An Overview on Homomorphic Encryption Algorithms

V. F. Rocha   *Julio López*

Technical Report – IC-PFG-18-28
Final Graduation Project
November - 2018

STATE UNIVERSITY OF CAMPINAS
INSTITUTE OF COMPUTING

An Overview on Homomorphic Encryption Algorithms

Vitor Falcão da Rocha, Julio López*

January 2019

Abstract

Homomorphic encryption is a cryptographic construction that allows an application to operate on ecrypted data, instead of the raw data itself. The possibility of homomorphic encryption had long been studied due to the applications it enables, especially the outsourcing of computing on sensitive data, while preserving the privacy of the data. Not until 2009 a fully homomorphic encryption scheme (FHE) was proposed by Craig Gentry [11], though not practical, this scheme paved the way for other schemes, both based on Gentry's ideas [13], [12] and others based on new ideas [17], [21]. In parallel, some applications were developed to operate on data encrypted by those FHE schemes, proving that FHE could power practical applications, in a more privacy preserving fashion. This paper presents the concepts that support modern homomorphic encryption schemes, together with the description of some FHE schemes. In addition, some practical applications for processing sensitive data are presented.

1 Introduction

Homomorphic encryption is a cryptographic method that allows mathematical operations on data to be carried out on a cipher text, instead of on the actual data itself. The cipher text is an encrypted version of the input data (also called plain text), that is operated on and then decrypted to obtain the desired output. The critical property of homomorphic encryption is that the same output should be obtained from decrypting the operated cipher text as from simply operating on the initial plain text. There are many interesting applications that arise from a system that is capable of doing homomorhpic operations, such as the processing of medical data, preserving the privacy of the patient and of the model [6], outsourcing of financial operations, anonymous database queries [4] and a more privacy preserving advertising system [5].

An encryption scheme that has the property described above is said to be a homomorhpic encryption scheme. Those schemes come in 3 forms: somewhat, partially and fully homomorphic. Somewhat homomorphic encryption supports mathematical operations with respect to adition and multiplication, but is limited to a certain number of operations since each operations adds noise and after a certain amount of noise is added, it is no longer possible to retrieve the data. Partially homomorhpic encryption on the other hand supports any number of operations, but is limited to just one type of operation. Finally, fully homomorhpic encryption supports both addition and multiplication, applied any number of times to the data.

Examples of partially homomorphic encryption schemes include the Paillier [9] cryptosystem, RSA and ElGamal. Paillier is additively homomorhpic, while RSA and ElGamal are both multilicatively homomorphic. RSA [8], for example, is multiplicatively homomorphic, because given two ciphertexts $c_1 = m_1^e (\bmod q)$ and $c_2 = m_2^e (\bmod q)$, where q is the product of two large primes and e

*Institute of Computing, UNICAMP, 13083-852 Campinas, SP

is such that $gcd(e, \phi(q)) = 1$, $c_3 = c_1 * c_2 = m_1^e * m_2^e (\mathrm{mod} q) = (m_1 * m_2)^e (\mathrm{mod} q)$. Therefore, the decryption of c_3 is equal to $m_1 * m_2$.

Fully homomorhpic encryption schemes had not been developed until 2009, when Craig Gentry published a seminal work [11], which introduced the idea of bootstraping. Since then many FHE schemes have been developed based on lattices and the learning with errors problem (LWE). Specifically, the schemes have been developed on the Ring-LWE problem, which is the ring version of LWE.

Since Gentry's seminal work many fully homomorphic encryption schemes have been developed [19][17][20][21][13] (to cite a few), with some using the bootstrapping technique and some leveraging other more efficient techniques for dealing with the noise that affects homomorphic operations. The problem with all those schemes is that they still lack efficiency, which does not make it possible to deploy FHE schemes in large scale.

In parallel with the development of fully homomorphic encryption schemes, many applications operating on encrypted were proposed [34] [5] [4] [6] [30], ranging from private database queries, to machine learning classification algortihms. Though not overall very practical, since their performance heavily degraded with the increase in the volume of data, the proposals showed how day to day tasks could be done in a more privacy aware manner.

In Section 2 the notation used throughout this paper is presented, while Section 3 will introduce the concept of a lattice and some problems on lattices. Section 4 presents some FHE schemes and Section 5 shows some implementations, like HElib [23]. Section 6 will give some examples of applications that were developed to operate encrypted data, including machine learning classification. Finally, Section 7 presents the conclusion of this work.

2 Notation

Throughout this article the notation $\vec{v}$ denotes a vector and capital letters denote matrices (e.g., A). The dot product of two vectors $\vec{a}, \vec{b}$ is denoted by $\langle \vec{a}, \vec{b} \rangle$ and the product between a vector $\vec{v} \in R^n$ and a matrix $A \in R^{n \times m}$, is written as $A\vec{v}$ or $A \cdot \vec{v}$, for some ring R. The product of a scalar c and a vector $\vec{v}$ is denoted by $c \cdot \vec{v}$ or simply $c\vec{v}$. The tensoring of two vectors $\vec{a}, \vec{b}$ is represented as $\vec{a} \otimes \vec{b}$.

The polynomial rings used in this paper are of the form $R = \mathbb{Z}[x]/(f(x))$, where $f(x)$ is an irreducible polynomial. Elements of a polynomial ring R are represented by a bold symbol $\boldsymbol{x} \in R$ and the product of two ring elements $\boldsymbol{a}, \boldsymbol{b}$ is denoted by $\boldsymbol{a} \cdot \boldsymbol{b}$. The notation R_q represents a polynomial ring where the coefficients are modulus q.

For $x \in \mathbb{R}$, we denote by $\lfloor x \rceil$ the rounding of x to its nearest intereger and by $\lceil x \rceil$ and $\lfloor x \rfloor$, the rouding up and down of x, respectively. Also, $\log x$ denotes $\log_2 x$.

For a distribution χ, $a \leftarrow \chi$ denotes the withdrawal of a from χ uniformly at random.

For a binary number a, a_i denotes its i-th bit ordered least significant to most significant. For an integer δ, $a \gg \delta$ denotes the right shift of a by δ positions. The $\wedge$ operator represents the logical AND, where for two binary numbers a, b, $a \wedge b$ denotes the bitwise AND operation and $a_i \wedge a_j$ represents the logical AND between the i-th and j-th bit of a. The same logic of $\wedge$ applies to the OR operator $\vee$ and the XOR operator $\oplus$.

3 Lattices

Lattice-based cryptography has, so far, showed itself as a very promising path to pursue in cryptography, because it is based on simple mathematical operations, which result in low computational costs, it is highly parallelizable, enables FHE, so far it is resistant to quantum attacks [40] and most important: it is based on worst-case hardness, a property always sought by cryptographers, but only achieved with lattice-based crypto. Worst-case hardness here means that given any instance of a lattice problem, solving it is at least as hard as solving several lattices problems in the worst case. There are other very interesting applications that lattices enable, but we are going to focus here on FHE.

A lattice L is an algebraic structure defined as the set of linear combinations of linearly independent vectors $b_1, ..., b_n \in \mathbb{R}^m$ with coefficients in $\mathbb{Z}$ [2]:

$$\mathrm{L} = \{\mathrm{a}_1 b_1 + a_2 b_2 + ... + a_n b_n : a_1, a_2, ..., a_n \in \mathbb{Z}\} \tag{1}$$

We refer to $b_1, ..., b_n$ as a *basis* for L [3], as is any set of linearly independent vectors that generates L.

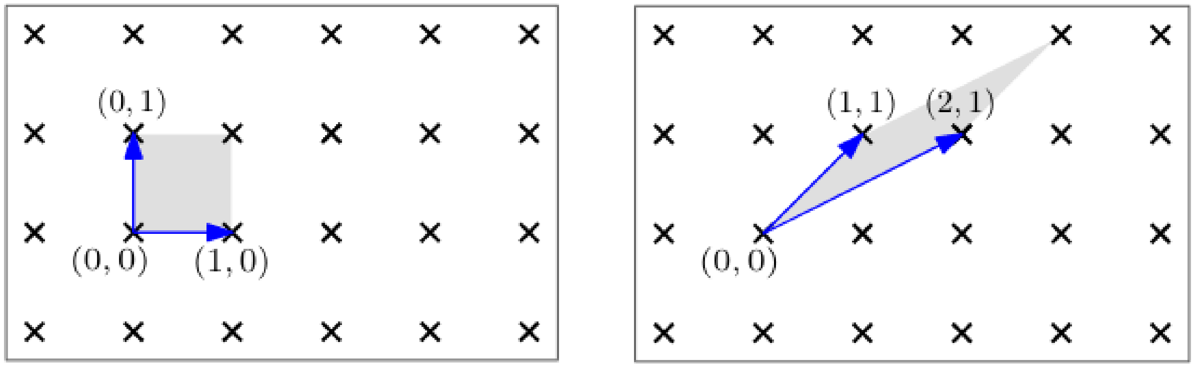

Figure 1: Example of a lattice ($\mathbb{Z}^2$) and 2 possible basis for it [3].

