IAIK

SCIENCE ▪ PASSION ▪ TECHNOLOGY

TU Graz

Peter Julius Waldert

Secure Classification as a Service

BACHELOR'S THESIS

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

Supervisors

Dipl.-Ing. Roman Walch
Dipl.-Ing. Daniel Kales

Institute of Applied Information Processing and Communications
Graz University of Technology

Graz, March 2022

Abstract

Abstract of your thesis (at most one page)

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

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

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

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

Goal:

Chapter 2

Background

2.1 Basics of Fully Homomorphic Encryption

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.

- Brakerski/Fan-Vercauteren (BFV) scheme for integer arithmetic (Fan and Vercauteren 2012, Brakerski 2012).
- Brakerski-Gentry-Vaikuntanathan (BGV) scheme for integer arithmetic (Brakerski, Gentry and Vaikuntanathan 2012).
- Cheon-Kim-Kim-Song (CKKS) scheme for real-number arithmetic (Cheon et al. 2017).
- Ducas-Micciancio (FHEW) and Chillotti-Gama-Georgieva-Izabachene (TFHE) schemes 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.

2.1.1 Mathematical Foundation

The following discussion of the homomorphic encryption schemes requires some mathematical background that will (at least partially) be introduced here.

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 a 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 modulo t: $\mathbb{Z}/t\mathbb{Z}$, sometimes also denoted as $\mathbb{Z}_t$.

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

2.1.3 Definition (Ring of Integers Modulo t: $\mathbb{Z}/t\mathbb{Z}$)

Using equivalence classes $\overline{x}_t$ modulo t referred to as congruence classes, define the commutative quotient ring of integers modulo t as

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

where $t\mathbb{Z} \triangleleft \mathbb{Z}$ denotes the t^{th} coset[a] of the integers and

$$\overline{x}_t = \{y \equiv x \mod t \,|\, y \in \mathbb{Z}\}$$

is the set of all multiples of t with remainder x.

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

2.1.4 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 $(\mathbf{p})_i = p_0, p_1, p_2, ...$ (which are sequences) and $\mathbf{q} = \{q_0, q_1, q_2, ...\}$, an addition of two polynomials $p, q \in \mathbb{Z}[X]$ is equivalent to the addition of

their coefficients

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

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 $\{r_k\}$ determined by the discrete convolution

$$r_k = \sum_{l=0}^{k} p_l q_{k-l} \;\Leftrightarrow\; \mathbf{r} = \mathbf{p} * \mathbf{q}$$

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

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

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.

2.1.5 Definition (Irreducible Polynomials)

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

Polynomials with degree ≥ 2 over the complex numbers can always be factorised using their roots due to the fundamental theorem of algebra.

2.1.1 Corollary (Polynomials Modulo an Irreducible Form)

Given an irreducible polynomial $\phi(x) \in P$, one can construct the interesting quotient group

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

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

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

2.1.2 Cyclotomic Polynomials

Due to their interesting structure and efficient computability, in the following schemes, polynomials modulo an irreducible form (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^{th} roots of unity.

2.1.1 Lemma (The n^{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. 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^{th} root of unity y is referred to as *primitive*, iff[a] there exists no $m < n$ for which that root y is also an m^{th} root of unity, i.e. $y^m \neq 1$.

[a] if and only if

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.

2.1.6 Definition (Cyclotomic Polynomial)

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

$$\Phi_n(x) = \prod_{k=1}^{\varphi(n)} (x - x_k)$$

with $\varphi(n)$ denoting Euler's totient function which counts the natural numbers m less than n who do not share a common divisor $\neq 1$, i.e. $\gcd(m, n) = 1$. It is unique for each given $n \in \mathbb{N}$.

2.1.1 Remark (Irreducibility of Cyclotomic Polynomials)

Cyclotomic polynomials are always irreducible..?

2.1.3 HE 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 (Definition 2.1.7) behind a cryptographic background.

With unpadded RSA, 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:

2.1.7 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:

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

2.1.4 Learning with Errors (LWE)

Next, we would like to consider Learning With Errors (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 (c.f. section 2.3).

