point numbers and vectors.

## 3.2 Homomorphic Encryption using RSA

In order to illustrate the basic idea behind HE, without distancing ourselves too far from the original goal of introducing basic HE operations used in practice, this short section aims to motivate the definition of ring homomorphisms (cf. Definition 3.2.1) behind a cryptographic background.

With unpadded RSA (Rivest, Shamir and Adleman 1983), some arithmetic can be performed on the ciphertext - looking at the encrypted ciphertext $\mathcal{E}(m_1) = (m_1)^r \mod n$ of the message $m_1$ and $m_2$ respectively, the following holds:

$$\begin{aligned}
\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) &\equiv (m_1)^r (m_2)^r \mod n \\
&\equiv (m_1 m_2)^r \mod n \\
&\equiv \mathcal{E}(m_1 \cdot m_2)
\end{aligned}$$

The encryption therefore partially fulfills the properties of a ring homomorphism, which in general terms is defined as follows:

**3.2.1 Definition (Ring Homomorphism)**

Given two rings $(R, +, \cdot)$ and $(S, \oplus, \otimes)$, we call a mapping $\varphi : R \to S$ a ring homomorphism when it satisfies the following conditions:

$$\forall a, b \in R : \varphi(a + b) = \varphi(a) \oplus \varphi(b) \land \varphi(a \cdot b) = \varphi(a) \otimes \varphi(b)$$

As we can see, the term HE stems from the ability to perform computations on encrypted data while ensuring the same results are obtained when the same operations are applied to the original data.

## 3.3 Gentry's FHE-Scheme and BGV

Gentry 2009 TODO: Eine kleine historical introduction hierzu? Bootstrapping erwähnen..?

## 3.4 The BFV scheme

This scheme was developed in two separate publications, whose authors initials it is named after, Brakerski 2012 and Fan and Vercauteren 2012. BFV is based on BGV and they are very similar in their core ideas, one can even convert a BFV ciphertext to an equivalent BGV ciphertext (A. Kim, Polyakov and Zucca 2021). In this section, we will focus on a slightly altered implementation introduced in Lepoint and Naehrig 2014, yet the main aspects are identical to their definitions in the original papers.

TODO: Ausführlichere Beschreibung..?

For two tuples $(\cdot,\cdot)$ defined over the same ring, denote their element-wise addition as $(\cdot,\cdot)+(\cdot,\cdot)$, element-wise multiplication by a scalar $u$ as $u\cdot(\cdot,\cdot)$ and element-wise rounding as $\lfloor(\cdot,\cdot)\rceil$.

### 3.4.1 Definition (The BFV-Scheme)

Let $R=\mathbb{Z}[X]/\Phi_d(X)$ be a polynomial ring with $\Phi_d(X)$ the $d^{\text{th}}$ cyclotomic polynomial ($\rightarrow d\in\mathbb{N}$) for ciphertexts $c\in R\times R$. Introduce $R/qR$ the associated quotient ring of the $q^{\text{th}}$ coset of $R$ with the modulus $q\in\mathbb{N}$. Further let $t\in\mathbb{N}$ denote the message modulus with $1<t<q$ for plain messages $m\in R/tR$ and define $\delta=\lfloor\frac{q}{t}\rfloor$, $\delta^{-1}=\frac{t}{q}$.
Introduce three bounded discrete probability distributions $\chi_{key}$, $\chi_{enc}$ and $\chi_{error}$ over $R/qR$, one which is only used once for key generation, another used for BFV.Encrypt and another (usually Gaussian-like) error distribution for manually inserted error terms (confer the LWE-problem). For BFV, usually $\chi_{key}=\chi_{enc}$.
For a polynomial $a\in R/qR$, consider the decomposition $a=\sum_{i=0}^{l-1}a_iw^i$ into base $w\in\mathbb{N}$ obtained by $\text{WordDecomp}: R\mapsto R^l$, $\text{WordDecomp}(a)=([a_i]_w)_{i=0}^{l-1}$.
Further let $\text{PowersOf}: R\mapsto R^l$ be defined as $\text{PowersOf}(a)=([aw^i]_q)_{i=0}^{l-1}$.
Let the parameters $\mathbb{P}=(d,q,t,\chi_{key},\chi_{error},w)$ and $l=\lfloor\log_w(q)\rfloor+1$.

BFV.

| | |
|---|---|
| ParamGen($\lambda$) | Choose parameters as defined above, given the security parameter $\lambda$, such that $1<t<q$, $w\geq 2$, initialize distributions $\chi_{key}$, $\chi_{enc}$ and $\chi_{error}$ $\rightarrow\mathbb{P}$ |
| KeyGen($\mathbb{P}$) | Generate the secret key $s\leftarrow\chi_{key}$, sample $\boldsymbol{\mu}\in(R/qR)^l$ from $\chi_{error}$ and choose some $\boldsymbol{a}\in(R/qR)^l$ uniformly at random, compute the relinearization key $\boldsymbol{\gamma}=(\text{PowersOf}(s^2)-(\boldsymbol{\mu}+\boldsymbol{a}\cdot s),\boldsymbol{a})$ and finally output the public key for uniformly random $a\in(R/qR)$ and $\mu\leftarrow\chi_{error}$ with $b=-(a\cdot s+\mu)$ as $\boldsymbol{p}=(b,a)$. $\rightarrow\boldsymbol{p},s,\boldsymbol{\gamma}$ |
| Encrypt($\boldsymbol{p},m$) | Let $(b,a)=\boldsymbol{p}$, $u\leftarrow\chi_{enc}$, $\mu_1,\mu_2\leftarrow\chi_{error}$, then the ciphertext is $\boldsymbol{c}=u\cdot\boldsymbol{p}+(\delta m+\mu_1,\mu_2)=(\delta m+bu+\mu_1,au+\mu_2)$ $\rightarrow\boldsymbol{c}$ |
| Decrypt($s,\boldsymbol{c}$) | Decrypt $\boldsymbol{c}=(c_0,c_1)$ as $m=\lfloor\delta^{-1}[c_0+c_1s]_q\rceil\in R/tR$ $\rightarrow m$ |
| Add($\boldsymbol{c}_1,\boldsymbol{c}_2$) | Let $(c_0^1,c_1^1)=\boldsymbol{c}_1$ and $(c_0^2,c_1^2)=\boldsymbol{c}_2$ then $\boldsymbol{c}_3=(c_0^1+c_0^2,c_1^1+c_1^2)=\boldsymbol{c}_1+\boldsymbol{c}_2$ $\rightarrow\boldsymbol{c}_3$ |
| Mult($\boldsymbol{c}_1,\boldsymbol{c}_2$) | Output $\overline{\boldsymbol{c}}=(\lfloor\delta^{-1}c_0^1c_0^2\rceil,\lfloor\delta^{-1}(c_0^1c_1^2+c_1^1c_0^2)\rceil,\lfloor\delta^{-1}c_1^1c_1^2\rceil)$ $\rightarrow\overline{\boldsymbol{c}}$ |
| ReLin($\overline{\boldsymbol{c}},\boldsymbol{\gamma}$) | Using the relin key $\boldsymbol{\gamma}=(\boldsymbol{b},\boldsymbol{a})$, relinearize from $\overline{\boldsymbol{c}}=(c_0,c_1,c_2)$ as $\boldsymbol{c}=(c_0+\text{WordDecomp}(c_2)\cdot\boldsymbol{b},c_1+\text{WordDecomp}(c_2)\cdot\boldsymbol{a})$ $\rightarrow\boldsymbol{c}$ |

(Fan and Vercauteren 2012; Brakerski 2012)

To summarise the parameters and variables, a brief overview of all used symbols is provided in Table 3.1.