Below some lattice problems are defined, namely SIS, LWE,ring-LWE and Sparse Subset Sum (SSSP) together with some concepts in lattices. This will serve as the necessary background to understand the problems that underpin the FHE schemes presented in the next section.

3.1 Short Integer Solution And Short Vector Problem

The SIS problem can be formulated as given an uniformly random matrix $A \in \mathbb{Z}_q^{n \times m}$, find a vector $\vec{z} \in \mathbb{Z}^m$ such that $A\vec{z} = 0$ (over $\mathbb{Z}_q^n$), $\| \vec{z} \| \leq \beta$ and $\vec{z} \neq \vec{0}$, that is, $\vec{z}$ cannot be trivial. Note that $\beta < q$, otherwise $\vec{z} = (q, 0, ..., 0) \in \mathbb{Z}^m$ would be a valid, but trivial solution.

It is now possible to see a relation to lattices: let S be the set of all possible solutions to an instance of the SIS problem, S is a lattice since all its vectors are linearly independent and in $\mathbb{Z}$. When seen from the aspect of lattices, the SIS problem becomes the SVP (Short Vector Problem).

3.2 Learning With Errors

The LWE problem is very similar to the SIS problem and has a search and a decision version. This problem is parametrized by positive integers n, q and an error distribution χ over $\mathbb{Z}$, which is often a discrete Gaussian [1]. Before introducing the two versions of the problem, their common structure is presented. This structure is the LWE distribution and is defined as:

Definition 1 *For a secret $\vec{s} \in \mathbb{Z}_q^n$, the LWE distribution $A_{\vec{s},\chi}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by choosing $\vec{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \leftarrow \chi$ and outputting:*

$$(\vec{a}, b = \langle \vec{s}, \vec{a} \rangle + e \mod q) \tag{2}$$

Oded Regev showed the main hardness results for LWE [16]:

- Solving LWE for an n-dimensional lattice with modulus $q = poly(n)$ implies an equally efficient solution to a hard lattice problem in dimension $\sqrt{n}$. Saying that a lattice problem is hard means that the best known algorithm is exponential in the dimension of the lattice.
- In a quantum setting, one can replace $\sqrt{n}$ with n in the claim above.

3.2.1 Search-LWE

The search-LWE problem asks to recover a secret $\vec{s} \in \mathbb{Z}_q^n$ given a sequence of approximate random linear equations on s. More precisely, given a vector $\vec{s} \in \mathbb{Z}_q^n$ called the secret, a distribution $A_{\vec{s},\chi} \in \mathbb{Z}_q^{n\times m}$, and linear equations on $\vec{s}$ of the form:

$$(\vec{a}, b = \langle \vec{s}, \vec{a} \rangle + e \mod q) \tag{3}$$

where $e \longleftarrow \chi$ is a small error, find $\vec{s}$.

If the linear equations were not pertubed by a small error term e, the system of linear equations could be easily solved with Gaussian Elimination.

3.2.2 Decision-LWE

The decision-LWE problem asks to determine that for m independent samples $(\vec{a_i}, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where every sample is distributed according to $A_{\vec{s},\chi} \in \mathbb{Z}_q^{n\times m}$ or the uniform distribution, if the sample was drawn from the former or the latter distribution.

3.3 Ideal Lattices

Before defining an ideal lattice, firstly the definition of cyclic lattice is given:

Definition 2 *We say that a set $L \in \mathbb{Z}^n$ is a cyclic lattice if:*

1. *For all $v, w \in L$, $v + w$ also is in L.*
2. *For all $v \in L$, $-v$ is also in L.*
3. *For all $v \in L$, a rotation of it is also in L.*

Definition 3 *We say that an ideal lattice is a cyclic lattice that corresponds to an ideal $\mathbb{Z}[x]/\langle f(x)\rangle$, where f has the properties below [15]:*

1. *Monic, that is, the coefficient of the largest exponent is 1.*
2. *Irreducible over $\mathbb{Z}$.*

3. *For all polynomials g, h in the ring, the norm of their products should be less than a polynomial approximation of the product of their norms.*

Ideal lattices are very interesting because it is possible to represent an n-dimensional lattice with just 1 vector, using the properties of cyclic lattices. Another very interesting characteristic is that it is possible to do fast arithmetic, for example, using the Fast Fourier Transform.

3.4 Ring-LWE

The more general definition of Ring-LWE is based on a ring ring $R = \mathbb{Z}[x]/(f(x))$, but the more classical setup uses the specific ring $R = \mathbb{Z}[x]/(x^n + 1)$, where n is a power of 2. As with LWE, the ring definition also allows for two versions of the problem, a search and a decision one, but only the decision version will be discussed, since it is the most widely used by encryption schemes. This problem is parametrized by the ring R shown, by a positive integer modulus q, which defines the quotient ring R_q, and an error distribution χ over R [1]. Before introducing the decision versions of the problem, its basic structure is presented. This structure is the RLWE distribution and is defined as:

Definition 4 *For a secret $\boldsymbol{s} \in R_q$, the RLWE distribution $A_{s,\chi}$ over $R_q \times R_q$ is sampled by choosing $\boldsymbol{a} \in R_q$ uniformly at random, choosing $e \leftarrow \chi$ and outputting:*

$$(\boldsymbol{a}, b = s \cdot a + e \mod q) \tag{4}$$

3.4.1 Decision-RLWE

The decision-RLWE problem asks to determine if a given sample $(\boldsymbol{a}_i, b_i) \in R_q \times R_q$ was taken from the uniform distribution or from $A_{s,\chi}$, for m independent samples.

As with LWE this problem would be easy to solve if no error had been introduced, since one could find $\boldsymbol{s}$ by inverting $\boldsymbol{a}_i \in R_q$, since many elements in R_q are invertible [1]:

$$\boldsymbol{s} = b_i \cdot \boldsymbol{a}_i^{-1} \tag{5}$$

3.5 Sparse Subset Sum Problem (SSSP)

The Sparse Subset Sum Problem is defined here due to the important role it plays on Gentry's [11] fully homomorphic encryption scheme, which was the first FHE scheme designed. How the SSSP is used on Gentry's scheme is described later in section 3.1.1.

The SSSP can be seen as a type of knapsack problem, where it asks if there is a sparse knapsack that sums up to 0 mod q, where q is a prime positive integer [11]. More formally the SSSP is defined as let $\gamma(n)$ and $\eta(n)$ be the sizes of a set and a subset, respectively, and q a prime positive integer. There is a challenger that sets $b \leftarrow 0, 1$ randomly.

- If $b = 0$ it generates τ as a set of $\gamma(n)$ integers $\{a_1, ..., a_{\gamma(n)}\}$ in $[-q/2, q/2]$ that are uniformly random, subject to the constraint that there exists a subset $S \subseteq 1, ..., \gamma(n)$ of cardinality $\eta(n)$, such that $\sum_{i \in S} a_i = 0 \mod q$.
- If $b = 1$ it generates τ as a set of $\gamma(n)$ integers $\{a_1, ..., a_{\gamma(n)}\}$ in $[-q/2, q/2]$ that are uniformly random. That is, it generates the set without the constraint for when $b = 0$.

The problem is to correctly guess the value of b, given τ.

4 Fully Homomorphic Encryption

An encryption scheme is called Fully Homomorphic if it allows an unlimited number of evaluation operations on the encrypted data and the resulting output is within the ciphertext space. Such a scheme was built for the first time in Craig Gentry's 2009 Stanford PhD thesis. Gentry's scheme motivated a large amount of research in the area, mainly focused on improving the efficiency. Some more efficient schemes were proposed, but later other design approaches were developed, which led to even more efficient schemes [13] [19].

