WTFHE: neural-netWork-ready Torus Fully Homomorphic Encryption

Jakub Klemsa[✉]
Czech Technical University in Prague
Prague, Czech Republic
jakub.klemsa@fel.cvut.cz

Martin Novotný
Czech Technical University in Prague
Prague, Czech Republic
novotnym@fit.cvut.cz

***Abstract*—We are currently witnessing two arising trends, which have a huge potential to threaten our privacy: the invasive sensors of the Internet of Things (IoT), and the powerful data mining techniques, in particular we focus on Neural Networks (NN's). For this reason, powerful countermeasures must be called for service: namely end-to-end encryption. Such an approach however requires an encryption scheme that enables processing of the encrypted data – this is known as the *Fully Homomorphic Encryption* (FHE).**

In this paper, we revisit an FHE scheme named TFHE, which is suitable for evaluation of NN's over encrypted input data, and we suggest to incorporate a verifiability feature to the evaluation process. Since there already exist other variants of the original TFHE scheme—currently only implemented in C++, which is rigid—we further introduce a library for rapid prototyping of new concepts related to TFHE. Our library is implemented in Ruby, which is an interpreted language and which goes with an interactive shell. Hence any new method can be speedily verified before implemented as a high-performance library.

***Index Terms*—fully homomorphic encryption, neural networks, internet of things, rapid prototyping**

I. Introduction

Privacy is a valuable thing, however, much more than ever before, it is being challenged – in particular with the rapid growth of the Internet of Things (IoT). Firstly, more and more IoT sensors are approaching our vicinity, hence sneakily violating our privacy. Secondly, current powerful data mining techniques can exploit literally any available piece of information, which means that even *seemingly innocent data* can lead to *serious privacy breaches*.

This work was supported by the Grant Agency of CTU in Prague, grant No. SGS19/109/OHK3/2T/13.

Limiting the information, which is allowed to be collected by an IoT device, is somewhat tricky: you need to find an—often impossible—balance between privacy protection and data usability. Hence an ultimately better approach is to have the information encrypted all the time it has been outside the device (aka *end-to-end encryption*). Now the tricky thing becomes *processing* of the encrypted data. This idea was originally suggested back in 1970's by Rivest et al. [1] and posed a major cryptographic challenge (aka the *Holy Grail of Cryptography*) until Gentry's breakthrough in 2009 [2].

The approach, which allows for operations over encrypted data, is referred to as the *Fully Homomorphic Encryption* (FHE) and it enables to evaluate any function (represented by a Boolean circuit) over encrypted input data. Since the Gentry's paper, there have emerged several improvements and modifications, including but not limited to theoretical advances [3], [4] or implementations [5], [6].

On the other hand, in the field of data processing, Neural Networks (NN's) are experiencing its renaissance. In addition, NN's can be handled by certain FHE schemes advantageously as Bourse et al. [7] outlined. They use an adaptation of a scheme known as TFHE: Fully Homomorphic Encryption over the Torus introduced by Chillotti et al. in 2016 [8], recently extended in [9]. Not only is TFHE currently one of the most promising FHE schemes in general, its internal structure in addition allows for evaluation of arbitrary function during the so called *bootstrapping phase*. Note that this property is a key to the evaluation of a NN.

Our Contribution

First, we summarize the prerequisities for prospective evaluation of NN's over encrypted data and we revisit the latest approaches. Next, we outline how common functions can be handled to deal with current limitations and exploit the maximum of available resources. We further suggest to incorporate a verifiability feature to the evaluation process, for which we express the need for a library with sufficient space for custom adjustments, improvements or modifications. Finally, we introduce such a library – an implementation of WTFHE in Ruby. The library aims to serve particularly for rapid prototyping of any future proposals as well as TFHE modifications, including but not limited to NN evaluation.

Paper Outline

In Section II, we define symbols and notation used in this paper and we provide a brief overview of TFHE as well as NN's. Next, we discuss NN evaluation over encrypted data and its prospective verifiability in Section III. In Section IV, we introduce our WTFHE Ruby library. We conclude this paper and outline future directions in Section V.

