New techniques for multi-value homomorphic evaluation and applications

Sergiu Carpov, Malika Izabachène, and Victor Mollimard

CEA LIST, Point Courrier 172, 91191 Gif-sur-Yvette Cedex, France

Abstract. In this paper, we propose a new technique to perform several homomorphic operations in one bootstrapping call over a multi-value plaintext space. Our construction relies on the FHEW-based gate bootstrapping: we analyze its structure and propose a strategy we call *multi-value bootstrapping* which allows to bootstrap an arbitrary function in an efficient way.

The security of our scheme relies on the LWE assumption over the torus. We give three applications: the first one is the efficient evaluation of an arbitrary boolean function (LUT), the second one is the optimization of the circuit bootstrapping from (Asiacrypt'2017) which allows to compose circuits in a leveled mode, the third one is the homomorphic evaluation of a neural network where the linear part is evaluated using a generalization of the key-switching procedure and the non-linear part is evaluated with our multi-value bootstrapping.

We have implemented the proposed method and were able to evaluate arbitrary 6-to-6 LUTs under 1.2 seconds. Our implementation is based on the TFHE library but can be easily integrated into other homomorphic libraries based on the same structure, such as FHEW (Eurocrypt'2015). The number of LUT outputs does not influence the execution time by a lot, e.g. evaluation of additional 128 outputs on the same 6 input bits takes only 0.05 more seconds.

Keywords: lwe-based FHE, multi-value bootstrapping, homomorphic LUT

1 Introduction

Fully homomorphic encryption (FHE) allows to perform arbitrary computations directly over encrypted data. The first FHE scheme has been proposed by Gentry [12]. The construction relies on a technique called *bootstrapping*, which handles noise increase in FHE ciphertexts. This construction theoretically enables to execute any computation directly over encrypted data but remains slow in practice. Several works ([11, 4, 15, 20, 17] for example) followed Gentry's initial proposal and contributed to further improve FHE efficiency.

Fully homomorphic encryption schemes are divided in two types of constructions. The first one is based on Gentry's initial proposal, where basically the bootstrapping procedure consists of the evaluation of the decryption circuit at gate level. In this case, the operations remain slow but their design allows to pack data efficiently using batching techniques.

The second one is based on the Gentry, Sahai and Waters Somewhat homomorphic scheme [13] proposed in 2013 which supports branching programs with polynomial noise overhead and deterministic automata logic. Alperin-Sheriff and Peikert [2] improved the bootstrapping by implementing an efficient homomorphic arithmetic function, showing that boolean function and Barighton circuit can be avoided in bootstrapping. In 2015, Ducas and Micciancio [10] gave a construction of bootstrapping with NAND gate evaluation, named FHEW, and suggested extension for larger gates. They provided an implementation for their scheme taking less than a second per bootstrapping on a single core. Biasse and Riuz [3] adapted the FHEW construction for arbitrary gates. Recently, Chillotti, Gama, Georgieva and Izabachène [7, 16] also improved the bootstrapping procedure and provided a construction named TFHE. Their implementation [9] runs in less than 13ms for any binary gate and 26ms for the MUX gate. They also proposed new techniques for the TFHE toolbox which allow to pack data and compose bootstrapped gates in a leveled mode with a new procedure they called *circuit bootstrapping*. Recently, Bonnoron, Ducas, and Fillinger [14] introduced a FHEW-based type scheme which allows to perform more computation per bootstrapping call. They implemented their method for the evaluation of a 6-to-6 bit LUT in about 10 seconds.

In this work, we also focus on the second line of FHE schemes based on the Gentry, Sahai and Waters scheme. In this case, the bootstrapping procedure is implemented via an homomorphic accumulator which evaluates the linear part of decryption function followed by the non-linear part. In particular, the bootstrapping algorithm can change the message. For this line of schemes, the structure of the bootstrapping can be divided in 3 steps:

1. In the first step, the coefficients $(\mathbf{a}, b)$ of an input LWE ciphertext $\mathbf{c} = (\mathbf{a}, b)$ are mapped to $\mathbb{Z}_T$. A cyclic multiplicative group $\mathcal{G}$, where $\mathbb{Z}_T \simeq \mathcal{G}$, is used for an equivalent representation of $\mathbb{Z}_T$ elements. $\mathcal{G}$ contains all the powers of X: $X^0, \dots, X^{T-1}$. T is defined as the smallest integer verifying $X^T \mod \Phi(X) = 1$ where $\Phi(X)$ is the quotient polynomial defining the input Ring-LWE scheme. Most of the times $\Phi(X)$ is the T-th cyclotomic polynomial.
2. In this step, a message m encrypted in input ciphertext $\mathbf{c} = (\mathbf{a}, b)$ is transformed to an intermediary GSW SHE scheme as an encryption of X^m. Message $m \in \mathbb{Z}_T$ is obtained from $\mathbf{c} = (\mathbf{a}, b)$ using linear transformation $b - \mathbf{a} \cdot \mathbf{s} \equiv m$ (i.e. the linear part of the decryption algorithm). Given encryptions of X^{s_i} one can homomorphically apply linear mapping φ to $\mathbf{c}$. We obtain the so-called accumulator ACC which contains an encryption of $X^{\varphi(\mathbf{c})} \in \mathcal{G}$.
3. At the third step, a test polynomial $\mathsf{TV}_F \in \mathcal{G}$ is multiplied to ACC. The test polynomial encodes output values of a function F for each possible input message $m \in \mathbb{Z}_T$. Here F is a function from $\mathbb{Z}_T$ to $\mathbb{Z}_T$. It finally extracts an LWE encryption of $F(m)$ from $\mathsf{TV}_F \cdot \mathsf{ACC}$ (or from $\mathsf{ACC} \cdot \mathsf{TV}_F$ if TV_F is applied after computing the accumulator) with a modified noise. As input message m is a noised version of the actual message encrypted in $\mathbf{c} = (\mathbf{a}, b)$