Table 3.1: Summary of the parameters and symbols in BFV.

| Symbol | Space | Explanation |
|---|---|---|
| $\lambda$ | $\in \mathbb{R}$ | Security parameter |
| $d$ | $\in \mathbb{N}$ | Index of the cyclotomic polynomial used in $R$ |
| $q$ | $\in \mathbb{N}$ | Modulus of the ciphertext space $R/qR$ |
| $t$ | $\in \mathbb{N}$ | Modulus of the plaintext message space $R/tR$ |
| $\delta$ | $\in \mathbb{N}$ | Ratio between ciphertext and plaintext modulus |
| $\delta^{-1}$ | $\in \mathbb{R}$ | Inversion coefficient of the effect of $\delta$ |
| $w$ | $\in \mathbb{N}$ | Word size used as basis, e.g. $w = 2$ for bits |
| $l$ | $\in \mathbb{N}$ | Number of words of size $w$ required to encode $q$ |
| $s$ | $\in R$ | Secret Key |
| $\boldsymbol{p}$ | $\in R/qR \times R/qR$ | Public Key $(b, a)$ |
| $\boldsymbol{\gamma}$ | $\in (R/qR)^l \times (R/qR)^l$ | Relinearization Key |
| $m$ | $\in R/tR$ | Plaintext Message |
| $\boldsymbol{c}$ | $\in R \times R$ | Ciphertext |
| $\overline{\boldsymbol{c}}$ | $\in R \times R \times R$ | Slightly larger ciphertext resulting from multiplication |

TODO: Etwas mehr Kontext...?

Placeholder TODO: Schematik an Notation anpassen

**3.4.1 Theorem (BFV encryption is homomorphic with respect to addition)**

BFV.Encrypt should encrypt in such a way that the addition algebra can be retained even in the transformed space, showing that we can indeed refer to it as homomorphic encryption.

TODO: Obiges Theorem kurz zeigen (ausrechnen)

*Microsoft SEAL* implements the scheme, enabled using `seal::scheme_type::bfv`.

## 3.5 The CKKS scheme

The CKKS scheme allows us to perform approximate arithmetic on floating point numbers. Essentially, the idea is to extend BFV which allows us to operate on vectors $\boldsymbol{y} \in \mathbb{Z}_t^n$, by an embedding approach that allows us to encode a (complex) floating point number vector $\boldsymbol{x} \in \mathbb{R}^n(\mathbb{C}^n)$ as an integer vector. A naïve approach would be to use a fixed-point embedding:

$$\mathrm{Embed}(\boldsymbol{x}) = \boldsymbol{x} \cdot F$$

with $F \in \mathbb{Z}$. In decimal form, for instance with $F = 1000$, we could effectively encode three decimal places of the original vector $\boldsymbol{x}$.

TODO: Obiges soll Motivation hinter CKKS zeigen.. fertig schreiben

Introduce $d, R, R/qR$ as in Definition 3.4.1 and further define $S = \mathbb{R}[X]/\Phi_d(X)$ a similar polynomial ring to $R$, but over the reals instead of the integers. Let $N = \varphi(d)$ be the degree of the reducing cyclotomic polynomial of $S$, confer Definition 2.1.6. For convenience, we usually choose $d$ a power of 2 and then, by Theorem 2.1.1, $N = \varphi(d) = \frac{d}{2}$ which yields very efficiently multipliable polynomials because the homomorphic multiplication operation can be performed using a Discrete Fourier Transform (DFT) and further optimized using the Fast Fourier Transform (FFT), which in its unmodified form only accepts power-of-2 vector sizes (Cheon et al. 2017).

**3.5.1 Definition (Canonical Embedding $\underline{\sigma}$)**

For a real-valued polynomial $p \in S$, define the canonical embedding of $S$ in $\mathbb{C}^N$ as a mapping $\underline{\sigma} : S \mapsto \mathbb{C}^N$ with

$$\underline{\sigma}(p) := \left(p(e^{-2\pi i j/N})\right)_{j \in \mathbb{Z}_d^*}$$

with $\mathbb{Z}_d^* := \{x \in \mathbb{Z}/d\mathbb{Z} \,|\, \gcd(x, d) = 1\}$ the set of all integers smaller than $d$ that do not share a factor $> 1$ with $d$. The image of $\underline{\sigma}$ given a set of inputs $R$ shall be denoted as $\underline{\sigma}(R) \subseteq \mathbb{C}^N$. Let the inverse of $\underline{\sigma}$ be denoted by $\underline{\sigma}^{-1} : \mathbb{C}^N \mapsto S$.

Note that evaluating a polynomial on the $n^{\text{th}}$ roots of unity corresponds to performing a Fourier-Transform.

Define the commutative subring $(H, +, \cdot)$ of $(\mathbb{C}^N, +, \cdot)$ on the set

$$H = \{\boldsymbol{z} = (z_j)_{j \in \mathbb{Z}_d^*} \in \mathbb{C}^N : z_j = \overline{z_{-j}} \ \forall j \in \mathbb{Z}_d^*\} \subseteq \mathbb{C}^N$$

of all complex-valued vectors $\boldsymbol{z}$ where the first half equals the reversed complex-conjugated second half.

**3.5.2 Definition (Natural Projection $\underline{\pi}$)**

Let $T$ be a mulitplicative subgroup of $\mathbb{Z}_d^*$ with $\mathbb{Z}_d^*/T = \{\pm 1\} = \{1T, -1T\}$, then the natural projection $\underline{\pi} : H \mapsto \mathbb{C}^{N/2}$ is defined as

$$\underline{\pi}\left((z_j)_{j \in \mathbb{Z}_M^*}\right) = (z_j)_{j \in T}$$

Let its inverse be denoted by $\underline{\pi}^{-1} : \mathbb{C}^{N/2} \mapsto H$ and consequently defined as

$$\underline{\pi}^{-1}\Big((z_j)_{j\in T}\Big) = \Big(\nu(z_j)\Big)_{j\in\mathbb{Z}_M^*} \text{ with } \nu(z_j) = \begin{cases} z_j & \text{if } j \in T \\ \overline{z_j} & \text{otherwise} \end{cases}$$

The natural projection $\underline{\pi}$ simply halves a vector $\boldsymbol{z} \in H$ to all elements where $j \in T$ to only contain its essential information (the first half), since the second half can easily be reconstructed by element-wise conjugation using $\nu$. The exact structure of $T$ is given by $\mathbb{Z}_d^*/T = \{\pm 1T\}$ with $+1T$ and $-1T$ denoting multiplicative left cosets of $T$, together forming the quotient group $(\mathbb{Z}_d^*/T, \cdot)$ over multiplication (denoted $\cdot$ instead of $+$ as in the quotient group definition in the previous chapter).