II. Preliminaries

A. Symbols and Notation

In this paper, we will use the following symbols.

- $\mathbb{B}$ the set $\{0, 1\}$,
- $\mathbb{T}$ the real torus $\mathbb{R}/\mathbb{Z}$, i.e., the fractional part of real numbers,
- $\mathbb{M}^{(N)}[X]$ the set of polynomials over an abelian group $\mathbb{M}$ modulo $X^N + 1$, i.e., $\mathbb{M}[X]/(X^N + 1)$,
- $\mathbb{M}^n$ the set of n-dimensional vectors of elements from $\mathbb{M}$,
- $\mathbb{M}^{n,m}$ the set of $n \times m$-dimensional matrices of elements from $\mathbb{M}$.

Note II.1. When multiplying an element of $\mathbb{M}^{(N)}[X]$ by X, coefficients rotate, while changing their sign over the "edge". Indeed, for $a \in \mathbb{M}$, $aX^{N-1} \cdot X \equiv -a \pmod{X^N + 1}$. We call this the *negacyclic property*.

B. TFHE

In this section, we briefly recall a FHE cryptosystem by Chillotti et al. [9] named TFHE, in its canonical version. TFHE builds upon Learning With Errors by Regev [10] (LWE) and upon a substantial improvement of the bootstrapping procedure by Ducas et al. [11]. TFHE further follows generalizations of LWE [4] and Gentry-Sahai-Waters cryptosystem [3] (GSW) in the bootstrapping procedure, denoted as T(R)LWE and T(R)GSW, respectively.

To estimate the bit-security of an LWE-based scheme, Albrecht et al. [12] summarized known results and provided an interactive estimator[1].

1) TRLWE: In the context of TRLWE, ciphertexts are referred to as *samples*, a definition follows.

Definition 1 (Canonical TRLWE Sample [9]; simplified)**.** Let $n \geq 1$ and N, a power of 2, be two integers, $\alpha \geq 0$ a real noise parameter and $\mathbf{k} \in \mathbb{B}^{(N)}[X]^n$ a vector of binary polynomials modulo $X^N + 1$, which serves as the secret key. A fresh TRLWE sample of a message $\mu \in \mathbb{T}^{(N)}[X]$ is a vector $(\mathbf{a}, b) \in \mathbb{T}^{(N)}[X]^{n+1}$ where b is picked from a Gaussian distribution around $\mu + \mathbf{k} \cdot \mathbf{a}$ with scale α and $\mathbf{a}$ is either: uniformly random in $\mathbb{T}^{(N)}[X]^n$ (*random* sample), or equal to $\mathbf{0}$ (*trivial* sample). The sample is *noiseless* if $\alpha = 0$, and *homogeneous* if $\mu = 0$.

The opposite (viewed as decryption) is referred to as *phase* function, a definition follows.

Definition 2 (Canonical TRLWE Phase [9]; simplified)**.** Let $\mathbf{c} = (\mathbf{a}, b) \in \mathbb{T}^{(N)}[X]^{n+1}$ be a canonical TRLWE sample and let $\mathbf{k} \in \mathbb{B}^{(N)}[X]^n$ be a TRLWE key. We call $\varphi_{\mathbf{k}}(\mathbf{c}) := b - \mathbf{k} \cdot \mathbf{a}$ the *phase* of the sample $\mathbf{c}$. Let further $\operatorname{msg}(\mathbf{c})$ denote the expectation of $\varphi_{\mathbf{k}}(\mathbf{c})$.

These definitions unify (T)LWE and RingLWE [13]. Indeed, for $n = 1$ and N large we have RingLWE, for $N = 1$ and n large we have LWE (or TLWE for its binary version).

In the following informal lemma, we outline the additive homomorphic property of TRLWE.