2.1.5 Ring-LWE

Learning With Errors on Rings (RLWE)

how to get from LWE to RLWE

2.1.6 The BFV scheme

Fan and Vercauteren 2012 Brakerski 2012

2.1.7 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 $\mathbf{y} \in \mathbb{Z}_t^n$, by an embedding approach that allows us to encode a (complex) floating point number vector $\mathbf{x} \in \mathbb{R}^n(\mathbb{C}^n)$ as an integer vector. A naïve approach would be to use a fixed-point embedding:

$$\text{embed}(\mathbf{x}) = \mathbf{x} \cdot F$$

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

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

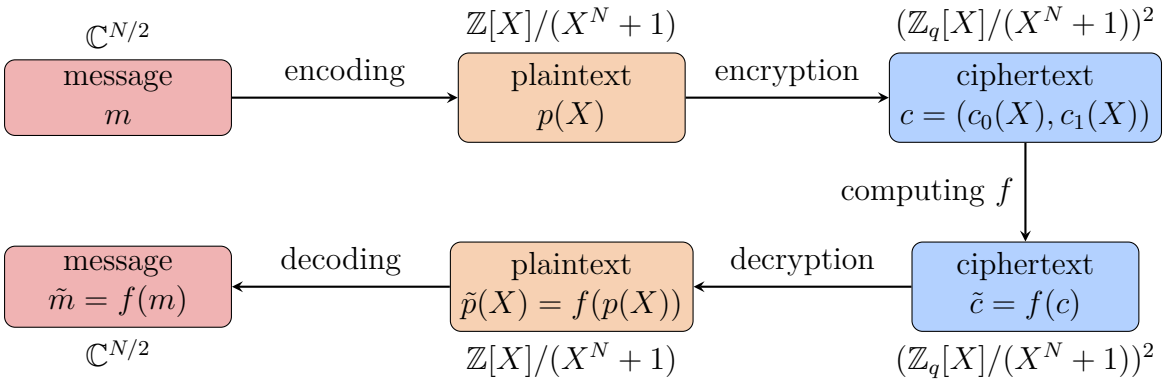

Figure 2.1: Overview of CKKS, adapted from Huynh 2020.

2.2 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 first used 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).

2.2.1 Linear Regression

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

2.2.2 Gradient Descent

2.2.3 Multi-Layered Neural Networks

2.2.1 Theorem (Universal Approximation)

f 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).

Matrix -> Activation Function

- Matrix Multiplication (Dense Layer)
- Convolutional Layer
- Sigmoid Activation
- Max Pooling