**Further studying $T$**, we first notice that by LAGRANGE's theorem on finite groups, the number of elements in $T$ is exactly $N/2$ since

$$\frac{|\mathbb{Z}_d^*|}{|T|} = |\{\pm 1\}| \Leftrightarrow \frac{N}{|T|} = 2 \Leftrightarrow |T| = \frac{N}{2}$$

leading to $\underline{\pi}(H) \subseteq \mathbb{C}^{N/2}$. Rephrased, we seek a $T \subseteq \mathbb{Z}_d^*$ with $1 \in T$ such that we can fully construct $\mathbb{Z}_d^*$ by the union of the cosets $1T$ and $-1T$, i.e. $\mathbb{Z}_d^* = (1T) \cup (-1T)$. Note that $T$ is not unique, we can find multiple sets $T$ for which the above holds, for instance by brute force computation:

```python
import itertools, math, numpy as np
d = 16; Zdstar = [z for z in range(d) if math.gcd(d, z) == 1]
possible_T = [T for T in itertools.combinations(Zdstar, len(Zdstar) // 2)
  if 1 in T and list(np.unique(list(T) + [(-1*t) % d for t in T])) == Zdstar]
```

**Example.** Let $d = 16$, then $\mathbb{Z}_d^* = \{1, 3, 5, 7, 9, 11, 13, 15\}$ and $N = |\mathbb{Z}_d^*| = 8$ and by LAGRANGE's theorem, $|T| = 4$. Since $(T, \cdot)$ forms a normal subgroup under multiplication, we must have that $1 \in T$ and we can identify all possible subgroups $T$ satisfying $\mathbb{Z}_d^*/T = \{\pm 1T\}$ to be one of

$$\{1,3,5,7\}, \{1,3,5,9\}, \{1,3,7,11\}, \{1,3,9,11\},$$
$$\{1,5,7,13\}, \{1,5,9,13\}, \{1,7,11,13\}, \{1,9,11,13\}$$

using the above Python code. An example of an invalid subset $T$ that does cover the whole original set $\mathbb{Z}_d^*$ would be $T = \{1, 7, 9, 15\}$.

For the purposes of CKKS, we simply choose a global $T$ as above that is constant for our encoding and decoding procedure and a given $d$. The inverse natural projection $\underline{\pi}^{-1}$ then uniquely constructs a vector in $H$ by filling in elements $\overline{z_j}$ for $j \notin T$ into $\boldsymbol{z}$. For simplicity, we commonly choose $T$ as the 'first half' of $\mathbb{Z}_d^*$ when sorting in an ascending manner as it is always a valid choice. [1]

[1] This can be seen from the coset $-1T$ which exactly equals the 'missing' half in $\mathbb{Z}_d^*$ when the first half is covered by $1T = T = \{1, 3, 5, ..., N-1\}$ since $-1T = \{-1, -3, -5, ..., -(N-1)\} \equiv \{d-1, d-3, d-5, ..., d-N+1\} \pmod d$ when $d$ a power of 2. Then, $(1T) \cup (-1T) = \{1, 3, 5, ..., N-1\} \cup \{d-1, d-3, d-5, ..., d-N+1\} = \{1, 3, 5, ..., N-1, N+1, ..., d-5, d-3, d-1\} = \mathbb{Z}_d^*$.

**3.5.3 Definition (Discretisation to an element of $\underline{\sigma}(R)$)**

Using one of several round-off algorithms (cf. Lyubashevsky, Peikert and Regev 2013), given an element of $H$, define a rounding operation $\underline{\rho} : H \mapsto \underline{\sigma}(R)$ that maps an $\boldsymbol{h} \in H$ to its closest element in $\underline{\sigma}(R)$, also denoted as

$$\underline{\rho}(\boldsymbol{h}) = \lfloor \boldsymbol{h} \rceil_{\underline{\sigma}(R)} .$$

Further let $\underline{\rho_\delta}(\boldsymbol{h}) = \lfloor \delta \cdot \boldsymbol{h} \rceil_{\underline{\sigma}(R)}$ denote the same rounding operation but with prior scaling by a scalar factor $\delta$. Note that $\underline{\rho}^{-1}$ is given directly as the identity operation because all elements of its domain are already elements of its image. Similarly, $\underline{\rho_\delta}^{-1}(\boldsymbol{y}) = \delta^{-1} \cdot \boldsymbol{y}$.

Because it is not essential to understanding the encryption scheme, we will skip over concrete implementations of the rounding procedure $\underline{\rho}$. Note that for choosing a 'close' element $\boldsymbol{g} \in H$, we must first introduce a sense of proximity, in this case done by the $l_\infty$-norm $||\boldsymbol{g} - \boldsymbol{h}||_\infty$ of the difference between $\boldsymbol{h} \in H$ and $\boldsymbol{g}$.

### 3.5.4 Definition (The CKKS Scheme)

Define $R, R/q_L R$ as in Definition 3.4.1.

CKKS.

| | |
|---|---|
| ParamGen($\lambda$) | Choose parameters as defined above, given the security parameter $\lambda$ and space modulus $q_L$, choose $d \in \mathbb{N}$ a power of 2, $P, h \in \mathbb{Z}$, $\sigma \in \mathbb{R}$ and initialize distributions $\chi_{key}$, $\chi_{enc}$ and $\chi_{error}$. $\rightarrow \mathbb{P}$ |
| KeyGen($\mathbb{P}$) | Sample the secret key $s \leftarrow \chi_{key}$, $a \in R_{q_L}$ uniformly at random, $\mu \leftarrow \chi_{error}$ and obtain the public key $\boldsymbol{p} = (b, a)$ with $b = -a \cdot s + \mu$. Sample $a' \in R_{P \cdot q_L}$ uniformly at random, $\mu' \leftarrow \chi_{error}$ and obtain the evaluation key $\boldsymbol{\gamma} = (b', a')$ with $b' = -a' \cdot s + \mu' + Ps^2$. $\rightarrow \boldsymbol{p}, s, \boldsymbol{\gamma}$ |
| Encode($\boldsymbol{z}$) | For a given input vector $\boldsymbol{z}$, output $m = (\underline{\sigma}^{-1} \circ \underline{\rho_\delta} \circ \underline{\pi}^{-1})(\boldsymbol{z}) = \underline{\sigma}^{-1}(\lfloor \delta \cdot \underline{\pi}^{-1}(\boldsymbol{z}) \rceil_{\underline{\sigma}(R)})$ $\rightarrow m$ |
| Decode($m$) | Decode plaintext $m$ as $\boldsymbol{z} = (\underline{\pi} \circ \underline{\rho_\delta}^{-1} \circ \underline{\sigma})(m) = (\underline{\pi} \circ \underline{\sigma})(\delta^{-1} m)$ $\rightarrow \boldsymbol{z}$ |
| Encrypt($\boldsymbol{p}, m$) | Let $(b, a) = \boldsymbol{p}$, $u \leftarrow \chi_{enc}$, $\mu_1, \mu_2 \leftarrow \chi_{error}$, then the ciphertext is $\boldsymbol{c} = u \cdot \boldsymbol{p} + (m + \mu_1, \mu_2) = (m + bu + \mu_1, au + \mu_2)$ $\rightarrow \boldsymbol{c}$ |
| Decrypt($s, \boldsymbol{c}$) | Decrypt the ciphertext $\boldsymbol{c} = (c_0, c_1)$ as $m = [c_0 + c_1 s]_{q_L}$ $\rightarrow m$ |
| Add($\boldsymbol{c}_1, \boldsymbol{c}_2$) | Output $\boldsymbol{c}_3 = \boldsymbol{c}_1 + \boldsymbol{c}_2$ $\rightarrow \boldsymbol{c}_3$ |
| Mult($\boldsymbol{c}_1, \boldsymbol{c}_2$) | Output $\overline{\boldsymbol{c}} = (c_0^1 c_0^2, c_0^1 c_1^2 + c_1^1 c_0^2, c_1^1 c_1^2)$ $\rightarrow \overline{\boldsymbol{c}}$ |
| ReLin($\overline{\boldsymbol{c}}, \boldsymbol{\gamma}$) | Using the evaluation key $\boldsymbol{\gamma}$, relinearize from $\overline{\boldsymbol{c}} = (c_0, c_1, c_2)$ to $\boldsymbol{c} = (c_0, c_1) + \lfloor P^{-1} c_2 \boldsymbol{\gamma} \rceil$ $\rightarrow \boldsymbol{c}$ |
| ReScale($\boldsymbol{c}$) | In order to rescale a ciphertext from level $l_{old}$ to $l_{new}$, multiply by a factor $\frac{q_{l_{new}}}{q_{l_{old}}} \in \mathbb{Q}$ and round to the nearest element of $(R/q_{l_{new}} R) \times (R/q_{l_{new}} R)$: $\boldsymbol{c}_{new} = \left\lfloor \frac{q_{l_{new}}}{q_{l_{old}}} \boldsymbol{c} \right\rceil$ $\rightarrow \boldsymbol{c}_{\boldsymbol{new}}$ |