4.1 Gentry's Scheme

In this scheme, Craig starts by developing a SHE scheme and then applies two techniques to turn it into an FHE scheme. As mentioned earlier, SHE schemes add noise to the ciphertext in each homomorphic operation, thus, at a certain level the amount of noise is so high, that decryption becomes impossible. Therefore, to achieve full homomorhpic encryption from an SHE scheme, onde needs to keep the noise level below a certain bound and to that end Gentry introduced two concepts called *squashing* and *bootstrapping*. We will first present the scheme and then explain what are the *squashing* and *bootstrapping* techniques.

To build the scheme, Craig used ideals and rings, where each ideal in $\mathbb{Z}/(f(x))$, with $f(x)$ of degree n, would be represented by a lattice base B_I of size n. The key pair (B_J^{sk}, B_J^{pk}) is generated from a given ring R and the basis B_I, where B_J^{sk} is a good basis for the lattice and B_J^{pk} a bad basis for the same lattice. A good basis for a lattice is defined as a set of linearly independent vectors that are almost orthogonal to each other and have a small norm, while a bad basis is defined as a set of linearly independent vectors that are very skewed to each other and have a bigger norm. Encryption follows by choosing random vectors $\vec{r}, \vec{g}$ using the public key, which is B_J^{pk}. The message $\vec{m} \in \{0,1\}^n$ is encrypted as follows:

$$\vec{c} = E(\vec{m}) = \vec{m} + \vec{r} \cdot B_I + \vec{g} \cdot B_J^{pk} \tag{6}$$

The cyphertext $\vec{c}$ is decrypted as follows:

$$\vec{m} = D(\vec{c}) = \vec{c} - B_J^{sk} \cdot \lceil (B_J^{sk})^{-1} \cdot \vec{c} \rfloor \mod B_I \tag{7}$$

Homomorphism with respect to addition and multiplication is now verified, starting with addition:

$$\vec{c_1} + \vec{c_2} = E(\vec{m_1}) + E(\vec{m_2}) = \vec{m_1} + \vec{m_2} + (\vec{r_1} + \vec{r_2}) \cdot B_I + (\vec{g_1} + \vec{g_2}) \cdot B_J^{pk} \tag{8}$$

When decrypting the above sum, one gets $\vec{m_1} + \vec{m_2} + (\vec{r_1} + \vec{r_2}) \cdot B_I$ for cyphertexts that have noise smaller than $B_J^{pk}/2$. Homomorphism with respect to multiplication is shown below:

$$\vec{c_1} \times \vec{c_2} = E(\vec{m_1}) \times E(\vec{m_2}) = \vec{e_1}\vec{e_2} + (\vec{e_1} \times \vec{g_2} + \vec{e_2} \times \vec{g_1} + \vec{g_1} \times \vec{g_2}) \cdot B_J^{pk} \tag{9}$$

where $\vec{e_1} \times \vec{e_1} = \vec{m_1} \times \vec{m_2} + (\vec{m_1} \times \vec{r_2} + \vec{m_2} \times \vec{r_1} + \vec{r_1} \times \vec{r_2})$.

In order to turn this SHE scheme into an FHE Gentry needed to keep the amount of noise in the ciphertext below a certain bound, after which it is not possible to recover the message. To this end he came up with the *bootstrapping* procedure, which allows one to get a new ciphertext with less noise than the noisy ciphertext, for the same plaintext, but the ciphertext must be boostrappable. What this means is that the scheme that generated the ciphertext must be able to evaluate its own descryption algorithm circuit, plus one homomorphic operation. In order to make the ciphertext bootstrappable, Gentry came up with the concept of *squashing*.

4.1.1 Squashing

This procedure consists of multiplying the ciphertext by the elements of a set of vectors, which will basically reduce the polynomial degree of the decrytion circuit to a level that the scheme can handle. This set of vectors consists of a secret sparse subset of vectors, which have a special property where their sum is equal $(B_J^{sk})^{-1}$ [11]. The security of this procedure relies on the set of vectors and this can be seen as the SSSP, described earlier.

4.1.2 Bootstrapping

On a high level, bootstrapping is a procedure that given a ciphertext c with noise e, outputs a ciphertext c' with noise $e' \ll e$. This allows for more homomorphic operations at which some point will generate a new ciphertext with a level of noise that does not allow another homomoprhic encryption, without losing the ability to recover the plaintext. At this point, squashing and bootstrapping must be done again.

Given that the encryption of a plaintext m using a key pk is given by $Enc_{pk}(m)$ and that the decryption of a ciphertext c using a key sk is $Dec_{sk}(c)$, we have that the process of boostrapping starts with a ciphertext $c = Enc_{pk1}(m)$ and the encryption of the secret key $Enc_{pk2}(sk1)$, where we have two key pairs $(pk1, sk1)$ and $(pk2, sk2)$. Given that c has too much noise and is bootstrappable, after squashing, the procedure of bootstrapping it consists of homomorphically decrypting it: $c^* = Dec_{Enc_{pk2}(sk1)}(c)$, where c^* is equivalent to $Enc_{pk2}(m)$. Gentry shows that this procedure reduces the amount of noise in the resulting ciphertext [11], thus allowing for more homomorhpic operations.

4.2 DGHV Scheme

In 2010 Van Dijk, Gentry, Halevi and Vaikuntanathan proposed an SHE scheme that was based on the hardness assumption of the AGCD problem and was turned into an FHE by leveraging the bootstrapping technique, introduced a year earlier by Gentry. In the paper they initially proposed a symmetric encryption scheme and then evolved to an asymmetric encryption scheme. Here their symmetric proposal is described.

This scheme has all its parameters as polynomials in the security parameter λ. Namrely, η is the size in bits of the key p. They key generation outputs a key p, which is a large η-bit prime number. For encryption one must choose a random large prime q and a smal number $r \ll p$. Therefore, the message $m \in 0, 1$ is encrypted by doing:

Input: message m, key p, prime q, integer r
Output: ciphertext c

$$c = Enc(m) = m + 2r + pq \tag{10}$$

The decryption procedure is simple:

Input: ciphertext c, key p
Output: message m

$$m = Dec(c) = (c \mod p) \mod 2 \tag{11}$$

The homorphism over addition is shown below:

$$\begin{aligned} Dec(c_1 + c_2) = Dec(Enc(m_1) + Enc(m_2)) &= ((m_1 + m_2 + 2(r_1 + r_2) + p(q_1 + q_2)) \mod p) \mod 2 \\ &= (m_1 + m_2 + 2(r_1 + r_2)) \mod 2 \\ &= m_1 + m_2 \end{aligned} \tag{12}$$

The homorphism over multiplication is shown below:

$$\begin{aligned} Dec(c_1 * c_2) &= Dec(Enc(m_1) * Enc(m_2)) \\ &= ((m_1 m_2 + 2(m_1 r_2 + m_2 r_1 + 2 r_1 r_2) + q_1 q_2 p) \mod p) \mod 2 \end{aligned} \tag{13}$$

From this equation one can see that decryption is only achieveable if the absolute value of the noise $2(m_1 r_2 + m_2 r_1 + 2 r_1 r_2)$ is less than p [14].

4.3 BGV

The BGV encryption scheme proposed in 2011 by Brakerski, Gentry and Vaikuntanathan [21] is a fully homomorphic encryption scheme that works for both an LWE and an RLWE instance, but they achieve a better performance with the RLWE instance, therefore the focus here will be on the RLWE setup.

The way they found for keeping the ciphertext error within a given bound was to use the technique of modulus switching as introduced by Brakerski and Vaikuntanathan [22]. This modulo reduction maps a ciphertext $\vec{c}$ defined in a ring R_q, to a ring R_p, where $p < q$, which keeps the error e contained within the ciphertext at the same level. This enables one to multiply two ciphertexts and keep the error level constant [21]. By leveraging the bootstrapping technique, one can do this process indefinitely, which opens the way for an FHE scheme.

Their encryption scheme starts with a setup procedure in which one must choose an odd modulus q, μ-bit modulus q, set $N = \lceil 3 \log q \rceil$ and pick a distribution χ, which is the same as defined in the previous section for RLWE. The secret key sk is generated by drawing $\vec{s'} \leftarrow \chi$ and setting $sk = (1, \vec{s'}[1]) \in R_q^2$. The public key pk is obtained by generating a column matrix $A' \leftarrow R_q^{N \times 1}$ uniformly and a vector $e \leftarrow \chi^N$. Set $\vec{b} \leftarrow A's' + 2\vec{e}$. Then $pk = A$, where A is the 2-column matrix consisting of $\vec{b}$ followed by the the column matrix A.