function F is a composition of a "payload" function $f : \mathbb{Z}_t \to \mathbb{Z}_t$ and a rounding function $r : \mathbb{Z}_T \to \mathbb{Z}_t$. The rounding function corresponds to the final non-linear step of ciphertext $\mathbf{c}$ decryption.

Figure 1 gives a schematic overview of bootstrapping steps.

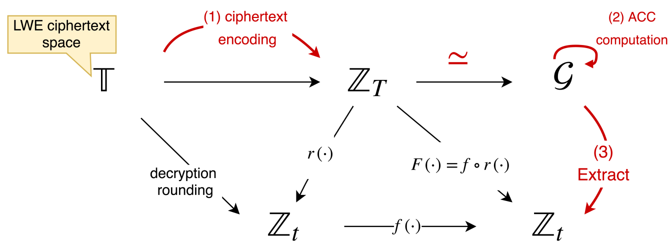

Fig. 1. Structure of the bootstrapping Algorithm. Setp (1): The ciphertext of m is rescaled modulo T and the operations are mapped over the cyclic group $\mathcal{G}$ where $\mathcal{G} = \langle X \rangle$ is the group of T-th roots of unity associated to the cyclotomic polynomial $\Phi_T(X)$ (for example). Step (2): the accumulator ACC is computed using blind shift operations in $\mathcal{G}$ which use encryptions of secret key in the powers of X. Step (3): a test polynomial is applied to ACC, it can also be applied before blind shift operations, and an LWE ciphertext of $f(m)$ is extracted from ACC using the encoding of an alternative representation of f over $\mathbb{Z}_T$.

For example, in [14], step (1) corresponds to a modulus switching from Q to $T = pq$, step (2) computes the accumulator operation in the groups $\mathcal{G} = \{1, \ldots, X^p - 1\}$ and $\mathcal{G} = \{1, \ldots, Y^q - 1\}$ for primes p and q and recomposes the result in the circulant ring $\mathbb{Z}[Z]/(Z^{pq} - 1)$; at step (3), a test polynomial (encoding $F(x) = f(\lfloor tx/pq \rceil)$ where f is an arbitrary function) is applied to the accumulator and a LWE ciphertext of $f(m)$ is extracted, where the extraction is implemented by the trace function. In [9], $\mathcal{G}$ is the multiplicative group $\{1, X, \ldots, X^{2N-1}\}$ where N is a power of 2. Function f implements a rounding (i.e. torus most significant bit extraction); step (1) does the rounding from $\mathbb{T}$ to $\mathbb{Z}_{2N}$ and the test polynomial is applied before the computation of the accumulator ACC; step (2) computes $\mathsf{ACC} \in \mathcal{G}$ with a blind rotation; step (3) extracts $\mathsf{LWE}(f(m))$ by extracting the constant coefficient of $\mathsf{TV}_F \cdot \mathsf{ACC}$.

Our technique and comparisons to other works. In previous constructions, except [9], test polynomial TV_F is integrated at the end, after the accumulator is computed, we have $\mathsf{ACC} \cdot \mathsf{TV}_F$ [1]. In the TFHE gate bootstrapping of [9], the test polynomial TV_F is embedded in the accumulator from the very start when the accumulator is still noiseless and, at step 2 the accumulator is $\mathsf{TV}_F \cdot \mathsf{ACC}$.

[1] In this paragraph only the evaluation order of an expression matters and is used for a better illustration.

This allows to save a factor $\sqrt{N}$, where N is the dimension. On the other end, they are only able to encode two possible values in TFHE Gate bootstrapping. A naive idea for computing multi-value input function f would be to decompose f into p `Mux` gate functions and then combine the results of the p gate bootstrapping calls, but this method is quite inefficient. To optimize this construction, we define a common factor $\mathsf{TV}_F^{(0)}$ which is shared between all the p calls. The most expensive part is made once for the p calls. Then the specification with respect to the 2-value functions is made at the end using a second test polynomial $\mathsf{TV}_F^{(1)}$. This last step consists only of a multiplication by constant polynomial, which is much cheaper than p blind rotations.

The method we propose allows to evaluate multi-value functions efficiently. In the same vein, we can also evaluate several multi-value functions over the same input. Also, our scheme allows to decrease the output ciphertext noise by choosing a low-norm second-stage test polynomials when compared to previous methods integrating the test polynomial at the end.