(Cheon et al. 2017)

For more details on the probability distributions, refer to the original CKKS paper (Cheon et al. 2017), with the following naming relations: $\chi_{key} = \mathcal{HWT}(h)$ over $\{0, \pm 1\}^N$, $\chi_{error} = \mathcal{DG}(\sigma^2)$ over $\mathbb{Z}^N$ and $\chi_{enc} = \mathcal{ZO}(0.5)$ another distribution over $\{0, \pm 1\}^N$.

It should also be noted that the encoding procedure represents an isometric ring isomorphism between its domain and image, as does the decoding procedure. This reflects in the observation that the plaintext sizes and errors are preserved under the transformations (Cheon et al. 2017).

To summarise the parameters and variables, a brief overview of all used symbols is provided in Table 3.2.

Table 3.2: Summary of the parameters and symbols in CKKS.

| Symbol | Space | Explanation |
|---|---|---|
| $\lambda$ | $\in \mathbb{R}$ | Security parameter |
| $d$ | $\in \mathbb{N}$ | Index of the cyclotomic polynomial used in $R$ |
| $P$ | $\in \mathbb{Z}$ | TODO: Hmmm... wie sieht das genau aus? |
| $h$ | $\in \mathbb{Z}$ | Hamming weight of the secret key (used by $\chi_{key}$) |
| $\sigma$ | $\in \mathbb{R}$ | Standard deviation of the Gaussian $\chi_{error}$ |
| $q_L$ | $\in \mathbb{N}$ | Modulus of $R/q_L R$ at level $L$ |
| $\delta$ | $\in \mathbb{N}$ | Scaling factor used when encoding |
| $\delta^{-1}$ | $\in \mathbb{R}$ | Inversion coefficient of the effect of $\delta$ |
| $s$ | $\in \{0, \pm 1\}^N$ | Secret Key |
| $\boldsymbol{p}$ | $\in R/q_L R \times R/q_L R$ | Public Key $(b, a)$ |
| $\boldsymbol{\gamma}$ | $\in R/(P \cdot q_L)R \times R/(P \cdot q_L)R$ | Relinearization Key |
| $m$ | $\in R$ | Plaintext Message |
| $\boldsymbol{c}$ | $\in R/q_L R \times R/q_L R$ | Ciphertext Message |
| $\overline{\boldsymbol{c}}$ | $\in R/q_L R \times R/q_L R \times R/q_L R$ | Slightly larger ciphertext from multiplication |

TODO: Etwas mehr Kontext...?

Placeholder TODO: Schematik an CKKS anpassen

**3.5.1 Theorem (CKKS encryption is homomorphic with respect to addition)**

CKKS.Encrypt should encrypt in such a way that the addition algebra can be retained even in the transformed space, showing that we can indeed refer to it as homomorphic encryption.

TODO: Obiges Theorem kurz zeigen (ausrechnen)

*Microsoft SEAL* implements the scheme, enabled using `seal::scheme_type::ckks`.

# Chapter 4

# Implementation

## 4.1 Chosen Software Architecture

In the given setting, the most accessible frontend is commonly a JavaScript web application.

To still make the classification run as quickly and efficiently as possible, a C++ binary runs in the backend providing an HTTP API to the frontend application. In order to allow for more flexibility of the HTTP server, the initial approach was to pipe requests through a dedicated web application framework with database access that would allow, for instance, user management next to the basic classification. However, the resulting communication and computation overhead, even when running with very efficient protocols such as ZeroMQ, was too high.

Extending the accessibility argument to reproducibility, Docker is a very solid choice (Nüst et al. 2020). To run the attached demo project, simply execute

```
docker-compose build
docker-compose up
```

in the 'code' folder and point your browser to `https://localhost`.

TODO: Hier noch mehr ins Detail gehen?

### 4.1.1 Docker Multi-Stage Build

An enterprise-grade, scalable deployment is achieved by means of zero-dependency Alpine Linux images which contain nothing but compiled binaries and linked libraries.

## 4.2 The MNIST dataset

The MNIST dataset (LeCun and Cortes 1998) contains X train and Y test images with corresponding labels. In order to stick to the traditional feedforward technique with data represented in vector format, therefore it is common to reshape data from $(28, 28)$ images (represented as grayscale values in a matrix) into a 784 element vector.

TODO: Ein paar Samples einfügen, oder auf ciphertext visual referenzieren

## 4.3 Matrix-Vector Multiplication

The dot product that is required as part of the neural network evaluation process needs to be implemented on SEAL ciphertexts as well.

There are multiple methods to achieve a syntactically correct dot product (matrix-vector multiplication) as described by Juvekar, Vaikuntanathan and Chandrakasan (2018) for (square) matrices.

1. **Naïve MatMul** - very simple to derive but impractical in practice due to the limited further applicability of the result consisting of multiple ciphertexts. Applicable to arbitrary matrix dimensions, i.e. matrices $M \in \mathbb{R}^{s \times t}$, of course limited by the unreasonably high memory consumption and computation time of this approach.
2. **Diagonal MatMul** - a simple and practical solution applicable to square matrices $M \in \mathbb{R}^{t \times t}$ that has a major advantage compared to the previous method as the computation yields a single ciphertext object instead of many which can be directly passed on to a following evaluation operation.
3. **Hybrid MatMul** - essentially extending the diagonal method by generalising the definition of the diagonal extraction mechanism to 'wrap around' in order to match the dimensionality of the input vector. Applicable to arbitrary matrix dimensions, i.e. matrices $M \in \mathbb{R}^{s \times t}$ and favourable compared to the Naïve Method.
4. **Babystep-Giantstep MatMul** - a more sophisticated technique aiming to significantly reduce the number of Galois rotations as they are rather expensive to carry out, with a performance boost especially noticeable for higher matrix dimensions. Without further modification, applicable to square matrices.