The encryption of a message $m \in R_2$ is done by setting $\vec{m} \leftarrow (m, 0, ..., 0) \in R_q^{n+1}$, sampling $\vec{r} \leftarrow R_2^N$ and computing:
Input: message $\vec{m}$, public key A, modulus q
Output: ciphertext $\vec{c}$

$$\vec{c} \leftarrow \vec{m} + A^T \vec{r} \in R_q^2 \tag{14}$$

Decryption is done by:

Input: ciphertext $\vec{c}$, private key $\vec{s}$, modulus q
Output: message m

$$m \leftarrow [[\langle \vec{c}, \vec{s} \rangle]_q]_2 \tag{15}$$

where $\langle \vec{c}, \vec{s} \rangle$ is the dot product of $\vec{c}$ and $\vec{s}$ and $[\langle \vec{c}, \vec{s} \rangle]_q$ is the reduction of the dot product modulo q into the interval $(-q/2, q/2)$.

In order to show the homomorphic properties of this scheme, some operations must be defined, namely *BitDecomp*, *Powersof2*, *SwitchKeyGen*, *SwitchKey* and *Scale*.

Let $\vec{a} \in R_q^n$, then *Powersof2* is defined as:

$$Powersof2(\vec{a}) = (\vec{a}, 2 \cdot \vec{a}, ..., 2^l \cdot \vec{a}) \in R_q^{n \cdot l} \tag{16}$$

where its output is an N-dimensional vector, $l = \lfloor \log q \rfloor$ and $N = k \cdot l$.

BitDecomp is defined as:

$$BitDecomp(\vec{a}) = (a_{1,0}, ..., a_{1,l-1}, ..., a_{k,0}, ..., a_{k,l-1}) \tag{17}$$

where $(a_{1,0}, ..., a_{1,l-1}, ..., a_{k,0}, ..., a_{k,l-1})$ is an N-dimensional vector, $N = k \cdot l$ and $a_{i,j}$ is the j-th bit in a_i's binary representation, bits ordered from the least significant to the most significant.

The *SwitchKeyGen* operation enables one to: given an encryption c_1 of a message m under a secret key sk_1, obtain an encryption c_2 of the same message m, but under a secret key sk_2, with a not much higher noise. This operation starts by generating a public key A as defined above for another secret key $\vec{sk_2}$ and $N = n_1 \cdot \lceil \log q \rceil$, where n_1 is the dimension of the secret key sk_1. Then it outputs a matrix $B \leftarrow A + Powersof2(sk_1)$.

The *SwitchKey* takes the output B of *SwitchKeyGen* and outputs $\vec{c_2} = BitDecomp(\vec{c_1})^T \cdot B \in R_q^{n_2}$, where n_2 is the dimension of the vector sk_2.

The *Scale* operation outputs $\vec{x'}$ which is defined as the R-vector closest to $(p/q) \cdot x$ that sastifies $x' = x \mod r$, where $q > p$ and x are integers [21].

From the scheme and operations presented above it is now possible to define a partially homomorphic encryption scheme. The scheme that will be presented now can be transformed into a fully homomorphic encryption scheme by using Gentry's bootstrapping technique.

Initially one must choose the number of levels L and set $\mu' = \mu'(\lambda, L)$, where λ is the security parameter. Then the setup procedure defined earlier must be called with $\mu = (1 + j) \cdot \mu'$, where j goes from L to 0. This will give a decreasing sequence of moduli that will be used for modulus switching.

For generating the keys one must, for each j from L to 0 generate $\vec{s_j}$ by executing the secret key creation procedure and $\vec{A_j}$ by executing public key creation procedure. Then set $\vec{s'_j} = \vec{s_j} \otimes \vec{s_j}$, where $\otimes$ denotes the vector tensoring operator. Finally set $\vec{s''_j} = BitDecomp(s'_j)$ and run $B_j \leftarrow SwitchKeyGen(s''_j, s_{j-1})$. The secret sk consists of the s_j's and the public key pk consists of the A_j's and B_j's.

Encryption is done by executing the encryption procedure defined earlier using the public key A_j (B_j is not used here). Decryption is done in a similar fashion, but using the secret key s_j, which is the secret key in which the ciphertext is under.

Addition and multiplication are achieved by applying the respective operation on the ciphertexts and calling a procedure named *Refresh*, passing to it the resulting ciphertext and B_j. The *Refresh* procedure basically calls the *Scale* procedure, in order to switch moduli and then uses the *SwitchKey* procedure to switch the key under which the resulting ciphertext will be encrypted. By combining this with the bootstrapping technique one can achieve an FHE scheme.

4.4 BFV

In 2012 Fan and Vercauteren [19] ported the scheme proposed by Brakerski [20] from the LWE setting to the RLWE setting. As in [20] they make use of relinearization, but their version is more efficient. They also simplify the bootstrapping procedure, by using modulus switching. The scheme will be presented here, along with the relinearization technique and the modulus switching trick used during boostrapping.

For a B-bounded distribution χ over the ring R, where R is the ring used to define the RLWE problem, the secret key $sk = \boldsymbol{s}$ is sampled $\boldsymbol{s} \leftarrow \chi$. The public key is defined as:

$$pk = ([-(\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e})]_q, \boldsymbol{a}) \tag{18}$$

where $\boldsymbol{e} \leftarrow \chi$ and $\boldsymbol{a} \leftarrow R_q$, where R_q is the set of polynomials in $\mathbb{Z}_q$, where $\mathbb{Z}_q$ is the set of integers in the interval $(-q/2, q/2]$.

The message space is defined as being R_t for an integer $t > 1$. Therefore, to encrypt a message $\boldsymbol{m} \in R_t$, given that $\boldsymbol{p}_0 = pk[0], \boldsymbol{p}_1 = pk[1]$, sample $\boldsymbol{u}, \boldsymbol{e}_1, \boldsymbol{e}_2 \leftarrow \chi$ and compute:

Input: message $\boldsymbol{m}$, public key $\boldsymbol{p}_0, \boldsymbol{p}_1$, errors $\boldsymbol{u}, \boldsymbol{e}_1, \boldsymbol{e}_2$

Output: ciphertext $\boldsymbol{c}$

$$\boldsymbol{c} = ([\boldsymbol{p}_0 \cdot \boldsymbol{u} + \boldsymbol{e}_1 + \Delta \cdot \boldsymbol{m}]_q, [\boldsymbol{p}_1 \cdot \boldsymbol{u} + \boldsymbol{e}_2]) \tag{19}$$

where $\Delta = \lfloor q/t \rfloor$.
Decryption is done by:

Input: ciphertext $\boldsymbol{c}$, moduli $\boldsymbol{t}, \boldsymbol{q}$, private key $\boldsymbol{s}$

Output: message $\boldsymbol{m}$

$$\boldsymbol{m} = \left[\left\lfloor \frac{t \cdot [\boldsymbol{c}(s)]_q}{q} \right\rceil \right]_t = \left[\left\lfloor \frac{t \cdot [\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}]_q}{q} \right\rceil \right]_t \tag{20}$$

When showing that decryption works, a lemma on the bound of the error is stated in [19], where its norm is less than or equal to $2 \cdot \delta_R \cdot B^2 + B$. In the sequence they prove the lemma and show that since $\boldsymbol{m} \in R_t$, decryption works.