Multi-value, multi-bit function. The multi-bit construction of [3] can be used to evaluate a function on 2^n values by replacing the encrypted output messages by the value of the function at this point. It can be seen as dividing the torus in 2^n sections and associating one value of the function to each section. In this work, we construct a multi-value bootstrapping which is used to evaluate a function on n' values. It decomposes the torus in at least n' sections and associates n' values of the function to the n' sections. This is why we call it multi-value instead of multi-bit.

Application and Implementation results. We show that the multi-value bootstrapping can be used to optimize the homomorphic evaluation of LUT functions. As additional applications, we also show how to efficiently compose LUTs, how to optimize the circuit bootstrapping of [16] and how to combine the multi-value bootstrapping with functional key-switch for generic neural networks evaluation. We implemented a 6-to-6 LUT bootstrapping in order to have a comparative case study with [14] and obtained an implementation running in less than 2 seconds for a concrete security of about 100 bits (asserted using the estimator from [18]).

Organization of the paper. We first review the mathematical backgrounds for LWE and GSW encryption over the torus and give the building blocks from the TFHE framework [9] used in our constructions. In section 3, we present the optimized multi-value bootstrapping together with test polynomial factorization. In section 4, we present applications to the homomorphic evaluation of arbitrary functions. We then describe our implementation results for the case of a 6-to-6 LUT function. Finally, we show how it could be used to optimize the circuit bootstrapping from [16] and to evaluate a computational neural network system.

2 Preliminaries

The set $\{0,1\}$ is written as $\mathbb{B}$. The set of vectors of size n in E is denoted E^n, and the set of $n \times m$ matrices with entries in E is noted $\mathcal{M}_{n,m}(E)$. The real torus $\mathbb{R} \mod 1$ is denoted $\mathbb{T}$. $\mathbb{T}_N[X]$ denotes the $\mathbb{Z}$-module $\mathbb{R}[X]/(X^N+1) \mod 1$ of torus polynomials, here N is a fixed power of 2 integer. The ring $\mathbb{Z}[X]/(X^N+1)$ is denoted $\mathfrak{R}$. The set of polynomials with binary coefficients is denoted $\mathbb{B}_N[X]$

2.1 Backgrounds on TFHE

In this work, we will use the torus representation from [7] of the LWE encryption scheme introduced by Regev [19] and the ring variant of Lyubashevsky et al [21].

Distance, Norm and Concentrated distribution We use the ℓ_p distance for torus elements. By abuse of notation, we denote as $\|\boldsymbol{x}\|_p$ the p-norm of the representative of $\boldsymbol{x} \in \mathbb{T}^k$ with all its coefficients in $\left]-\frac{1}{2}, \frac{1}{2}\right]$. Note that it satisfies the triangular inequalities while it is not a norm. For a torus polynomial $P(X)$ modulo X^N+1, we take the norm of its unique representative of degree $\leq N-1$. A distribution on the torus is concentrated iff its support is included in a ball of radius $\frac{1}{4}$ of $\mathbb{T}$ except with negligible probability. In this case, we can define the usual notion of expectation and variance over $\mathbb{T}$.

For a normal distribution $\mathcal{N}(0,\sigma^2)$ centered in 0 and of variance σ^2, we denote $\kappa(\varepsilon) = \min_k\{\Pr_{X\leftarrow\mathcal{N}(0,\sigma^2)}[|X| > k\cdot\sigma] < \varepsilon\}$. And we have $\Pr_{X\leftarrow\mathcal{N}(0,\sigma^2)}[|X| > k\cdot\sigma] = \mathrm{erf}(k/\sqrt{2})$. For example, for $\varepsilon = 2^{-64}$, we can take $\kappa(\varepsilon) > 9.16$ and for $\epsilon = 2^{-32}$, we can take $\kappa(\varepsilon) > 6.33$.

A real distribution X is said σ-subgaussian iff for all $t \in \mathbb{R}$, $\mathbb{E}(\mathsf{exp}(tX)) \leq \mathsf{exp}(\sigma^2 t^2/2)$. If X and X' are two independent σ and σ' subgaussian variables, then for all $\alpha, \gamma \in \mathbb{R}$, $\alpha X + \gamma X'$ is $\sqrt{\alpha^2\sigma^2 + \gamma^2\sigma'^2}$-subgaussian. All the errors in this document will follow subgaussian distributions.

In what follows, we review TFHE for encryption of torus polynomial elements.

TRLWE samples. To encrypt a message $\mu \in \mathbb{T}_N[X]$, one picks a Gaussian approximation of the preimage of $\varphi_s^{-1}(\mu)$ over the Ω-probability space of all possible choices of Gaussian noise. If the Gaussian noise α is small, we can define the expectation and the variance over the torus. The expectation of $\varphi_s(c)$ is equal to μ and its variance is equal to the variance of α. We refer to [7] for a more complete definition of the Ω-probability space.