For the following, define

$$\mathrm{rot}_j : \mathbb{R}^t \mapsto \mathbb{R}^t,\ \{\mathrm{rot}_j(\boldsymbol{x})\}_i = x_{i+j} \tag{4.1}$$

$$\mathrm{diag}_j : \mathbb{R}^{t \times t} \mapsto \mathbb{R}^t,\ \{\mathrm{diag}_j(M)\}_i = M_{i,(i+j)} \tag{4.2}$$

with all indices $i, j \in \mathbb{Z}_t$ member of the cyclic quotient group $\mathbb{Z}_t := \mathbb{Z}/t\mathbb{Z}$ of all integers modulo $t$, meaning that overflowing indices simply wrap around again starting at index 0 to simplify notation. For the sake of compactness, we stick to this notation for the rest of this section.

### 4.3.1 The Naïve Method

Placeholder

Figure 4.1: The naïve method to multiply a square matrix with a vector.

Term by term, one can express a matrix-vector product of $M \in \mathbb{R}^{s \times t}$ and $\boldsymbol{x} \in \mathbb{R}^s$ as follows:

$$\{M\boldsymbol{x}\}_i = \sum_{j=1}^{t} M_{ij} x_j \,.$$

Accordingly, a natural (or rather, naïve) way to model this multiplication in *Microsoft SEAL* would be to

1. encode each $i$-th matrix row $(M_{i,1}, M_{i,2}, ..., M_{i,t})$ using the `Encoder` with matching parameters to the ciphertext of the encoded vector $\boldsymbol{x}$.
2. multiply each encoded row with the encrypted vector using `Evaluator.multiply_plain()` to obtain the ciphertext vector $\boldsymbol{y_i} \in \mathbb{R}^s$ for row $i$.
3. perform the 'rotate-and-sum' algorithm (Juvekar, Vaikuntanathan and Chandrakasan 2018) on each resulting vector (ciphertext) $\boldsymbol{y_i}$ to obtain the actual dot product of the matrix row with the vector $\boldsymbol{x}$:
   (a) using Galois automorphisms, rotate the entries of $\boldsymbol{y_i}$ by $\frac{s}{2}$ elements to obtain $\mathrm{rot}_{\frac{s}{2}}(\boldsymbol{y_i})$.
   (b) perform an element-wise sum $\boldsymbol{y_i} + \mathrm{rot}_{\frac{s}{2}}(\boldsymbol{y_i})$ whose first (and also second) half now contains the sum of the two halves of $\boldsymbol{y_i}$.
   (c) repeat the previous two steps $\log_2(s)$ times, halving the split parameter $s$ each time until one obtains 1 element, which yields us the requested sum of all entries $\sum_{k=1}^{s}\{\boldsymbol{y_i}\}_k$ as the dot product of $\boldsymbol{x}$ and $\boldsymbol{y_i}$.
4. Given all the 'scalar' results of each row-vector dot product, we can construct the resulting matrix-vector product.

**Adapting to non-square matrices** The weight matrices in the given classification setting are by no means square, on the contrary their output dimension tends to be much lower than the input dimension as the goal is to reduce it from $28^2 = 784$ to 10 overall.

However, that also means one cannot directly apply the diagonal method as described in the proceedings above. This 'flaw' can be mitigated by a simple zero-padding approach in order to make the matrix square, filling in zeroes until the lower dimension reaches the higher one.

### 4.3.2 The Diagonal Method

Placeholder

Figure 4.2: The diagonal method to multiply a square matrix with a vector.

TODO: Kontext einfügen

**4.3.1 Theorem (Diagonal Method)**

Given a matrix $M \in \mathbb{R}^{t \times t}$ and a vector $\boldsymbol{x} \in \mathbb{R}^t$, the dot product between the two can be expressed as

$$M\boldsymbol{x} = \sum_{i=0}^{t} \mathrm{diag}_i(M)\mathrm{rot}_i(\boldsymbol{x})$$

Placeholder

Figure 4.3: Diagonal Method error development after each rotation of the input vector

TODO: Interpretation des Obigen

### 4.3.3 The Hybrid Method

Placeholder

Figure 4.4: The hybrid method to multiply an arbitrarily sized matrix with a vector.

To further extend the previous matrix multiplication method to solve the problem (cf. section 4.3.1), it is first necessary to extend the definition of the diag operator to non-square matrices $M \in \mathbb{R}^{s \times t}$. For the following, extending the above definition:

$$\mathrm{diag}_j : \mathbb{R}^{s \times t} \mapsto \mathbb{R}^t,\ \{\mathrm{diag}_j(M)\}_i = M_{i,(i+j)}$$

To exemplarily describe an implementation of an HE algorithm, we break down the following matrix multiplication using the method described above.

```
void DenseLayer::matmulHybrid(seal::Ciphertext &in_out, const Matrix &mat, seal::GaloisKeys &galois_keys,
    seal::CKKSEncoder &encoder, seal::Evaluator &evaluator) {
  size_t in_dim = mat.shape(0);
  size_t out_dim = mat.shape(1);

  // diagonal method preparation
  std::vector<seal::Plaintext> diagonals = encodeMatrixDiagonals(mat, encoder);

  // perform the actual multiplication
  seal::Ciphertext original_input = in_out; // makes a copy
  seal::Ciphertext sum = in_out; // makes another copy
  evaluator.multiply_plain_inplace(sum, diagonals[0]);
  for (auto offset = 1ULL; offset < in_dim; offset++) {
    seal::Ciphertext tmp;
    evaluator.rotate_vector(original_input, offset, galois_keys, in_out);
    evaluator.multiply_plain(in_out, diagonals[offset], tmp);
    evaluator.add_inplace(sum, tmp);
  }
  in_out = sum;
  evaluator.rescale_to_next_inplace(in_out); // scale down once
}
```

TODO: Describe the main commands used above (sehr illustrativ denke ich..?)

### 4.3.4 The Babystep-Giantstep Optimization

Since Galois rotations are the most computationally intensive operations in most cryptographic schemes used today (Dobraunig et al. 2021), they take a large toll on the efficiency. In order to reduce the number of rotations required, one can make use of the *Babystep-Giantstep* optimization as described in Halevi and Shoup 2018, which works as follows:

**4.3.2 Theorem (Babystep-Giantstep Optimization)**

Given a matrix $M \in \mathbb{R}^{t \times t}$ and a vector $\boldsymbol{x} \in \mathbb{R}^t$, with $t = t_1 \cdot t_2$ split into two BSGS parameters $t_1, t_2 \in \mathbb{N}$ and

$$\mathrm{diag}'_p(M) = \mathrm{rot}_{-\lfloor p/t_1 \rfloor \cdot t_1}(\mathrm{diag}_p(M))\,,$$

one can express a matrix-vector multiplication as follows:

$$M\boldsymbol{x} = \sum_{k=0}^{t_2-1} \mathrm{rot}_{(kt_1)}\left( \sum_{j=0}^{t_1-1} \mathrm{diag}'_{(kt_1+j)}(M) \cdot \mathrm{rot}_j(\boldsymbol{x}) \right)$$

where $\cdot$ denotes an element-wise multiplication of two vectors.

A proof of the above theorem can be found in the Appendix.

TODO: Proof lieber hier oder im Appendix?

Note that the optimized matrix-vector multiplication only requires $t_1 + t_2$ as we can store the $t_1$ inner rotations of the vector $\boldsymbol{x}$ for the upcoming evaluations. For larger matrices and vectors (larger $t$), $t_1 + t_2$ are indeed much smaller than the conventional number of required rotations $t = t_1 \cdot t_2$ in the diagonal or hybrid method for instance, which was the point of this modification in the first place.