To show that this encryption scheme is homomorphic with respect to addition, a is defined ciphertext as a polynomial in $\boldsymbol{s}$ [19]:

$$[\boldsymbol{c}(\boldsymbol{s})]_q = [\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}]_q = \Delta \cdot \boldsymbol{m} + \boldsymbol{v} \tag{21}$$

Then the addition of two ciphertexts is shown to be:

$$[\boldsymbol{c}_1(\boldsymbol{s}) + \boldsymbol{c}_2(\boldsymbol{s})]_q = \Delta \cdot [\boldsymbol{m}_1 + \boldsymbol{m}_2]_t \tag{22}$$

By multplying two ciphertexts $ct_1(s)$ and $ct_2(s)$ one gets:

$$ct_1(\boldsymbol{s}) \cdot ct_2(\boldsymbol{s}) = \boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} + \boldsymbol{c}_2 \cdot \boldsymbol{s}^2 \tag{23}$$

As noted in [19], the problem here is that the resulting ciphertext has degree 2 and must be reduced to a degree 1 ciphertext, before any other homomorphic operation or decryption is done. In order to achieve this they introduced an operation called *relinearization*, which is similar to modulus switching [19]. This operation requires a relinearization key rlk, which, given the secret key sk and an integer p, is obtained by sampling $\boldsymbol{a} \leftarrow R_{p \cdot q}$ and $e \leftarrow \chi'$ $(\chi' != \chi)$ and computing:

$$rlk = ([-(\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}) + p \cdot \boldsymbol{s}^2]_{p \cdot q}, \boldsymbol{a}) \tag{24}$$

In order to apply relinearization to the product of two ciphertexts $ct_1 \cdot ct_2$, one must first compute:

$$\boldsymbol{c}_0 = \left[\left\lfloor \frac{t \cdot (ct_1[0] \cdot ct_2[0])}{q} \right\rceil\right]_q \tag{25}$$

$$\boldsymbol{c}_1 = \left[\left\lfloor \frac{t \cdot (ct_1[0] \cdot ct_2[1] + ct_1[1] \cdot ct_2[0])}{q} \right\rceil\right]_q \tag{26}$$

$$\boldsymbol{c}_2 = \left[\left\lfloor \frac{t \cdot (ct_1[1] \cdot ct_2[1])}{q} \right\rceil\right]_q \tag{27}$$

$$(\boldsymbol{c}_{2,0}, \boldsymbol{c}_{2,1}) = \left(\left[\left\lfloor \frac{\boldsymbol{c}_2 \cdot rlk[0]}{p} \right\rceil\right]_q, \left[\left\lfloor \frac{\boldsymbol{c}_2 \cdot rlk[1]}{p} \right\rceil\right]_q \right) \tag{28}$$

Then the output of relinearization is:

$$([\boldsymbol{c}_0 + \boldsymbol{c}_{2,0}]_q, [\boldsymbol{c}_1 + \boldsymbol{c}_{2,1}]_q) \tag{29}$$

In order to turn the encryption scheme into a fully homomorphic encryption scheme Gentry's bootstrapping technique without the squashing procedure was used in [19] . This was achieved by limiting the error $\boldsymbol{v}$ in $[\boldsymbol{c}(\boldsymbol{s})]_q = [\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot s]_q = \Delta \cdot \boldsymbol{m} + \boldsymbol{v}$ [19].

4.5 GSW

In 2013 Gentry, Sahai and Waters proposed a FHE scheme based on the LWE problem. More specifically, they built a scheme using a technique they called the *approximate eigenvector* method [17]. In order to define their scheme they introduced four key operations called *BitDecomp*, *BitDecomp*$^{-1}$, *Flatten* and *Powersof2*. *BitDecomp* and *Powersof2* were defined when presenting BGV, but here a slightly different definition for *Powersof2* is given:

Definition 5 *Let $\vec{a} \in \mathbb{Z}_q^n$ and $l = \log q + 1$, then:*

$$Powersof2(\vec{a}) = (a_1, 2a_1, 4a_1, ..., 2^{l-1}a_1, ..., a_k, 2a_k, ..., 2^{l-1}a_k) \tag{30}$$

4.5.1 Flattening

Let $\vec{a}$, $\vec{b}$ be vectors of some dimension k over $\mathbb{Z}_q$. Let $l = \lfloor \log_2 q \rfloor + 1$, that is, the number of bits required to represent q, plus one, therefore, l bits is enough to represent all the numbers in $\mathbb{Z}_q$. Let $N = k \cdot l$.

For $\vec{a}' = (a_{1,0}, ..., a_{1,l-1}, ..., a_{k,0}, ..., a_{k,l-1})$, $BitDecomp^{-1}$ is defined as:

$$BitDecomp^{-1}(\vec{a}') = (\sum 2^j \cdot a_{i,j}, ..., \sum 2^j \cdot a_{k,j}) \tag{31}$$

The $BitDecomp^{-1}$ is the inverse of *BitDecomp*, but with the exception of being well-defined even for when the input vector is not binary. *Flatten* is defined as:

$$Flatten(\vec{a}') = BitDecomp(BitDecomp^{-1}(\vec{a}')) \tag{32}$$

where the output is a N-dimensional vector with 0/1 coefficients.

For matrices the operations are applied in a row basis, thus, generating a new matrix.

4.5.2 Key Generation, Encryption and Decryption

Before diving into the key generation, encryption and decryption procedures, the parameters used in those procedures are defined. First a modulus q, lattice dimension n and an error distribution χ are chosen appropriately for LWE, such that 2^λ security is achieved against known attacks. Also set $m = O(n \log q)$, $l = \lfloor \log q \rfloor + 1$ and $N = (n+1) \cdot l$.

The key generation procedure outputs a secret key and a public key, where the secret key sk is generated by sampling $\vec{t} \leftarrow \mathbb{Z}_q^n$ and outputting $sk = \vec{s} \leftarrow (1, -t_1, ..., -t_n) \in \mathbb{Z}_q^{n+1}$. Set $\vec{v} = Powersof2(\vec{s})$. For the public key pk, generate $B \leftarrow \mathbb{Z}_q^m$ uniformly and a vector $\vec{e} \leftarrow \chi^m$. Set A to be the $(n+1)$-column matrix consisting of $\vec{b}$ followed by the n columns of B. Output $pk = A$, where $A \cdot \vec{s} = \vec{e}$.

To encrypt a message $\mu \in \mathbb{Z}_q$, sample an uniform matrix $R \in \{0,1\}^{N \times m}$ and do the following:
Input: message μ, public key A, uniform matrix R, N-dimensional identity matrix I_N
Output: ciphertext C

$$C = Flatten(\mu \cdot I_N + BitDecomp(R \cdot A)) \tag{33}$$

Decryption is done as follows [18]:
Input: ciphertext C, $\vec{v}$
Output: message μ

$$\begin{aligned} C \cdot \vec{v} &= \mu \cdot \vec{v} + BitDecomp(R \cdot A) \cdot \vec{v} \\ &= \mu \cdot \vec{v} + R \cdot A \cdot s \\ &= \mu \cdot \vec{v} + small \end{aligned} \tag{34}$$

where *small* is a small error.

Addition and multiplication are defined as the flattening of the resulting ciphertext:

$$C_1[\cdot]C_2 = Flatten(C_1[\cdot]C_2) \tag{35}$$

where $[\cdot]$ is either multiplication or addition.

Scheme	Public Key Size	Private Key Size	Ciphertext Size
Gentry	n^7	n^3	$n^{1.5}$
DGHV	$\tilde{\mathcal{O}}(\lambda^{10})$	$\tilde{\mathcal{O}}(\lambda^2)$	$\tilde{\mathcal{O}}(\lambda^5)$
BGV	$2dn\log q$	$2d\log q$	$2d\log q$
BFV	$2d\log q$	d	$2d\log q$
GSW	$\mathcal{O}(n^2\log^2 q)$	$(n+1)\log q$	$(n+1)^2\log^3 q$

Table 1: Public key, private key and ciphertext sizes for each FHE scheme presented.

4.6 Comparing the Schemes

Below the public key, private key and ciphertext sizes for each of the encryption schemes above are presented, where λ is the security parameter, n is the lattice dimension and d is a power of 2.

Getry's scheme public key, private key and ciphertext sizes are exponential on the underlying lattice dimension, which leads to the question: how big is n? According to Gentry himself $n \approx \lambda^2$ [38], which shows that the actual size of the public key can be on the magnitude of Megabytes to Gigabytes [12]. Although not very practical, this scheme was the first FHE scheme in history and also introduced a very important concept: *bootstrapping*.

The DGHV scheme basic idea had been explored before by two other encryption schemes, but the full potential of their homomorphic properties was never achieved [13]. In their work, the authors of the DGHV scheme not only realize the full potential of the homomorphic operations, but they also do an extensive security analysis of their scheme, showing that this kind of scheme, with the appropriate choice of security parameters, can be made secure against all known attacks [13]. Another contribution of their work is that they use addition and multiplication over the integers, instead of using ideal lattices over a polynomial ring, as did Gentry [11]. This gives their scheme a conceptual simplicity. Another advantage with respect to Gentry's scheme [11] is that the public key, private key and ciphertext sizes rely only on the security parameter λ. The drawback is that their scheme still uses the squashing and bootstrapping procedure, making it not so efficient.

The BGV scheme is very interesting due to the fact that it does not use bootstrapping for noise management, instead it uses, in a smarter way, the idea of modulus switching, which is cheaper computationally [21]. Another interesting feature is that its security is based on the hardness of RLWE for quasi-plynomial factors, which is a very good improvement security-wise, when compared to previous schemes, which relied on sub-exponential factors [21].