Definition 2.1 (TRLWE). *Let $\mathcal{M}$ be a discrete subspace of $\mathbb{T}_N[X]$ and $\mu \in \mathcal{M}$ a message. Let $\boldsymbol{s} \in \mathbb{B}_N[X]^k$ a* TRLWE *secret key, where each coefficient is chosen uniformly at random. A* TRLWE *sample is a vector $\mathbf{c} = (\mathbf{a}, b)$ of $\mathbb{T}_N[X]^{k+1}$ which can be either :*

- *A trivial sample: $\mathbf{a} = 0$ and $b = \mu$. Note that this ciphertext is independent of the secret key.*

– *A fresh* TRLWE *sample of* μ *of standard deviation* α*:* $\mathbf{a}$ *is uniformly chosen in* $\mathbb{T}_N[X]^k$ *and* b *follows a continuous Gaussian distribution of standard deviation* α *centered in* $\mu + \mathbf{s} \cdot \mathbf{a}$ *and of variance* α^2*.*

– *Linear combination of fresh or trivial* TRLWE *samples.*

We define the phase $\varphi_{\boldsymbol{s}}(\boldsymbol{c})$ of a sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ under key $\boldsymbol{s} \in \mathbb{B}_N[X]^k$ as $\varphi_{\boldsymbol{s}}(\boldsymbol{c}) = b - \boldsymbol{s} \bullet \boldsymbol{a}$. Note that the phase function is a linear $(kN+1)$-lipschitzian function from $\mathbb{T}_N[X]^{k+1}$ to $\mathbb{T}_N[X]$.

We say that $\boldsymbol{c}$ is a valid TRLWE sample iff there exists a key $\boldsymbol{s} \in \mathbb{B}_N[X]^k$ such that the distribution of the phase $\varphi_{\boldsymbol{s}}(\boldsymbol{c})$ is concentrated over the Ω-space around the message μ, i.e. included in a ball of radius $< \frac{1}{4}$ around μ.

Note that $\mathbf{c} = \sum_{j=1}^{p} r_j \cdot \mathbf{c}_j$ is a valid TRLWE sample if $\mathbf{c}_1, \ldots, \mathbf{c}_p$ are valid TRLWE samples (under the same key) and $r_1, \ldots, r_p \in \mathfrak{R}$. We also use the function $\mathsf{msg}()$ defined as the expectation of the phase over the Ω-space.

If μ is in $\mathcal{M}$, one can decrypt a TRLWE sample $\mathbf{c}$ under secret key $\mathbf{s}$ with small noise (smaller that the packing radius) by rounding its phase to the nearest element of the discrete message space $\mathcal{M}$. We also use the function error $\mathsf{Err}(\cdot)$ of a sample defined as the difference between the phase and the message of the sample. We write $\mathsf{Var}(\mathsf{Err}(X))$ the variance of the error of X and $\|\mathsf{Err}(X)\|_\infty$ its amplitude. When X is a normal distribution we have $\|\mathsf{Err}(X)\|_\infty \leq \kappa(\varepsilon) \cdot \mathsf{Var}(\mathsf{Err}(X))$ with probability $1 - \varepsilon$.

Given p valid and independent TRLWE samples $c_1, \ldots, c_p$ under key s, if $c = \sum_{i=1}^{p} e_i \cdot c_i$, then $\mathsf{msg}(c) = \sum_{i=1}^{p} e_i \cdot \mathsf{msg}(c_i)$ with $\|\mathsf{Err}(c)\|_\infty \leq \sum_{i=1}^{p} \|e_i\|_1 \cdot \|\mathsf{Err}(c_i)\|$ and $\mathsf{Var}(\mathsf{Err}(c)) = \sum_{i=1}^{p} \|e_i\|_2^2 \cdot \mathsf{Var}(\mathsf{Err}(c_i))$.

The TRLWE problem consists of distinguishing TRLWE encryptions of $\mathbf{0}$ from random samples in $\mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$. When $N = 1$ and k is large, the TRLWE problem is the Scalar LWE problem over the torus and the TRLWE encryption is the LWE encryption over the torus. We denote it TLWE. When N is large and $k = 1$, the TRLWE problem is the LWE problem over torus polynomials with binary secrets. In addition, the TLWE and the TRLWE correspond to the Scale invariant variants defined in [22, 6, 8] and to the Ring-LWE from [21]. We refer to Section 6 of [7] for more details on security estimates on the LWE problem of the torus.

TRGSW samples . We define a gadget matrix that will be used to decompose over ring elements and to reverse back. Other choices of gadget basis are also possible.

$$\mathbf{H} = \left(\begin{array}{c|c|c} 1/B_g & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 1/B_g^{\ell} & \cdots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \cdots & 1/B_g \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/B_g^{\ell} \end{array}\right) \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X]).$$

A vector $v \in \mathbb{T}_N[X]^{k+1}$ can approximately be decomposed as $Dec_{H,\beta,\epsilon}(\boldsymbol{v}) = \boldsymbol{u}$ where $\boldsymbol{u} \in \mathfrak{R}^{(k+1)\ell}$, s.t. $\|\boldsymbol{u}\|_\infty \leq \beta$ and $\|\boldsymbol{u} \cdot H - \boldsymbol{v}\|_\infty \leq \epsilon$. We call $\beta \in \mathbb{R}_{>0}$ the quality parameter and $\epsilon \in \mathbb{R}_{>0}$ the precision of the decomposition.