## 4.4 Polynomial Evaluation

From the implementation perspective, there are three properties to watch out for when working with SEAL ciphertexts:

1. Scale (retrieved using `x.scale()`)

   Can be adjusted with: `evaluator.rescale_inplace()`

2. Encryption Parameters (retrieved using `x.parms_id()`)
   Can be adjusted with: `evaluator.mod_switch_to_inplace()`

3. Ciphertext Size (retrieved using `x.size()`)
   Can be adjusted with: `evaluator.relinearize_inplace()`

TODO: Explain the challenges of polyval

**Multiplication** Each time one multiplies two ciphertexts, the scales multiply (logarithmically, they add up, i.e. the bits are added together). The chain index reduces by 1. The chain index of an encoded ciphertext depends on the coeff moduli. There must be enough bits remaining to perform the multiplication, namely $\log 2(\mathit{scale})$ bits.

**Addition** The scales must be the same, but luckily they will not change.

## 4.5 Neural Network

The neural network was trained using the unencrypted standard Modified National Institute of Standards and Technology database (MNIST) dataset of 50,000 images, split into 90 % training and 10 % validation data.

Placeholder

Figure 4.5: Comparison of the Relu activation function vs. its Taylor expansion

TODO: Describe Taylor approximation of Relu function a bit

To gain some intuition on what the two layers look like internally, the following plots of weights and biases have been made:

Placeholder

Figure 4.6: First Layer Weights and Biases

Placeholder

Figure 4.7: Second Layer Weights and Biases

TODO: Beschreibung der obigen Figures, gehen beide Plots auf eine Seite?

# Chapter 5

# Results

In order to visually demonstrate the encryption, visualisations of the ciphertext polynomial $c_0$ (refer to section 3.5) were generated using a Chinese Remainder Theorem (CRT) decomposition of the Residue Number System (RNS) representation of $c_0$. Each pixel corresponds to a coefficient $a \in \mathbb{Z}/q\mathbb{Z}$ scaled down by the modulus $q$ to obtain a brightness value between 0 and 1.

Placeholder

Figure 5.1: Ciphertext Visualisation TODO: Eigene Subsection hierfür?

Placeholder

Figure 5.2: Development of the classification accuracy and the mean squared error during training.

The machine learning framework behind the project, Tensorflow, splits its training process into *epochs*, which can be found on the x-axis in the plot above. For each training epoch, we find the progress that has been made in a single epoch by looking at the new accuracy (which percentage of the images has been classified correctly) and the loss function (MSE in this case). Per training run, we make a differentiation between training metrics and validation metrics, illustratively shown above for the given network. Validation data is not involved in the training process, it is used to find a point in time when training accuracy still rises while validation accuracy starts to drop. At this point we are very likely to find the network's learning process in an *overfitting* situation, so the training process terminates.

Placeholder

Figure 5.3: Confusion Matrix of the trained network. TODO: Ein bisschen beschreiben..

## 5.1 Accuracy, Precision, Recall

The network classifies 97.62 % of the 10,000 test images correctly.

For a binary classification, two further metrics of interest are

$$\text{Precision} = \frac{\text{tp}}{\text{tp} + \text{fp}} \qquad \text{Recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

with tp ... True Positives, fp ... False Positives, fn ... False Negatives.

Precision (also referred to as PPV, positive predictive value) refers to the ability of the network to classify positive samples correctly, while Recall explains the completeness of the classified samples (i.e. how few true positives have been left out).

Table 5.1: Precision and Recall of the trained network for each digit individually

| **Digit** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Precision** | 0.978 | 0.990 | 0.959 | 0.960 | 0.985 | 0.968 | 0.977 | 0.976 | 0.963 | 0.978 |
| **Recall** | 0.986 | 0.989 | 0.975 | 0.977 | 0.975 | 0.964 | 0.980 | 0.964 | 0.967 | 0.955 |

Averaged over all digits, the mean precision amounts to 97.37 % while the average recall is similarly high at 97.36 %.

## 5.2 Performance Benchmarks

This chapter includes runtime and communication overhead analysis.

The following benchmarks were accumulated on an Intel® i7-5600U CPU running at 2.6 GHz as the average over 3 individual runs with different test vectors, consistent accross different parameter runs.

Table 5.2: Performance Benchmarks / Communication Overhead

$\boldsymbol{B_1}$ ... Coefficient Moduli start bits (also equal to the last)
$\boldsymbol{B_2}$ ... Coefficient Moduli middle bits
$\boldsymbol{N}$ ... Polynomial Modulus Degree, found in the exponent of $p(X) = X^N + 1$
$\boldsymbol{T}$ ... Runtime of encryption, classification, decryption
$\boldsymbol{M}$ ... Message Size (Relin Keys + Galois Keys + Request Ciphertext + Response Ciphertext)
$\boldsymbol{\Delta}$ ... Mean Max-Relative Error compared to the exact result, i.e. $\frac{\langle|\boldsymbol{y}_{prediction} - \boldsymbol{y}_{exact}|\rangle}{\max|\boldsymbol{y}_{exact}|}$

| SecLevel | MatMul | $\boldsymbol{B_1}$ | $\boldsymbol{B_2}$ | $\boldsymbol{N}$ | $\boldsymbol{T}$ / s | $\boldsymbol{M}$ / MiB | $\boldsymbol{\Delta}$ / 1 | Mode |
|---|---|---|---|---|---|---|---|---|
| none | BSGS | 34 | 25 | 8192 | 2.9197 | 132.72 | 0.13616 | Release |
| none | Hybrid | 34 | 25 | 8192 | 10.6905 | 132.72 | 0.01408 | Release |
| none | BSGS | 60 | 40 | 16384 | 5.9881 | 286.50 | 0.13328 | Release |
| none | Hybrid | 60 | 40 | 16384 | 19.2554 | 286.50 | 0.00185 | Release |
| tc128 | BSGS | 34 | 25 | 8192 | 2.8693 | 132.72 | 0.13662 | Release |
| tc128 | Hybrid | 34 | 25 | 8192 | 9.0900 | 132.72 | 0.01359 | Release |
| tc128 | BSGS | 60 | 40 | 16384 | 5.9848 | 286.50 | 0.13328 | Release |
| tc128 | Hybrid | 60 | 40 | 16384 | 19.0962 | 286.50 | 0.00185 | Release |
| tc256 | BSGS | 60 | 40 | 32768 | 13.9787 | 615.16 | 0.13328 | Release |
| tc256 | Hybrid | 60 | 40 | 32768 | 41.8026 | 615.16 | 0.00185 | Release |
| tc128 | BSGS | 34 | 25 | 8192 | 7.2043 | 132.72 | 0.13650 | Debug |
| tc128 | Hybrid | 34 | 25 | 8192 | 13.2971 | 132.72 | 0.01369 | Debug |

TODO: Multi-row table for better overview? TODO: Interpretation der Tabelle

Without any encryption, the neural network classifies the full 10,000 image dataset in 515 ms on the same machine.

# Chapter 6

# Conclusion

TODO: To be written

Considering the implications of mass surveilance, the importance of privacy-preserving/enhancing technologies should not be forgotten.