The BFV scheme leverages the relinearization procedure, but the way suggested in [19] is more efficient than what Brakerski initally proposed [20] and the the modulus switching technique is also used, but to simplify the bootstraping step. Also, their pulbic key, private key and ciphertext sizes are independent of the underlying lattice dimension.

The GSW scheme is very interesting due to the fact that eliminates a very expensive procedure: relinearization. By doing that, they not only made their scheme less expensive overall, but also eliminate a complex and hard to understand procedure. Another improvement was the elimination of vector tensoring, used in BFV for multiplying the ciphertexts, which increased a lot the ciphertext size and resulted in the need for relinearization. Instead of vector tensoring, they were able to make ciphertext multiplication as simple as matrix multiplication, which is cheaper than vector tensoring plus relinearization.

5 FHE Implementations

5.1 GH Implementation

In [32] Gentry and Halevi describe their implementation of Gentry's FHE scheme [11] and also present the results they obtained, in terms of performance. From table 1 it is possible to see that Gentry's scheme [11] presents a big overhead in terms of the sizes of its keys and ciphertexts. This is reflected on Gentry and Haveli's implementation [32], where, considering that the scheme is practically secure for a lattice with dimension $n = 2^{13}$ or $n = 2^{15}$ [32], the time to generate the keys and to recrypt the ciphertext were:

Dimension	KeyGen	Recrypt
2^{13}	8.4 min	2.8 min
2^{15}	2.2 hour	31 min

Table 2: Running time for key generation and recryption.

Although practically secure, the scheme is not practical for use in large scale.

5.2 HElib

HElib is library written in C++ by Halevi and Shoup [23], which implements the BGV encryption scheme [21], but with the optimizations proposed by Gentry-Halevi-Smart [29]. These optimizations are more focused on how to better handle packed ciphertexts.

A Benes network is a special kind of a shift network, that is cheaper and can be constructed efficiently from any permutation. This first leads to the definition of a shift network, which is a method to realize an arbitrary permutation [23].

More formally, a shift network is a d-column matrix with n rows, where d is called the depth of the network and each row is a vector sh_π and its entries $sh_\pi[i] = \pi(i) - i$. That is, the amount of positions one needs to move element i to put it in $\pi(i)$. π is a permutation $\pi : [n] \to [n]$, where $[n]$ is the set $\{0, ..., n-1\}$. The shift is done by:

$$w \leftarrow \textstyle\sum_{\delta \in sh_\pi} (m_\delta \times v) \gg \delta \tag{36}$$

where m_δ is an n-dimensional vector called a mask with entries equal to 1 where $sh_\pi[i] = \delta$ and 0 elsewhere. "$\gg \delta$" means the shift by δ positions and "$\times$" is the entry-wise multiplication.

A Benes network is a permutation π on $[n]$, where $n = 2^r$, for a positive integer r. This network has depth $2r - 1$, where every level in the network has a cost at most 2 [23]. This network has depth $2r - 1 = O(\log n)$ and cost at most $4r - 2 = O(\log n)$, which is a very efficient bound.

5.2.1 HEtest

HEtest is a homomorphic encryption testing framework which was used in [39] to measure the performance of [23]. The results achieved in [39] are for a security parameter $\lambda = 80$ and for the following parameters:

l	d	w
378	{6, 7}	{4, 10, 20, 50, 100, 1000}
630	12	{4, 10, 20, 50, 100, 200}
600	18	{4, 10, 20, 50, 100}
682	{21, 24}	{4, 10, 20, 50, 100}

Table 3: Parameters used for testing based on batch width l, circuit depth d, and maximum number of inputs w.

The full results are displayed and explained in [39], but average times for key generation, encryption and decryption are shown below:

Key Generation	Encryption	Decryption
$2.02 \cdot 10^8$	0.033	0.116

Table 4: Average time, in seconds, for key generation, encryption and decryption.

All the tests were run on a machine with two Intel Xeon X5650 processors, 96 GB of RAM and a 64-bit Ubuntu 12.04 LTS Linux distribution.

5.3 Simple Encryption Arithmetic Library (SEAL)

The Simple Encryption Arithmetic Library (SEAL) is a homomorphic encryption library developed by researches in the Cryptography Research group at Microsoft Research [24]. This library implements the BFV [19] and the CKKS [26] encyption schemes. The focus here will be on their implementation of the BFV encryption scheme.

The way SEAL implements the BFV encryption scheme has fundamental differences when compared to the *textbook* version of BFV. The only property that remains the same is the message space, which are still polynomials in R_t, but ciphertexts are no longer strictly in $R_q \times R_q$, they are defined as tuples of polynomials in R_q of length at least 2 [24]. This is a fundamental difference and its result is that their implementation allows for ciphertexts of arbitrary size, with the drawback of losing the compactness property of homomorphic encryption [25]. Addition is done very similar to the way it was defined in the original paper of BFV, the main difference is that it supports arbitrary size ciphertexts. Multiplication follows the same logic, but since it supports arbitrary size ciphertexts, there is no need to use *relinearization* before decrypting the resulting ciphertext. Lastly, the secret key is not drawn from R_2, but from R_3, that is, its coefficients are not in $\{0, 1\}$, but in $\{-1, 0, 1\}$.

6 Applications of Homomorphic Encryption

Here some applications of homomorphic encryption schemes are presented, with the goal of showing how FHE schemes enable regular data processing applications to compute on encrypted data, preserving the privacy of the data. Specifically for machine learning algorithms, examples are presented that not only preserve the privacy of the data, but also the privacy of the model, since the model itself can leak information about the data it was trained on [34].

6.1 ML Confidential

Some applications of machine learning algorithms on homomorphically encrypted data have been proposed and even implemented, for example [27], which proposes a protocol called *ML Confidential*. In [27] the BFV encryption scheme [19] was leveraged, but their implementation resembles the one by SEAL, in the sense that their implementation supports abitrary size ciphertexts, eliminating the need for *relinearization* [27].

In [27] two machine learning algorithms are implemented: Linear Means (LM) and Fisher's Linear Discriminant Classifier. The Linear Means classifier determines a vector of weights $\vec{w}$ and a class c, such that $f(\vec{x}; \vec{w}, c) = 0$ defines a hyper-plane midway on and orthogonal to the line through the two class-conditional means [27]. Fisher's Linear Discriminant Classifier is very similar to LM, but it takes into account the class-conditional covariances, by aiming at finding a projection that maximizes the separation between classes [27].

The performace of the algorithms on training both on encrypted and unencrypted were compared. For both the encrypted version perfomed slower, having had a slow-down of 6 to 7 orders of magnitude for both classifiers. In [27] it is noted that such result was achieved without doing any significant optimizations when implementing the BFV homomoprhic encryption scheme, nor when implementing the algebraic operations using the Magma package [28].

6.2 Homomorphic Computation of Edit Distance

In [30] Cheon, Kim and Lauter implement the Edit Distance algorithm [31] in the context of genomic sequence analysis, leveraging homomorphic encryption, in order to preserve the privacy of the patients data. The homomorphic encryption scheme they use is BGV [21], specifically the implementation in HElib [23].

The challenge in adapting the Edit Distance algorithm to a HE setting is describing the algorithm as a combination of circuits, since homomorphic operations are defined as circuits. In [**?**]the algorithm for a classic setting is presented and then circuits are developed to adapt it to the HE setting. Here the original algorithm and the circuits are presented, but all the reasoning behind the circuits is left out, but the reader can check it on [30].

The Edit Distance algorithm takes as input two strings $\alpha = \alpha_1, ..., \alpha_n$ and $\beta = \beta_1, ..., \beta_m$ of lengths n and m, respectively. The output is the edit distance.

Algorithm 1: Edit Distance Algorithm

```
Input: Strings $\alpha$ and $\beta$
Output: $D_{n,m}$ which is the edit distance
function calculateEditDistance($\alpha, \beta, n, m$)
    for $i \leftarrow 0$ to $n$ do
        $D_{i,0} \leftarrow i$
    for $j \leftarrow 0$ to $m$ do
        $D_{0,j} \leftarrow j$
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $m$ do
            $t \leftarrow (\alpha_i = \beta_j)$ ? 0 : 1
            $D_{i,j} \leftarrow min\{D_{i-1,j-1} + t, D_{i,j-1} + 1, D_{i-1,j} + 1\}$
    return $D_{n,m}$
```

The algorithm presented above has three mathematical operations that have to be done homomorphically, therefore, they must be represented as circuits, namely: equality (line 8), comparison (line 9) and addition (line 9). Equality can be defined by a circuit that takes as input the binary representation of two strings x and y and outputs 1 if they are equal and 0 otherwise:

$$Equality(x, y) = \wedge_{i=1}^{\mu}(1 \oplus x_i \oplus y_i) \tag{37}$$

Comparison:

$$c_i = ((x_i \oplus 1) \wedge y_i) \oplus ((x_i \oplus 1 \oplus y_i) \wedge c_{i-1}) \tag{38}$$

for $i \geq 2$ and $c_1 = (x_1 \oplus 1) \wedge y_1$.