In this paper, we use the gadget H where the decomposition in base B_g is a power of 2. We have $\beta = B_g/2$ and $\epsilon = 1/2B_g^{\ell}$.

Definition 2.2 (TRGSW Sample). *Let ℓ and $k \geq 1$ be two integers and $\alpha \geq 0$ be a noise parameter. Let $\mathbf{s} \in \mathbb{B}_N[X]^k$ be a* TRLWE *key, we say that $\mathbf{C} \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ is a fresh* TGSW *sample of $\mu \in \mathfrak{R}/\mathbf{H}^{\perp}$ with standard deviation α iff $\mathbf{C} = \mathbf{Z} + \mu \cdot \mathbf{H}$ where each row of $\mathbf{Z} \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ is a* TRLWE *sample of $\mathbf{0}$ with Gaussian standard deviation α. Reciprocally, we say that an element $\mathbf{C} \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ is a valid* TRGSW *sample iff there exists a unique polynomial $\mu \in \mathfrak{R}/\mathbf{H}^{\perp}$ and a unique key $\mathbf{s}$ such that each row of $\mathbf{C} - \mu \cdot \mathbf{H}$ is a valid* TRLWE *sample of 0 under the key $\mathbf{s}$. We call the polynomial μ the message of $\mathbf{C}$.*

Since a TRGSW sample consists of $(k+1)\ell$ TRLWE under the same secret key, the definition of the functions for the phase, message, error, norm and variance and the result on the sum of TRLWE samples can easily be extended for TRGSW samples.

External Product. We review the module multiplication of the messages of TRGSW and TRLWE samples from [5, 7]. The external product operation is defined as: $\boxdot : \mathbb{T}_N[X]^{k+1} \times \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X]) \to \mathbb{T}_N[X]^{k+1}$. The operation $\boxdot$ has the following property :

Theorem 2.3 (Homomorphic module multiplication). *If A is a valid* TRGSW *sample of μ_A and b is a valid* TRLWE *sample of μ_b. Then, if $\|Err(A \boxdot b)\|_\infty \leq \frac{1}{4}$, $A \boxdot b$ is a valid* TRLWE *sample of $\mu_A \cdot \mu_b$. We have $\mathsf{Var}(\mathsf{Err}(A \boxdot b)) \leq (k+1)\ell N\beta^2 \mathsf{Var}(\mathsf{Err}(A)) + (1+kN)\|\mu_A\|_2^2\epsilon^2 + \|\mu_A\|_2^2 \mathsf{Var}(\mathsf{Err}(b))$ where β and ϵ are the parameters used in the decomposition $Dec_{h,\beta,\epsilon}()$.*

Assumption 2.4 (Independence heuristic). All the previous results rely on the Gaussian Heuristic: all the error coefficients of TRLWE or TRGSW samples of the linear combinations we consider are independent and concentrated. In particular, we assume that they are σ-subgaussian where σ is the square-root of their variance.

2.2 TFHE gate bootstrapping

We review the TFHE gate bootstrapping and the key-switching procedure from [7, 16]. The TFHE gate bootstrapping change the noise and can also change the dimension of the ciphertexts. We note with an under-bar the input parameters and with an upper-bar the output parameters, where needed.

Definition 2.5. *Let* $\underline{\mathfrak{K}} \in \mathbb{B}^n$*,* $\bar{\mathfrak{K}} \in \mathbb{B}_N^k$ *and* α *be a noise parameter. We define the bootstrapping key* $\mathsf{BK}_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\alpha}$ *as the sequence of* n TGSW *samples* $\mathsf{BK}_i \in \mathsf{TGSW}_{\bar{\mathfrak{K}},\alpha}(\underline{\mathfrak{K}}_i)$*.*

TFHE gate bootstrapping. The ternary `Mux` gate takes three boolean values c, d_0, d_1 and returns $\texttt{Mux}(c, d_0, d_1) = (c \wedge d_1) \oplus ((1 - c) \wedge d_0)$. We also write $\texttt{Mux}(c, d_0, d_1) = c?d_1 : d_0$.

The controlled `Mux` gate, `CMux` takes in input samples $\mathbf{d}_0, \mathbf{d_1}$ of messages μ_0, μ_1, a TRGSW sample $\mathbf{C}$ of a message bit m and returns a TRLWE sample of message μ_0 if $m = 0$ and μ_1 if $m = 1$. Lemma 2.6 gives the error propagation of `CMux`.