2.2.4 The Backpropagation Algorithm

2.3 Post-Quantum Security

2.3.1 Shor's Algorithm

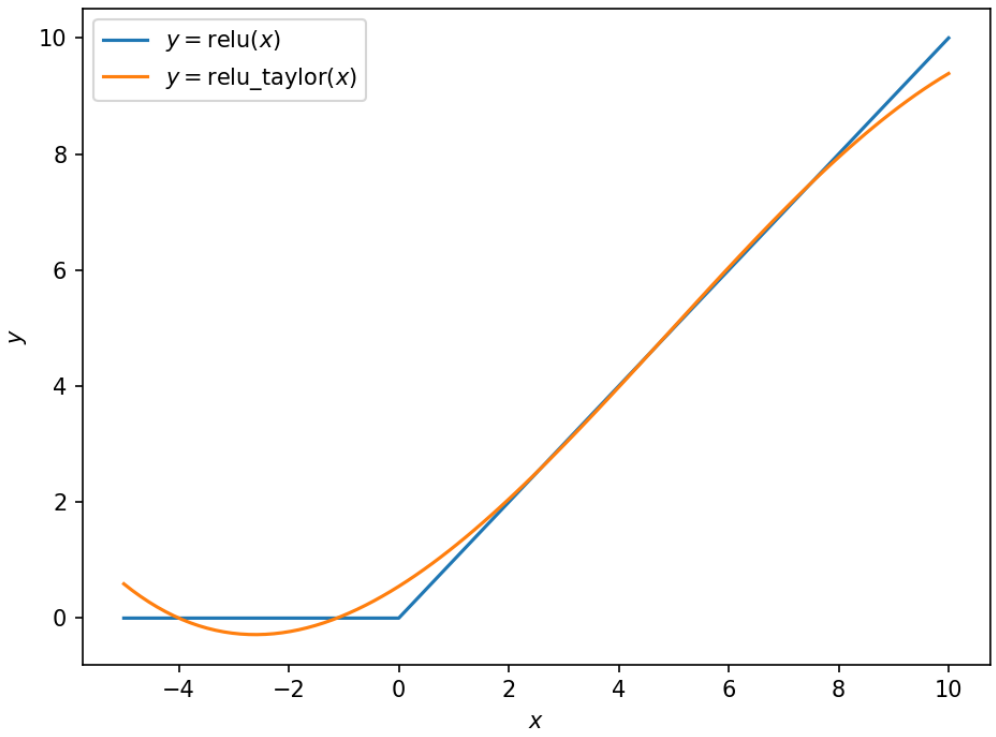

Figure 2.2: Comparison of the Relu activation function vs. its Taylor expansion

Chapter 3

Implementation

3.1 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

```
1  docker-compose build
2  docker-compose up
```

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

3.1.1 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.

3.2 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.

3.3 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

$$\mathrm{rot}_j : \mathbb{R}^t \mapsto \mathbb{R}^t,\ \{\mathrm{rot}_j(\mathbf{x})\}_i = x_{i+j} \tag{3.1}$$

$$\mathrm{diag}_j : \mathbb{R}^{t\times t} \mapsto \mathbb{R}^t,\ \{\mathrm{diag}_j(M)\}_i = M_{i,(i+j)} \tag{3.2}$$

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.

3.3.1 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.

3.3.2 The Naïve Method

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

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

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

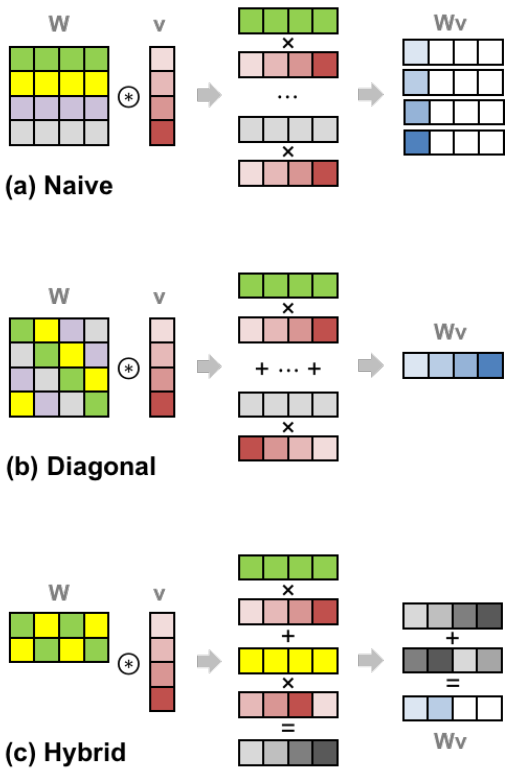

Figure 3.1: Different techniques to compute a dot product between a matrix and a vector, each having their up- and downsides.