[1]https://estimate-all-the-lwe-ntru-schemes.github.io/docs/

Lemma II.1 (Additive Homomorphism; informal)**.** *Let $k \in \mathbb{N}$, $\mathbf{c}_1, \dots, \mathbf{c}_k$ TRLWE samples under the same key $\mathbf{k}$ and $e_1, \dots, e_k \in \mathbb{Z}^{(N)}[X]$. Let $\mathbf{c} = \sum_{i=1}^{k} e_i \cdot \mathbf{c}_i$. If the noise is "not too large", then*

$$\mathrm{msg}(\mathbf{c}) = \sum_{i=1}^{k} e_i \cdot \mathrm{msg}(\mathbf{c}_i). \tag{1}$$

Note that multiplicative homomorphism is achieved by a combination with TRGSW samples, which is beyond the scope of this paper.

2) TFHE Bootstrapping: First, we provide a (version of) bootstrapping algorithm from [9] extended by a suggestion by Carpov et al. [14], which allows to evaluate arbitrary function during the bootstrapping procedure (namely we follow the $\mathsf{TV}_F(X) \cdot X^m$ option as per [14]).

Algorithm 1 TFHE Bootstrapping

Input: A TLWE sample $(\mathbf{a}, b)$ of $\mu = {}^m\!/_{2N}$, $m \in \mathbb{Z}_{2N}$, under key $\mathbf{k}$, and TRGSW samples $\mathsf{BK}_{\mathbf{k}\to\mathbf{k}'}$ of bits of $\mathbf{k}$ under key $\mathbf{k}'$ (aka bootstrapping keys).
Output: A TLWE sample of $f(m)$ under key $\mathbf{k}'$, where $f\colon \mathbb{Z}_{2N} \to \mathbb{T}$, $f(-N+k) = -f(k)$, is encoded within a *test vector* $\mathsf{TV}_f \in \mathbb{T}^{(N)}[X]$.

```
1: $\bar{a}_i \leftarrow \lfloor 2Na_i \rceil$ for $i \in [1, \dim(\mathbf{k})]$, $\bar{b} \leftarrow \lfloor 2Nb \rceil$
2: $\mathsf{ACC} \leftarrow (\mathbf{0}, X^{-\bar{b}} \cdot \mathsf{TV}_f) \in \mathbb{T}^{(N)}[X]^{n+1}$
3: for $i \in [1, \dim(\mathbf{k})]$ do
4:     $\mathsf{ACC} \leftarrow \mathsf{BK}_i \boxdot (X^{\bar{a}_i} \cdot \mathsf{ACC} - \mathsf{ACC}) + \mathsf{ACC}$
5: return $\mathrm{SampleExtract}(\mathsf{ACC})$
```

Line 1 rescales and rounds torus values to integers $\bmod 2N$. Line 2 initializes an *accumulator* ACC with a trivial sample of $X^{-\bar{b}} \cdot \mathsf{TV}_f$, where multiplication by $X^{-\bar{b}}$ performs a negacyclic rotation (cf. Note II.1) of TV_f. Line 4 further nega-rotates TV_f by $k_i\bar{a}_i$, where k_i is TRGSW-encrypted inside BK_i and $\boxdot$: $\mathrm{TRGSW} \times \mathrm{TRLWE} \to \mathrm{TRLWE}$ is an operation referred to as the *external product*. Note that lines 2–4 are also known as $\mathrm{BlindRotate}$, since the rotation is hidden due to the encrypted k_i's. Finally, line 5 extracts and returns a TLWE sample from ACC, which is a TRLWE sample.

Note that during bootstrapping, the decryption procedure with encrypted key is performed internally, hence the result remains encrypted. Indeed, the test vector gets (blindly) rotated by $X^{\sum_i k_i\bar{a}_i - \bar{b}} = X^{-\varphi_{\mathbf{k}}(\bar{\mathbf{a}}, \bar{b})}$. As a result, the desired value appears at the constant term, which is finally extracted.