Lemma 2.6. *Let* $\mathbf{d}_0, \mathbf{d}_1$ *be* TRLWE *samples and* $\mathbf{C} \in \mathsf{TRGSW}_{\mathbf{s}}(m)$ *where message* $m \in \{0, 1\}$*. Then,* $\mathit{msg}(\texttt{CMux}(\mathbf{C}, \mathbf{d}_1, \mathbf{d}_0)) = \mathit{msg}(\mathbf{C})?\mathit{msg}(\mathbf{d}_1) : \mathit{msg}(\mathbf{d}_0)$ *and we have:* $\mathit{Var}(\mathit{Err}(\texttt{CMux}(\mathbf{C}, \mathbf{d}_1, \mathbf{d}_0))) \leq \max(\mathit{Var}(\mathit{Err}(\mathbf{d}_0)), \mathit{Var}(\mathit{Err}(\mathbf{d}_1))) + \vartheta(\mathbf{C})$ *where* $\vartheta(\mathbf{C}) = (k + 1)\ell N\beta^2 \mathit{Var}(\mathit{Err}(\mathbf{C})) + (1 + kN)\epsilon^2$*.*

The gate bootstrapping from [16] also uses the $\mathsf{BlindRotate}$ algorithm recalled below. If $\mathbf{c} = (a_1, \dots, a_p, b)$ is a LWE ciphertext under secret key $\mathbf{s}$, the algorithm computes the blind rotation of v by the phase of c.

Algorithm 1 $\mathsf{BlindRotate}$

Input: A TRLWE sample $\boldsymbol{c}$ of $v \in \mathbb{T}_N[X]$ with key K.
1: $p + 1$ int. coefficients $a_1, \dots, a_p, b \in \mathbb{Z}/2N\mathbb{Z}$
2: p TRGSW samples $C_1, \dots, C_p$ of $s_1, \dots, s_p \in \mathbb{B}$ with key K
Output: A TRLWE sample of $X^{-\rho}.v$ where $\rho = b - \sum_{i=1}^{p} s_i.a_i \mod 2N$ with key K
3: $\mathsf{ACC} \leftarrow X^{-b} \bullet \boldsymbol{c}$
4: **for** $i = 1$ **to** p
5: $\quad \mathsf{ACC} \leftarrow \texttt{CMux}(C_i, X^{a_i} \cdot \mathsf{ACC}, \mathsf{ACC})$
6: **return** ACC

Theorem 2.7. *Let* $\alpha > 0 \in \mathbb{R}$ *be a noise parameter,* $\mathfrak{K} \in \mathbb{B}^n$ *be a* TLWE *secret key and* $K \in \mathbb{B}_N[X]^k$ *be its* TRLWE *interpretation. Given one sample* $\mathbf{c} \in \mathsf{TRLWE}_K(v)$ *with* $v \in \mathbb{T}_N[X]$*,* $p + 1$ *integers* $a_1, \dots, a_p, b \in \mathbb{Z}/2N\mathbb{Z}$*, and* p TRGSW *ciphertexts* $\mathbf{C}_1, \dots, \mathbf{C}_p$ *where each* $\mathbf{C}_i \in \mathsf{TRGSW}_{K,\alpha}(s_i)$ *for* $s_i \in \mathbb{B}$ *the* $\mathsf{BlindRotate}$ *algorithm outputs a sample* $\mathsf{ACC} \in \mathsf{TRLWE}_K(X^{-\rho} \cdot v)$ *where* $\rho = b - \sum_{i=1}^{p} a_i s_i$ *such that* $\mathit{Var}(\mathit{Err}(\mathsf{ACC})) \leq \mathit{Var}(\mathit{Err}(\mathbf{c})) + p(k+1)\ell N\beta^2\vartheta_C + p(1+kN)\epsilon^2$ *where* $\vartheta_C = \alpha^2$*.*

TRLWE-*to*-TLWE *sample extraction.* Given one TRLWE sample of message $\mu \in \mathbb{T}_N[X]$ the $\mathsf{SampleExtract}$ procedure allows to extract a TLWE sample of a single coefficient of polynomial μ. Indeed, a TRLWE ciphertext of message $\mu \in \mathbb{T}_N[X]$ of dimension k under a secret key $K \in \mathbb{B}_N[X]$ can alternatively be seen as N TLWE ciphertexts whose messages are the coefficients of μ. It is of dimension $n = kN$ and the secret key $\mathfrak{K}$ is in $\mathbb{B}^n$, where $K_i = \sum_{j=0}^{N-1} \mathfrak{K}_{N(i-1)+j+1} X^j$.

Functional key-switching. The functional key-switching procedure allows to switch between different parameter sets and between scalar and polynomial message space. It allows to homomorphically evaluate a morphism from $\mathbb{Z}$-module $\mathbb{T}^p$ to $\mathbb{T}_N[X]$. We recall the extension of the key-switching procedure given in Section 2.2 of [16] where the morphism f is public:

Algorithm 2 TLWE-to-TRLWE public functional key-switch

Input: p TLWE samples $\mathfrak{c}^{(z)} = (\mathfrak{a}^{(z)}, \mathfrak{b}^{(z)}) \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_z)$ for $z = 1, \ldots, p$, a public R-lipschitzian morphism f from $\mathbb{T}^p$ to $\mathbb{T}_N[X]$, $\mathsf{KS}_{i,j} \in \mathsf{TRLWE}_K(\frac{\mathfrak{K}_i}{2^j})$.
Output: A TRLWE sample $\boldsymbol{c} \in \mathsf{TRLWE}_K(f(\mu_1, \ldots, \mu_p))$

for $i \in [\![1, n]\!]$ **do**
  Let $a_i = f(\mathfrak{a}_i^{(1)}, \ldots, \mathfrak{a}_i^{(p)})$
  Let $\tilde{a}_i$ be the closest multiple of $1/2^t$ to a_i (i.e. $\|\tilde{a}_i - a_i\|_\infty < 2^{-(t+1)}$)
  Binary decompose each $\tilde{a}_i = \sum_{j=1}^{t} \tilde{a}_{i,j} \cdot 2^{-j}$ where $\tilde{a}_{i,j} \in \mathbb{B}_N[X]$
end for
return $(0, f(\mathfrak{b}^{(1)}, \ldots, \mathfrak{b}^{(p)})) - \sum_{i=1}^{n} \sum_{j=1}^{t} \tilde{a}_{i,j} \times \mathsf{KS}_{i,j}$

Theorem 2.8. *(Public functional key-switch) Given p TLWE samples $\mathfrak{c}^{(z)}$ under the same key $\mathfrak{K}$ of μ_z with $z = 1, \ldots, p$, a public R-lipschitzian morphism f from $\mathbb{T}^p$ to $\mathbb{T}_N[X]$, and a family of samples $\mathsf{KS}_{i,j} \in \mathsf{TRLWE}_{K,\gamma}(\frac{\mathfrak{K}_i}{2^j})$ with standard deviation γ, Algorithm 2 outputs a TRLWE sample $\boldsymbol{c} \in \mathsf{TRLWE}_K(f(\mu_1, \ldots, \mu_p))$ with $\mathsf{Var}(\mathsf{Err}(\boldsymbol{c})) \leq R^2 \mathsf{Var}(\mathsf{Err}(\mathfrak{c})) + ntN\vartheta_{\mathsf{KS}} + nN2^{-2(t+1)}$, where $\vartheta_{\mathsf{KS}} = \gamma^2$ is the variance of the error of KS.*

For $p = 1$ f is the identity function and we retrieve the classical key-switching where the $\mathsf{KS}_{i,j}$ is a sample $\mathsf{TLWE}_{\boldsymbol{s},\gamma}(\mathfrak{c}_i \cdot 2^{-j})$ for $i \in [\![1, n]\!]$ and $j \in [1, t]$. In this case, the output is a TLWE sample $\mathbf{c}$ of the same input message μ_1 and secret $\boldsymbol{s}$, where $\mathsf{Var}(\mathsf{Err}(\mathbf{c})) \leq \mathsf{Var}(\mathsf{Err}(\mathfrak{c})) + nt\gamma^2 + n2^{-2(t+1)}$.

We are now ready to recall the TFHE gate bootstrapping in Algorithm 3. The TFHE gate bootstrapping algorithm takes as inputs a constant $\mu \in \mathbb{T}$, a TLWE sample of $x \cdot \frac{1}{2}$ with $x \in \mathbb{B}$, a bootstrapping key and returns a TLWE sample of $x \cdot \mu$ with a controlled error.

Lines 1 to 4 compute a TRLWE sample of message $X^\varphi \cdot v$ where φ is the phase of $\underline{\mathfrak{c}}$ (actually an approximated phase because of rescaling in line 2). The $\mathsf{SampleExtract}$ extracts its constant coefficient ($\hat{\mu}$ if $x = 1$ and $-\hat{\mu}$ if $x = 0$) encrypted in a TLWE sample. The final addition allows to either obtain a TLWE

Algorithm 3 TFHE gate bootstrapping

Input: A constant $\mu \in \mathbb{T}$, a TLWE sample $\underline{\mathfrak{c}} = (\underline{\mathfrak{a}}, \underline{\mathfrak{b}}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}},\underline{\eta}}(x \cdot \frac{1}{2})$ with $x \in \mathbb{B}$, a bootstrapping key $\mathsf{BK}_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\alpha} = \left(\mathsf{BK}_i \in \mathsf{TRGSW}_{\bar{K},\alpha}(\underline{\mathfrak{K}}_i)\right)_{i \in [\![1,n]\!]}$ where $\bar{K}$ is the TRLWE interpretation of $\bar{\mathfrak{K}}$.

Output: A TLWE sample $\bar{\mathfrak{c}} = (\bar{\mathfrak{a}}, \bar{\mathfrak{b}}) \in \mathsf{TLWE}_{\bar{\mathfrak{K}},\bar{\eta}}(x \cdot \mu)$