For addition it is assumed that for two μ-bits numbers x, y, their sum is less than 2^{μ}. Addition is then defined as $(s_1, ..., s_{\mu})$ where:

$$s_i = \begin{cases} \mathrm{x}_1 \oplus y_1 & \text{if i = 1} \\ \mathrm{x}_i \oplus y_i \oplus e_{i-1} & \text{otherwise} \end{cases}$$

$$e_i = \begin{cases} \mathrm{x}_1 \wedge y_1 & \text{if i = 1} \\ (\mathrm{x}_i \wedge y_i) \oplus ((x_i \oplus y_i) \wedge e_{i-1}) & \text{otherwise} \end{cases}$$

In [30] the algorithm was homomorphically implemented using the circuits defined above and tested for sequences of size up to 8. They did not test it for larger (more realistic) sequences sizes due to the large use of memory of their implementation. For the largest sequence they tested, $n = m = 8$, the key generation was 27.5454 seconds and the encryption step was 16.4524 seconds [30]. The results were achieved on a machine with Intel Xeon i7 2.3GHz processor and 192GB of RAM, for 84 bit security.

6.3 Private Machine Learning Classification

In [33] Sun, Zhang, Liu, Yu and Xie present a way to machine learning classification on encrypted data leveraging fully homomorphic encryption. The FHE scheme they implement is a variant of HElib's [23] BGV scheme implementation and the classifiers they implement are: hyperplane decision-based classification, Naïve Bayes and decision tree. Here the differences between their FHE scheme and HElib's implementation are presented, then the 3 classifiers are described and finally the results achieved in [33] is discussed.

Differently than HElib's implemetation of BGV, where relinearization is done after modulus swicthing, which in turn is executed after each homomorphic multiplication, in [33]'s scheme first relinearization is done, in order to reduce the size of the ciphertext after multiplication [33]. Then modulus switching is done with the goal of reducing the ciphertext's modulus and noise. They further add that no relinearization or modulus switching is needed if the next operation is addition or if there is no homomorphic multiplication for the current ciphertext obtained from a homomorphic multiplication [33].

Hyperplane decision-based classification is defined as a model w that includes k weight vectors $(w_i)_{i=0,1,...,k-1}$, where $w_i \in \mathbb{R}^n$ and classification is done by:

$$class = argmax_{i\in[0,k)}\langle w_i, x\rangle \tag{39}$$

where $argmax$ outputs the index i that maximizes $\langle w_i, x\rangle$ and x is the feature vector.

Naïve Bayes classification chooses the class by maximum posteriori probability:

$$class = argmax_{i\in[0,k)} p(C = c_i|X = x) \tag{40}$$

where $p(C = c_i|X = x)$ is the probability that the class is c_i, given that the feature vector is x.

Decision tree classification is better explained using Figure 2, which illustrates a tree where the leaf nodes are the classes $(c_j)_{j=0,1,\ldots,k-1}$ and each interior node represents a division rule. Each division rule is basically a weight $(w_i)_{i=0,1,\ldots,n-1}$ which will be used for comparison: $x \geq w_i$. If $x \geq w_i$, then the right node is chosen and if $x < w_i$, then the left node is chosen. This comparison process goes on until a leaf node is reached and thus the class is defined.

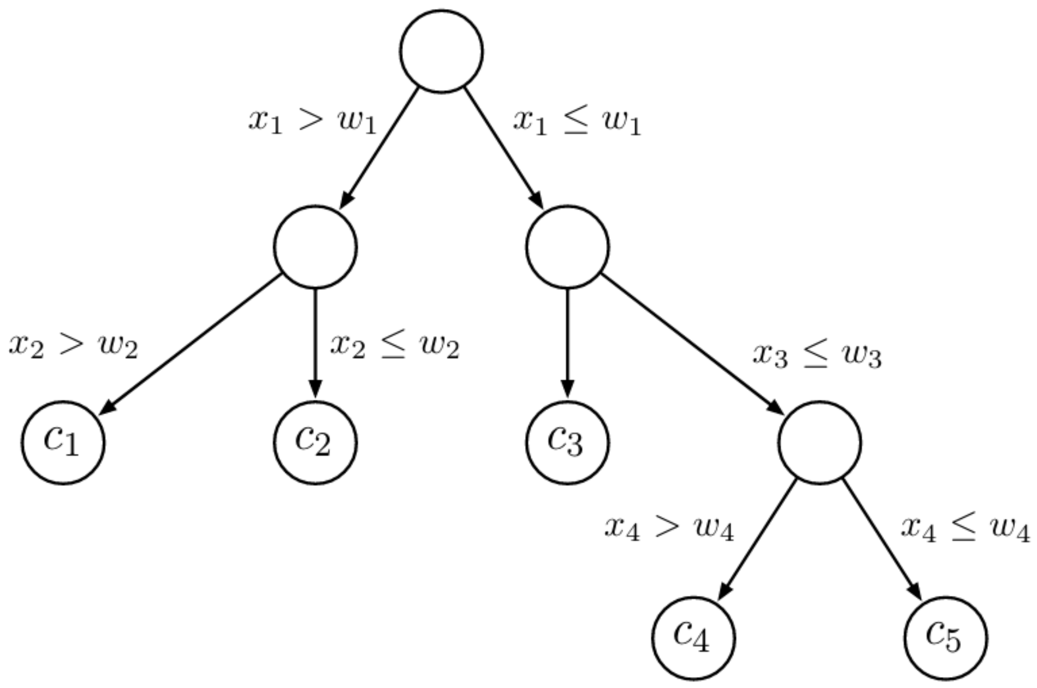

Figure 2: Visualization of a decision tree [34].

In order to run the classifiers above homomorphically, [33] defines how to homomorphically compute two operations, that are essential to the classifiers: comparison and $argmax$.

Comparison is done by first homomorphically computing the comparison ciphertext as below:

$$c_b = c_t + c_0 - c_1 \tag{41}$$

where c_0, c_1 are the encryptions of m_0, m_1, the plain texts, and addition and subtraction are done homomorphically.

Then the ciphertext c_b is returned to the user (who is the owner of the data and the private key), which decrypts the ciphertext into b and obtains the result of the comparison by checking b's l-th bit, where $l = \log_2 t + 1$. If $b_l = 1$, then $m_0 \geq m_1$ and $m_0 < m_1$, otherwise.

Their *argmax* operation is done by using the comparison operation defined above. Initially set max to zero and then, for each encryption c_i of each number, where $i = 0, ...k - 1$, execute the comparison operation between c_i and $c_m ax$, where $c_m ax$ is the encryption of the current maximum. After each comparison, the result c_b is sent to the user, which decrypts it to b and returns to the server b_l. If $b_l = 1$, then set $max = i$.

Now machine learning classification can be done on the encryption c_x of the feature vector x and on the encryption c_w of the model w.

Below some results are presented, starting with a comparison of the execution time for the [33] implementation of BGV and HElib's implementation (both with Single Instruction Multiple Data (SIMD)) for different security parameters and for 10 homomorphic operations, including a variable multiplicative depth, which is specified in each table. The results are in microseconds:

λ	5	50	53	55
[33] implementation	0.54	0.77	0.78	0.84
HElib	31.33	40.28	89.89	98.77

Table 5: Multiplicative depth 1.

λ	5	50	53	55
[33] implementation	7.72	10.73	21.63	23.41
HElib	77.75	84.20	170.56	214.66

Table 6: Multiplicative depth 2.

λ	5	50	53	55
[33] implementation	48.30	55.58	116.84	132.62
HElib	111.88	123.59	253.54	316.24

Table 7: Multiplicative depth 3.

From the tables it is possible to see that the BGV implementation of [33] is much faster than HElib's, executing from 10 to 100 times faster, depending on the multiplicative depth.