C. Neural Networks

For a very brief insight to how neural networks are evaluated, we consider the simplest scenario. An *artificial neural network* is a series of elementary building blocks referred to as *perceptrons* organized in *layers*, which are evaluated one after each other. A perceptron P inputs k_P values from perceptrons in the preceding layer (or NN inputs itself) and outputs single value, possibly to several perceptrons in the subsequent layer as their respective input (or NN output itself). A *weight* $w_i^{(P)}$ is assigned to each (i-th) input of the perceptron P. These weights, together with the structure, define the neural network. The perceptron P evaluates the input values v_i as follows:

$$\mathrm{eval}_P(v_i)_{i=1}^{k_P} = f\Big(\sum_{i=1}^{k_P} w_i^{(P)} v_i\Big), \tag{2}$$

where f is called the *activation function*; cf. Figure 1. f is a non-linear and usually also an odd function with bounded image, e.g., $\tanh$ or sgn.

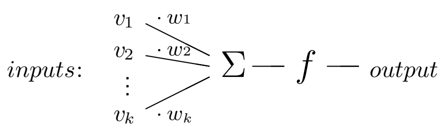

Fig. 1. Perceptron evaluation.

III. WTFHE

In order to evaluate a NN on encrypted data, two homomorphic operations must be supported, cf. (2):

1) *homomorphic addition* (which clearly enables scalar multiplication by a known integer), and
2) *homomorphic evaluation* of the (non-linear) activation function.

A. Addition

Addition, including scalar multiplication by a known integer via double-and-add algorithm, is cheap in TFHE—technically represented only by vector addition—and also it does not impose much noise overhead to the ciphertext. Note that for this reason, we will only consider NN's with integer weights $w_i^{(P)}$. In addition, let us emphasize that all operations are performed on a discretized and finite domain, hence the NN parameters must take this limitation into account in advance, i.e., during the NN creation process.

B. Evaluation of the Activation Function

The main focus now remains on the activation function, which can be incorporated within the bootstrapping procedure as it has been suggested by Carpov et al. [14]. As we already outlined within the bootstrapping algorithm, the evaluated function $f\colon \mathbb{Z}_{2N} \to \mathbb{T}$ must satisfy (a modified result of Carpov et al.)

$$f(-N+m) = -f(m) \tag{3}$$

for $m \in [0, N)$. This follows from the negacyclic property of the test vector, which encodes the function and which is technically a polynomial $\bmod X^N+1$. N.b., the property (3) cannot be completely avoided, e.g., by rescaling at line 1 of Algorithm 1 from $\bar{a}_i \leftarrow \lfloor 2Na_i \rceil$ to $\bar{a}_i \leftarrow \lfloor Na_i \rceil$ – in such a case, the result would have an unpredictable sign.

However, if we insist on an activation function *without* the property (3), e.g., $f = \tanh$, we must prevent the input values from reaching the negacyclic property by an appropriate choice of N; cf. Figure 2. We discuss choice of N later in detail. However, for TFHE plaintext resolution it means that $2N$ values must be distinguishable, represented by torus values $\frac{1}{2N}\mathbb{Z}/\mathbb{Z} \subset \mathbb{T}$.

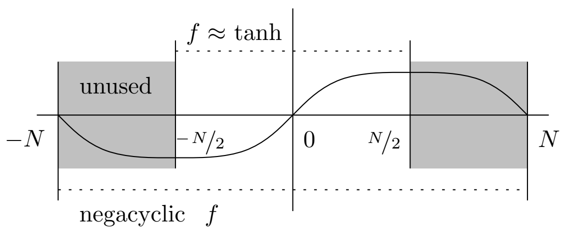

Fig. 2. Usage of a non-negacyclic activation function ($\tanh$) on a limited interval.

C. Example Discretization of Hyperbolic Tangent

Let us choose hyperbolic tangent, $\tanh\colon \mathbb{R} \to (-1, 1)$, as an example activation function. We show how discretization narrows down the unused interval. We choose a precision parameter $d \in \mathbb{N}$ and we rescale and discretize $\tanh$ to integers, we use $f_d(x) = \lfloor d \cdot \tanh(x/d) \rceil$. I.e., $f_d\colon \mathbb{Z}_{2N} \to [-d, d]$ and $f_d(1) = 1$. Note that f_d becomes constant starting

$$x_d = \lceil \tanh^{-1}(1 - 1/2d) \cdot d \rceil. \tag{4}$$

It follows that after discretization, the usable interval can be extended by constant values from previously unused interval, i.e., as far as $N - x_d$; cf. Figure 3, where we have chosen $d = 8$.