## 6.1 Summary

TODO: To be written

## 6.2 Outlook

TODO: To be written: describe existing solutions, approaches, current research, etc.

## 6.3 Related Works

TODO: Vielleicht als kleiner Teaser für mehr Literatur?

Gazelle (inferred ML) as described by Juvekar, Vaikuntanathan and Chandrakasan 2018.

Random Forests (RF) on HE as described by Huynh 2020.

# Acronyms

| | | |
|---|---|---|
| BFV | Brakerski-Fan-Vercauteren | 5, 21 |
| BGV | Brakerski-Gentry-Vaikuntanathan | 5, 21 |
| CKKS | Cheon-Kim-Kim-Song | 5, 21 |
| CRT | Chinese Remainder Theorem | 37 |
| DFT | Discrete Fourier Transform | 25 |
| FFT | Fast Fourier Transform | 25 |
| FHE | Fully Homomorphic Encryption | 5, 21 |
| FHEW | Fastest Homomorphic Encryption in the West | 21 |
| GD | Gradient Descent | 18 |
| HE | Homomorphic Encryption | 5, 6, 7, 12, 21, 22, 34 |
| iff | if and only if | 10, 20 |
| LWE | Learning With Errors | 13, 14 |
| ML | Machine Learning | 17 |
| MNIST | Modified National Institute of Standards and Technology database | 6, 36 |
| MSE | Mean-Squared-Error | 17, 37 |
| NP | Non-deterministic Polynomial time | 5 |
| PPML | Privacy-Preserving Machine Learning | 6 |
| RLWE | Learning With Errors on Rings | 13 |
| RNS | Residue Number System | 37 |
| RSA | Rivest-Shamir-Adleman | 5 |
| TFHE | Torus Fully Homomorphic Encryption | 21 |
| TLS | Transport Layer Security | 15 |

# Definitions



# Theorems



# Corollaries



# Lemmata

# Remarks

# Bibliography


Ajtai, Miklós (1996). 'Generating hard instances of lattice problems (extended abstract)'. In: *STOC '96*.

Bishop, Christopher M. and Nasser M. Nasrabadi (2007). *Pattern Recognition and Machine Learning*. Vol. 16, p. 049901.

Brakerski, Zvika (2012). 'Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP'. In: *IACR Cryptol. ePrint Arch.* 2012, p. 78. URL: https://link.springer.com/content/pdf/10.1007%2F978-3-642-32009-5_50.pdf.

Brakerski, Zvika, Craig Gentry and Vinod Vaikuntanathan (2012). '(Leveled) fully homomorphic encryption without bootstrapping'. In: *ITCS '12*.

Cheon, Jung Hee, Andrey Kim, Miran Kim and Yongsoo Song (2017). 'Homomorphic Encryption for Arithmetic of Approximate Numbers'. In: *ASIACRYPT*.

Chillotti, Ilaria, Nicolas Gama, Mariya Georgieva and Malika Izabachène (2019). 'TFHE: Fast Fully Homomorphic Encryption Over the Torus'. In: *Journal of Cryptology* 33, pp. 34–91.

Corrigan-Gibbs, Henry, Sam Kim and David J. Wu (2018). *Lecture 9: Lattice Cryptography and the SIS Problem*. URL: https://crypto.stanford.edu/cs355/18sp/lec9.pdf (visited on 04/06/2022).

Dobraunig, Christoph, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger and Roman Walch (2021). 'Pasta: A Case for Hybrid Homomorphic Encryption'. In: *IACR Cryptol. ePrint Arch.* 2021, p. 731.

Ducas, Léo and Daniele Micciancio (2015). 'FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second'. In: *EUROCRYPT*.

Fan, Junfeng and Frederik Vercauteren (2012). 'Somewhat Practical Fully Homomorphic Encryption'. In: https://eprint.iacr.org/2012/144. URL: https://eprint.iacr.org/2012/144.

Gentry, Craig (2009). 'Fully homomorphic encryption using ideal lattices'. In: *STOC '09*.

Goldwasser, Shafi (2018). 'From Idea to Impact, the Crypto Story: What's Next?' In: URL: https://www.youtube.com/watch?v=culuNbMPP0k (visited on 01/03/2022).

Halevi, Shai and Victor Shoup (2018). *Faster Homomorphic Linear Transformations in HElib*. Cryptology ePrint Archive, Report 2018/244. https://ia.cr/2018/244.

Hornik, Kurt, Maxwell B. Stinchcombe and Halbert L. White (1989). 'Multilayer feedforward networks are universal approximators'. In: *Neural Networks* 2, pp. 359–366.

Huynh, Daniel (2020). 'Cryptotree: fast and accurate predictions on encrypted structured data'. In: DOI: 10.48550/ARXIV.2006.08299. URL: https://arxiv.org/abs/2006.08299.

Juvekar, Chiraag, Vinod Vaikuntanathan and Anantha P. Chandrakasan (2018). 'Gazelle: A Low Latency Framework for Secure Neural Network Inference'. In: *CoRR* abs/1801.05507. arXiv: 1801.05507. URL: http://arxiv.org/abs/1801.05507.

Kim, Andrey, Yuriy Polyakov and Vincent Zucca (2021). *Revisiting Homomorphic Encryption Schemes for Finite Fields*. Cryptology ePrint Archive, Paper 2021/204. https://eprint.iacr.org/2021/204. URL: https://eprint.iacr.org/2021/204.

LeCun, Yann and Corinna Cortes (1998). *The MNIST database of handwritten digits*. URL: http://yann.lecun.com/exdb/mnist/.

Lepoint, Tancrède and Michael Naehrig (2014). 'A Comparison of the Homomorphic Encryption Schemes FV and YASHE'. In: *AFRICACRYPT*.

Lyubashevsky, Vadim, Chris Peikert and Oded Regev (2010). 'On Ideal Lattices and Learning with Errors over Rings'. In: *EUROCRYPT*.

– (2013). 'A Toolkit for Ring-LWE Cryptography'. In: *IACR Cryptol. ePrint Arch.*

Nüst, Daniel, Vanessa Sochat, Ben Marwick, Stephen J. Eglen, Tim Head, Tony Hirst and Benjamin D. Evans (2020). 'Ten simple rules for writing Dockerfiles for reproducible data science'. In: *PLOS Computational Biology* 16.11, e1008316. DOI: 10.1371/journal.pcbi.1008316.

Peikert, Chris (2016). 'A Decade of Lattice Cryptography'. In: *IACR Cryptol. ePrint Arch.* 2015, p. 939.

ProofWiki (2020). *Cyclotomic Polynomial of Index Power of Two*. URL: https://proofwiki.org/wiki/Cyclotomic_Polynomial_of_Index_Power_of_Two (visited on 06/06/2022).

Regev, Oded (2005). 'On lattices, learning with errors, random linear codes, and cryptography'. In: *STOC '05*.

– (2010). 'The learning with errors problem'. English (US). In: *Proceedings - 25th Annual IEEE Conference on Computational Complexity, CCC 2010*. Proceedings of the Annual IEEE Conference on Computational Complexity. 25th Annual IEEE Conference on Computational Complexity, CCC 2010 ; Conference date: 09-06-2010 Through 11-06-2010, pp. 191–204. ISBN: 9780769540603. DOI: 10.1109/CCC.2010.26.

Rescorla, Eric (2018). *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/info/rfc8446.