In [33] a comparison of the execution time of its implementation of BGV and HElib's implementation for the Decision Tree classifier only (both with SIMD) was provided:

λ	5	50	53	55
[33] implementation	61.39	74.85	161.26	177.06
HElib	209.36	244.53	517.88	1788.60

Table 8: Decition Tree classifier running time.

All tests were run on a virtual machine on top of Windows 7, running Ubuntu 12.04. The machine physical hardware was: two Intel (R) Core (TM) i5-3470 CPU processors, running at 3.20

GHz, with 8.00 GB RAM. And the virtual hardware was: single Intel (R) Core (TM) i5-3470 CPU processor, with 3.70 GB RAM.

7 Conclusion

The main takeaway of this report is that the current state of homomorphic encryption does not allow for very efficient and highly performant algortihms that operate on encrypted data. Although many efforts have been made in optimizing the implementation of fully homomorphic encryption schemes [23] [24] [12], they are still not enough to enable large scale applications. On the other hand, it has been proven that algorithms that people interact with on a daily basis can be adapted to operate on encrypted data, which opens the way for applications that are more privacy preserving, thus, making users safer online.

One very interesting aspect of fully homomorphic encryption schemes to note is that what makes the schemes secure, from a mathematical standpoint, is the exact same thing that makes it harder to build practical FHE schemes. Namely: the small error used in lattices problems like (R)LWE. This basic property of the problems underpinning FHE schemes is what made it necessary for the cryptography community to come up with operations like bootstrapping, squashing, modulus switching and relinearization, to name a few, since the error growth on homomorphic operations is what compromises decryption.

The mathematical problems powering FHE are already very mature and have not been proven to be insecure, yet. Also, the implementations of the encryption schemes have introduced many important optimizations, but they are still not enough to enable practicality.

References

[1] Peikert, C. (2016). A decade of lattice cryptography. Foundations and Trends® in Theoretical Computer Science, 10(4), 283-424.

[2] Hoffstein, J., Pipher, J. C., Silverman, J. H., & Silverman, J. H. (2014). An introduction to mathematical cryptography (Vol. 2). New York: springer.

[3] Regev, O. (2004). Lattices in Computer Science, Lecture 1. Tel Aviv University.

[4] Boneh, D., Gentry, C., Halevi, S., Wang, F., & Wu, D. J. (2013, June). Private database queries using somewhat homomorphic encryption. In International Conference on Applied Cryptography and Network Security (pp. 102-118). Springer, Berlin, Heidelberg.

[5] Naehrig, M., Lauter, K., & Vaikuntanathan, V. (2011, October). Can homomorphic encryption be practical?. In Proceedings of the 3rd ACM workshop on Cloud computing security workshop (pp. 113-124). ACM.

[6] Li, T., Huang, Z., Li, P., Liu, Z., & Jia, C. (2018). Outsourced privacy-preserving classification service over encrypted data. Journal of Network and Computer Applications, 106, 100-110.

[7] Martins, P., Sousa, L., & Mariano, A. (2017). A survey on fully homomorphic encryption: An engineering perspective. ACM Computing Surveys (CSUR) , 50(6), 83.

[8] Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.

[9] Paillier, P. (1999, May). Public-key cryptosystems based on composite degree residuosity classes. In International Conference on the Theory and Applications of Cryptographic Techniques (pp. 223-238). Springer, Berlin, Heidelberg.

[10] ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory, 31(4), 469-472.

[11] Gentry, C., & Boneh, D. (2009). A fully homomorphic encryption cheme (Vol. 20, No. 09). Stanford: Stanford University.

[12] Gentry, C., & Halevi, S. (2011, May). Implementing gentry's fully-homomorphic encryption scheme. In Annual international conference on the theory and applications of cryptographic techniques (pp. 129-148). Springer, Berlin, Heidelberg.

[13] Van Dijk, M., Gentry, C., Halevi, S., & Vaikuntanathan, V. (2010, May). Fully homomorphic encryption over the integers. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 24-43). Springer, Berlin, Heidelberg.

[14] Acar, A., Aksu, H., Uluagac, A. S., & Conti, M. (2018). A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys (CSUR), 51(4), 79.

[15] Lyubashevsky, V. (2013). Vadim Lyubashevsky on Ring-LWE. https://www.youtube.com/watch?v=okJwRM0Yu7E

[16] Regev, O. (2010). The learning with errors problem. Invited survey in CCC, 7.

[17] Gentry, C., Sahai, A., & Waters, B. (2013). Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Advances in Cryptology–CRYPTO 2013 (pp. 75-92). Springer, Berlin, Heidelberg.

[18] Gentry, C. (2013). Homomorphic Encryption from Learning with Errors: Conceptually Simpler, Asymptotically-Faster, Attribute-Based. CRYPTO Santa Barbara 2013. https://www.youtube.com/watch?v=uabmoq4-tGQ

[19] Fan, J., & Vercauteren, F. (2012). Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive, 2012, 144.

[20] Brakerski, Z. (2012). Fully homomorphic encryption without modulus switching from classical GapSVP. In Advances in cryptology–crypto 2012 (pp. 868-886). Springer, Berlin, Heidelberg.

[21] Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2014). (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT), 6(3), 13.

[22] Brakerski, Z., Vaikuntanathan, V. Efficient fully homomorphic encryption from (standard) lwe. Manuscript, to appear in FOCS 2011, available at http://eprint.iacr.org/2011/344.

[23] Halevi, S., & Shoup, V. (2014, August). Algorithms in helib. In International Cryptology Conference (pp. 554-571). Springer, Berlin, Heidelberg.

[24] Simple Encrypted Arithmetic Library (release 3.1.0). Microsoft Research, Redmond, WA. https://github.com/Microsoft/SEAL

[25] Armknecht, F., Boyd, C., Carr, C., Gjøsteen, K., Jäschke, A., Reuter, C. A., & Strand, M. (2015). A Guide to Fully Homomorphic Encryption. IACR Cryptology ePrint Archive, 2015, 1192.

[26] Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017, December). Homomorphic encryption for arithmetic of approximate numbers. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 409-437). Springer, Cham.

[27] Graepel, T., Lauter, K., & Naehrig, M. (2012, November). ML confidential: Machine learning on encrypted data. In International Conference on Information Security and Cryptology (pp. 1-21). Springer, Berlin, Heidelberg.

[28] Bosma, W., Cannon, J., & Playoust, C.The Magma algebra system. I. The user language. J. Symbolic Comput., 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[29] Gentry, C., Halevi, S., & Smart, N. P. (2012, April). Fully homomorphic encryption with polylog overhead. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 465-482). Springer, Berlin, Heidelberg.

[30] Cheon, J. H., Kim, M., & Lauter, K. (2015, January). Homomorphic computation of edit distance. In International Conference on Financial Cryptography and Data Security (pp. 194-212). Springer, Berlin, Heidelberg.

[31] Wagner, R. A., & Fischer, M. J. (1974). The string-to-string correction problem. Journal of the ACM (JACM), 21(1), 168-173.

[32] Gentry, C., & Halevi, S. (2011, May). Implementing gentry's fully-homomorphic encryption scheme. In Annual international conference on the theory and applications of cryptographic techniques (pp. 129-148). Springer, Berlin, Heidelberg.

[33] Sun, X., Zhang, P., Liu, J. K., Yu, J., Xie, W. (2018). Private machine learning classification based on fully homomorphic encryption. IEEE Transactions on Emerging Topics in Computing.

[34] Bost, R., Popa, R. A., Tu, S., Goldwasser, S. (2015, February). Machine learning classification over encrypted data. In NDSS.

[35] Rivest, R. L., Adleman, L., Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. Foundations of secure computation, 4(11), 169-180.

[36] Brickell, E. F., Yacobi, Y. (1987, April). On privacy homomorphisms. In Workshop on the Theory and Application of of Cryptographic Techniques (pp. 117-125). Springer, Berlin, Heidelberg.

[37] Homomorphic Encryption. Brilliant.org. https://brilliant.org/wiki/homomorphic-encryption/.

[38] Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. Proceedings of the 41st annual ACM symposium on Symposium on theory of computing-STOC'09. Vol. 9.

[39] Varia, M., Yakoubov, S., Yang, Y. (2015, January). HEtest: A homomorphic encryption testing framework. In International Conference on Financial Cryptography and Data Security (pp. 213-230). Springer, Berlin, Heidelberg.

[40] Gentry, C., Peikert, C., Vaikuntanathan, V. (2008, May). Trapdoors for hard lattices and new cryptographic constructions. In Proceedings of the fortieth annual ACM symposium on Theory of computing (pp. 197-206). ACM.