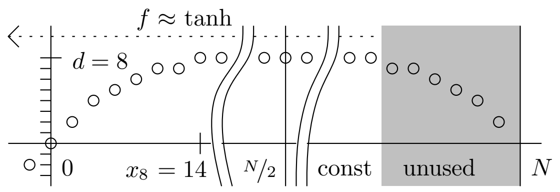

Fig. 3. Extending the usable interval by constant values (n.b., only every second value is displayed).

Let us now discuss how N must be chosen in order to prevent an overflow. Note that during NN evaluation, the largest value may occur after (weighted) summation, cf. Figure 1 and (2). Let us denote

$$W = \max_{P \in \mathcal{P}} \Big[\sum_{i=1}^{k_P} \big| w_i^{(P)} \big| \Big], \tag{5}$$

where $\mathcal{P}$ is the set of all perceptrons of given NN. Then, since inputs v_i in (2) are limited by the (bounded) image of f, we need $d \cdot W \leq N - x_d$, i.e.,

$$N \geq x_d + d \cdot W. \tag{6}$$

Note that the size x_d of the unused interval only depends on d and grows $\approx 1/2(d \log d)$. Indeed,

$$\lim_{d \to \infty} \frac{\lceil \tanh^{-1}(1 - 1/2d) \cdot d \rceil}{d \log d} = \frac{1}{2}. \tag{7}$$

This approach allows us to make use of most of the interval, even without a negacyclic function.

D. Motivation: Verifiable NN Evaluation

In our recent research, we introduced a framework named the VERAGREG Framework [15], which extends Additively Homomorphic Encryption (AHE) by a verification feature. This feature aims to guarantee that the computing party indeed computes what it claims to. In case of addition, this basically means an index list of values, which are (supposed to be) involved in the encrypted aggregate sum. Our current aim is to implement the verification feature to homomorphic evaluation of NN's.

There exists a scheme by Fiore et al. [16], which allows for arbitrary verifiable computations over encrypted data, i.e., a verifiable FHE scheme. Note that their scheme is currently tailored for evaluation of multiple kinds of polynomials, i.e., not for a generic (and bounded) activation function; cf. Table 1 in their paper. Interestingly, for some types of polynomials, both input and function privacy can be achieved. Another analogous problem with polynomials is addressed in a recent paper by Gajera et al. [17].

Our goal is a scheme, which would allow for verifiable evaluation of a NN over encrypted data *without disclosure* of the actual NN, but rather in a zero-knowledge manner. Such an approach aims to increase credibility of the whole system. As the first step, we decided to implement a library with (W)TFHE operations. The library aims to help us, as well as others, with rapid prototyping of our/their novel ideas related to, or enhancements of TFHE.

IV. LIBRARY FOR RAPID PROTOTYPING

Since there exist several variants and modifications of the original LWE (e.g., [13], [4], [18], [19]), including those of TFHE (e.g., [20], [14]), it has shown to be convenient to have an implementation, which would be easy to modify as well as would it allow for interactivity. Currently, TFHE is only available as a C++ library [21], which is much more rigid, compared to a library written in an interpreted language. For this reason, we decided to write the library in Ruby, named WTFHE and released under MIT license at [22].

Ruby is shipped with an interactive shell (e.g., `irb`), hence it becomes very easy to create own instances of all underlying objects ranging from the torus elements and polynomials over the torus to T(R)LWE and T(R)GSW ciphers. Note that it is also possible to modify the methods on-the-fly, hence one can easily verify own ideas of new enhancements or features.