Rivest, Ronald L, Adi Shamir and Leonard M Adleman (Sept. 1983). *Cryptographic communications system and method*. US Patent 4,405,829.

Serge, Lang (2002). *Algebra*. 3rd ed. Springer. DOI: 10.1007/978-1-4613-0041-0.

Shor, Peter W. (Oct. 1997). 'Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer'. In: *SIAM Journal on Computing* 26.5, pp. 1484–1509. DOI: 10.1137/s0097539795293172. URL: https://doi.org/10.1137%2Fs0097539795293172.

User, StackExchange (2020). *Plot gradient descent*. URL: https://tex.stackexchange.com/a/544832/155678 (visited on 07/07/2022).

# List of Figures

# Appendix

*Proof of Theorem 2.1.1.* With $k \in \mathbb{N}$ a positive integer, we want to show that

$$\Phi_{2^k}(x) = x^{2^{k-1}} + 1\,.$$

A polynomial $p \in \mathbb{Z}[X]$ with

$$p(x) = x^n - a$$

of degree $n$ has $n$ roots

$$\{x_j\} = \{a^{\frac{1}{n}} e^{2\pi i \frac{j}{n}} \,|\, j \in \mathbb{N}, j \leq n\}$$

related by a factor $a^{\frac{1}{n}}$ to the $n^{\text{th}}$ roots of unity given by powers of $\xi = e^{2\pi i \frac{1}{n}}$.

It is clear from the fundamental theorem of algebra that the polynomial $p$ with roots $\{x_j\}$ can be factorised as

$$p(x) = \prod_{j=1}^{n}(x - x_j) = \prod_{j=1}^{n}(x - a^{\frac{1}{n}} e^{2\pi i \frac{j}{n}})\,.$$

Fixing $a = -1$, we obtain $p(x) = x^n + 1$ with roots given by

$$x_j = (-1)^{\frac{1}{n}} e^{2\pi i \frac{j}{n}} = (e^{i\pi})^{\frac{1}{n}} e^{2\pi i \frac{j}{n}} = e^{\frac{i\pi(2j+1)}{n}}$$

and according factorisation

$$p(x) = \prod_{j=1}^{n}(x - e^{\frac{i\pi}{n}(2j+1)})\,.$$

Further letting $n = 2^{k-1}$ and observing that

$$\gcd(2^k, l) = \begin{cases} 1 & \text{if } l \text{ odd} \\ 2 & \text{if } l \text{ even} \end{cases} \qquad l, k \in \mathbb{N}$$

since a number $2^k$ that can only be decomposed into multiples of 2 never shares a factor with an odd number, in accordance with Lemma 2.1.2 we can conclude that the set of all odd roots of unity is exactly the set of all primitive roots (satisfying $\gcd(2^k, l) = 1$).

Following from above,

$$p(x) = \prod_{j=1}^{2^{k-1}}(x - e^{\frac{i\pi}{n}(2j+1)}) = \prod_{\substack{l=1 \\ l \text{ odd}}}^{2^k}(x - e^{\frac{i\pi}{n} l}) = \prod_{\substack{l=1 \\ \xi^l \text{ primitive}}}^{2^k}(x - \xi^l) = \Phi_{2^k}(x)$$

we arrive exactly at the definition of a cyclotomic polynomial (Definition 2.1.5).
(ProofWiki 2020) □

*Proof of Theorem 4.3.2.* Starting from the adapted matrix-multiplication expression $P = (P_1, P_2, ..., P_t)^T \in \mathbb{R}^t$, we want to show that we indeed end up with an authentic matrix-vector product.

$$P = \left\{ \sum_{k=0}^{t_2-1} \mathrm{rot}_{(kt_1)} \Big( \sum_{j=0}^{t_1-1} \mathrm{diag}'_{(kt_1+j)}(M) \cdot \mathrm{rot}_j(\boldsymbol{x}) \Big) \right\}_i = \sum_{k=0}^{t_2-1} \sum_{j=0}^{t_1-1} m'_{kt_1+j,(i+kt_1)} x_{(i+kt_1)+j}$$

with

$$m'_{p,i} = \left\{ \mathrm{diag}'_p(M) \right\}_i = \left\{ \mathrm{rot}_{-\lfloor p/t_1 \rfloor \cdot t_1}(\mathrm{diag}_p(M)) \right\}_i = M_{i-\lfloor \frac{p}{t_1} \rfloor t_1, i-\lfloor \frac{p}{t_1} \rfloor t_1 + p}$$

and therefore

$$\begin{aligned}
m'_{kt_1+j,i} &= M_{i-\lfloor \frac{kt_1+j}{t_1} \rfloor t_1, i-\lfloor \frac{kt_1+j}{t_1} \rfloor t_1 + kt_1 + j} \\
&= M_{i-kt_1-\lfloor \frac{j}{t_1} \rfloor t_1, i-kt_1-\lfloor \frac{j}{t_1} \rfloor t_1 + kt_1 + j} \\
&= M_{i-kt_1-\lfloor \frac{j}{t_1} \rfloor t_1, i+j-\lfloor \frac{j}{t_1} \rfloor t_1} \\
m'_{kt_1+j,(i+kt_1)} &= M_{i+kt_1-kt_1-\lfloor \frac{j}{t_1} \rfloor t_1, i+kt_1+j-\lfloor \frac{j}{t_1} \rfloor t_1} \\
&= M_{i-\lfloor \frac{j}{t_1} \rfloor t_1, i+kt_1+j-\lfloor \frac{j}{t_1} \rfloor t_1}
\end{aligned}$$

leading to

$$P_i = \sum_{k=0}^{t_2-1} \sum_{j=0}^{t_1-1} m'_{kt_1+j,(i+kt_1)} x_{(i+kt_1)+j} = \sum_{k=0}^{t_2-1} \sum_{j=0}^{t_1-1} M_{i-\lfloor \frac{j}{t_1} \rfloor t_1, i+kt_1+j-\lfloor \frac{j}{t_1} \rfloor t_1} x_{(i+kt_1)+j} \,.$$

Noticing that the downward rounded fraction $\lfloor \frac{j}{t_1} \rfloor$ vanishes in a sum with $j$ running from 0 to $t_1 - 1$, we can simplify to

$$P_i = \sum_{k=0}^{t_2-1} \sum_{j=0}^{t_1-1} M_{i,i+kt_1+j} x_{i+kt_1+j}$$

which contains two sums running to $t_1$ and $t_2$ respectively, containing an expression of the form $k \cdot t_1 + j$, which allows us to condense the nested sums into one single summation expression, as

$$\sum_{k=0}^{t_2-1} \sum_{j=0}^{t_1-1} f(kt_1 + j) = \sum_{l=0}^{t-1} f(l)$$

indeed catches every single value $l \in \{0, 1, 2, ..., t = t_1 \cdot t_2\}$ with $l = kt_1 + j$.
In summary, we obtain

$$\begin{aligned}
P_i &= \sum_{k=0}^{t_2-1} \sum_{j=0}^{t_1-1} M_{i,i+kt_1+j} x_{i+kt_1+j} \\
&= \sum_{l=0}^{t-1} M_{i,i+l} x_{i+l} = \sum_{\nu=0}^{t-1} M_{i,\nu} x_\nu \\
&= \left\{ M\boldsymbol{x} \right\}_i
\end{aligned}$$

which indeed equals the conventional definition of a matrix-vector product. □