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 $\mathbf{x}$.
2. multiply each encoded row with the encrypted vector using `Evaluator.multiply_plain()` to obtain the ciphertext vector $\mathbf{y_i} \in \mathbb{R}^s$ for row i.
3. perform the 'rotate-and-sum' algorithm (Juvekar, Vaikuntanathan and Chandrakasan 2018) on each resulting vector (ciphertext) $\mathbf{y_i}$ to obtain the actual dot product of the matrix row with the vector $\mathbf{x}$:
   (a) using Galois automorphisms, rotate the entries of $\mathbf{y_i}$ by $\frac{s}{2}$ elements to obtain $\text{rot}_{\frac{s}{2}}(\mathbf{y_i})$.
   (b) perform an element-wise sum $\mathbf{y_i} + \text{rot}_{\frac{s}{2}}(\mathbf{y_i})$ whose first (and also second) half now contains the sum of the two halves of $\mathbf{y_i}$.
   (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 $\sum_{k=1}^{s}\{\mathbf{y_i}\}_k$ as the dot product of $\mathbf{x}$ and $\mathbf{y_i}$.
4. Given all the 'scalar' results of each row-vector dot product, we can construct the resulting matrix-vector product.

3.3.3 The Diagonal Method

3.3.1 Theorem (Diagonal Method)

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

$$M\mathbf{x} = \sum_{i=0}^{t} \text{diag}_i(M)\text{rot}_i(\mathbf{x}) \tag{3.3}$$

3.3.4 The Hybrid Method

To further extend the previous matrix multiplication method to solve the problem (cf. subsection 3.3.1), 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:

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

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
}
```

3.3.5 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:

3.3.2 Theorem (Babystep-Giantstep Optimization)

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

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

, one can express a matrix-vector multiplication as follows:

$$M\mathbf{x} = \sum_{k=0}^{t_2-1} \text{rot}_{(kt_1)}\left(\sum_{j=0}^{t_1-1} \text{diag}'_{(kt_1+j)}(M) \cdot \text{rot}_j(\mathbf{x}) \right) \tag{3.4}$$

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

Proof. 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.

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

with

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

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

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

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

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 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 in the Diagonal or Hybrid method for instance which was the point of this modification in the first place.

3.4 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()`)

   > Scale has nothing to do with noise. "Scale out of bounds" can appear even if noise is extremely low. Although repeated multiplication of a ciphertext by a plaintext will slowly increase the noise, it is not the reason why you see "scale out of bounds". "Scale out of bounds" error specifically means that the scale of a ciphertext or plaintext is larger than the product of all elements in coeff_modulus. If you perform multiplications without rescaling, you can quickly see this error. The more rescaling you perform, the less elements will be left in coeff_modulus. Even if you managed to have the same scale in a ciphertext after every multiplication and rescale, eventually the coeff_modulus can be too small to accommodate another multiplication.

   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()`

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 log2(scale) bits.

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

3.5 Transparent Ciphertext

> The problem is that you are subtracting a ciphertext from itself. This kind of operation results in a ciphertext that is identically zero; this is called a transparent ciphertext. Such a transparent ciphertext is not considered to be valid because the ciphertext reveals its underlying plaintext to anyone who sees it, even if they don't have the secret key. By default SEAL throws an exception when such a situation is encountered to protect you from a problem you may not have noticed. If you truly know what you are doing and want to enable the creation of transparent ciphertexts, you can configure SEAL [...]. (Laine 2020)

'transparent ciphertexts (a.k.a. ciphertexts whose second polynomial is zero) are malformed and do not need the secret key to decrypt'

Chapter 4

Results

4.1 Performance / Runtime Benchmarks

4.2 Communication Overhead

4.3 Accuracy

Chapter 5

Conclusion

5.1 Outlook

Gazelle (inferred ML) as described by Juvekar, Vaikuntanathan and Chandrakasan 2018.
Random Forests (RF) on HE as described by Huynh 2020.

Acronyms

FHE	Fully Homomorphic Encryption	6
HE	Homomorphic Encryption	6, 9, 16
LWE	Learning With Errors	10
ML	Machine Learning	11
RLWE	Learning With Errors on Rings	10

Bibliography

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.

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.

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.

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.

Laine, Kim (2020). *Result Ciphertext is Transparent*. GitHub. URL: https://github.com/microsoft/SEAL/issues/224#issuecomment-702516479 (visited on 04/03/2022).

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

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.

List of Figures

List of Tables