A. Code Structure

Our library consists of several classes, each defined in a corresponding file and supported by a comprehensive documentation. In Figure 4, we outline their dependencies.

```
WTFHE ................ Global cipher interface
|__ TRGSW .... TRGSW cipher (for bootstrapping)
|   |__ TRLWE .................... TRLWE cipher
|       |__ TrlweSample ........ TRLWE sample
|           |__ PolyN ... Polynomials mod X^N + 1
|               |__ Torus .......... Torus elements
|                   |__ TorusInt ...... int repres.
|                   |__ TorusFloat ... float rep.
|__ TRLWE
|   |__ TLWE ................ Actual TFHE cipher
|       |__ (dtto)
|__ Noise........... Module for noise operations
```

Fig. 4. Dependencies of implemented objects.

V. CONCLUSIONS & FUTURE DIRECTIONS

We revisited the TFHE scheme and discussed its usage for evaluation of NN's over encrypted data. In particular, we focused on effective resource utilization under the limitations that follow from TFHE construction. On top of that, we suggested to incorporate a verifiability feature to the evaluation process in order to increase credibility of the whole system.

In order to simplify the development of new modifications or enhancements to the TFHE cipher with particular respect to NN evaluation, we implemented a scalable, modifiable and interactive Ruby library named WTFHE. Thanks to its design, our library enables to decrease the idea-to-proof-of-concept time considerably.

As per our future plans, we aim to implement the variants suggested by Carpov et al. [14] and evaluate them with respect to prospective involvement in NN computation, particularly from the performance point of view. Finally, our ultimate goal is to introduce the verification feature to NN evaluation, similarly to our VERAGREG Framework, which does that only for AHE.

REFERENCES

[1] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[2] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*. Stanford University, 2009, vol. 20, no. 09.

[3] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Annual Cryptology Conference*. Springer, 2013, pp. 75–92.

[4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.

[5] S. Halevi and V. Shoup, "Algorithms in HElib," in *Annual Cryptology Conference*. Springer, 2014, pp. 554–571.

[6] "Microsoft SEAL (release 3.2)," https://github.com/Microsoft/SEAL, Feb. 2019, microsoft Research, Redmond, WA.

[7] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Annual International Cryptology Conference*. Springer, 2018, pp. 483–512.

[8] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *international conference on the theory and application of cryptology and information security*. Springer, 2016, pp. 3–33.

[9] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[10] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[11] L. Ducas and D. Micciancio, "Fhew: bootstrapping homomorphic encryption in less than a second," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 617–640.

[12] M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer, "Estimate all the {LWE, NTRU} schemes!" in *International Conference on Security and Cryptography for Networks*. Springer, 2018, pp. 351–367.

[13] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23.

[14] S. Carpov, M. Izabachène, and V. Mollimard, "New techniques for multi-value input homomorphic evaluation and applications," in *Cryptographers' Track at the RSA Conference*. Springer, 2019, pp. 106–126.

[15] J. Klemsa, L. Kencl, and T. Vaněk, "VeraGreg: A Framework for Verifiable Privacy-Preserving Data Aggregation," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 1820–1825.

[16] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 844–855.

[17] H. Gajera, M. Giraud, D. Gérault, M. L. Das, and P. Lafourcade, "Verifiable and private oblivious polynomial evaluation," in *IFIP International Conference on Information Security Theory and Practice*. Springer, 2019, pp. 49–65.

[18] J. H. Cheon and D. Stehlé, "Fully homomophic encryption over the integers revisited," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 513–536.

[19] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "A homomorphic lwe based e-voting scheme," in *Post-Quantum Cryptography*. Springer, 2016, pp. 245–265.

[20] C. Boura, N. Gama, and M. Georgieva, "Chimera: a unified framework for b/fv, tfhe and heaan fully homomorphic encryption and predictions for deep learning." *IACR Cryptology ePrint Archive*, vol. 2018, p. 758, 2018.

[21] "TFHE: Fast Fully Homomorphic Encryption Library over the Torus," https://github.com/tfhe/tfhe, 2016.

[22] "WTFHE: neural-netWork-ready Torus Fully Homomorphic Encryption," https://gitlab.fit.cvut.cz/klemsjak/wtfhe, 2020.