1: Let $\hat{\mu} = \frac{1}{2}\mu \in \mathbb{T}$ (Pick one of the two possible values)
2: Let $b = \lfloor 2N\underline{\mathfrak{b}} \rceil$ and $a_i = \lfloor 2N\underline{\mathfrak{a}}_{\mathbf{i}} \rceil \in \mathbb{Z}$ for each $i \in [\![1,n]\!]$
3: Let $\mathsf{TV}_F := (1 + X + \cdots + X^{N-1}) \cdot X^{\frac{N}{2}} \cdot \hat{\mu} \in \mathbb{T}_N[X]$
4: $\mathsf{ACC} \leftarrow \mathsf{BlindRotate}((\mathbf{0}, v), (a_1, \ldots, a_n, b), (\mathsf{BK}_1, \ldots, \mathsf{BK}_n))$
5: Return $(\mathbf{0}, \hat{\mu}) + \mathsf{SampleExtract}(\mathrm{ACC})$

sample of 0 or a TLWE sample of $2 \cdot \hat{\mu} = \mu$. The error of the output ciphertext is obtained from Theorem 2.7 and the error of the $\mathsf{SampleExtract}$ procedure. An internal error δ is introduced in line 2 by the rescaling. We have $\delta \leq \frac{h+1}{4N}$ where h is the number of non-zero coefficients of TLWE secret key $\underline{\mathfrak{K}}$ and $4N$ comes from the rescaling by $2N$ and rounding of $(\underline{\mathfrak{a}}, \underline{\mathfrak{b}})$ coefficients. This error does not influence the output.

Theorem 2.9 (TFHE gate boostrapping). *Let $\underline{\mathfrak{K}} \in \mathbb{B}^n$ and $\bar{\mathfrak{K}} \in \mathbb{B}^{kN}$ be two TLWE secret keys, $\bar{K} \in \mathbb{B}_N[X]^k$ be the TRLWE interpretation of $\bar{\mathfrak{K}}$ and $\alpha > 0 \in \mathbb{R}$ a noise parameter. Let $\mathsf{BK}_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\alpha}$ be a bootstrapping key, i.e n samples $\mathsf{BK}_i \in \mathsf{TRGSW}_{\bar{K},\alpha}(\underline{\mathfrak{K}}_i)$ for $i \in [\![1,n]\!]$. Given a constant $\mu \in \mathbb{T}$ and a sample $\underline{\mathfrak{c}} \in \mathbb{T}^{n+1}$, Algorithm 3 outputs a TLWE sample $\bar{\mathfrak{c}} \in \mathsf{TLWE}_{\bar{\mathfrak{K}}}(\bar{\mu})$ where $\bar{\mu} = 0$ if $|\varphi_{\underline{\mathfrak{K}}}(\underline{\mathfrak{c}})| < \frac{1}{4} - \delta$ and $\bar{\mu} = \mu$ if $|\varphi_{\underline{\mathfrak{K}}}(\underline{\mathfrak{c}})| > \frac{1}{4} + \delta$. We have $\mathsf{Var}(\mathsf{Err}(\bar{\mathfrak{c}})) \leq n(k+1)\ell N\beta^2\vartheta_{BK} + n(1+kN)\epsilon^2$ where ϑ_{BK} is $\mathsf{Var}(\mathsf{Err}(\mathsf{BK}_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\alpha})) = \alpha^2$.*

3 Multi-value bootstrapping

In the previous section, we recall the bootstrapping procedures based on an auxiliary GSW scheme. Instead of the bootstrapping procedures where only a "re-encryption" of input ciphertext is made, we explain here how to bootstrapp an arbitrary function of the input message. For example in [7] the arbitrary function was the rounding (or modulus switching) of ciphertext decryption function. Recall, $\mathcal{G} = \langle X \rangle$ is the group of powers of X where X is a $2N$-th root of unity. This corresponds to the cyclotomic polynomial $\Phi_{2N}(X) = X^N + 1$ defining the TRLWE ciphertext polynomials.

The bootstrapping procedure consists of a linear step where an approximate phase $m \in \mathbb{Z}_{2N}$ of the input ciphertext $\mathbf{c}$ is computed followed by a non-linear step described by the following relation, here $R(X) \in \mathbb{Z}_N[X]$ is a polynomial with zero-degree coefficient equal to zero:

$$\mathsf{TV}_F(X) \cdot X^m \equiv F(m) + R(X) \mod \Phi_{2N}(X) \tag{1}$$

To ease the exposition, only the plaintext counterpart is presented. The $\mathsf{BlindRotate}$ procedure is used to obtain ACC which encrypts the phase m in the form of a power of X. This new representation is then multiplied by a test polynomial TV_F, for a function $F : \mathbb{Z}_{2N} \to \mathbb{Z}_{2N}$. In the zero-degree coefficient of the resulting polynomial the evaluation of function F in point m is obtained. Several possibilities to evaluate relation (1) exist. Hereafter we present 3 different ways to perform this evaluation and discuss their advantages and drawbacks.

$\mathbf{\mathsf{TV}_{\boldsymbol{F}}\left(\boldsymbol{X}\right) \cdot \boldsymbol{X}^{\boldsymbol{m}}}$ – The first one is to start the $\mathsf{BlindRotate}$ procedure with TV_F already encoded in ACC. The main advantage is that the output noise is independent of the test polynomial and is the lowest possible. The drawback is that only one function can be computed per bootstrapping procedure. This is how TV_F is encoded in the bootstrapping of [7].

$\boldsymbol{X}^{\boldsymbol{m}} \cdot \mathbf{\mathsf{TV}_{\boldsymbol{F}}\left(\boldsymbol{X}\right)}$ – Another possibility is to integrate TV_F after the $\mathsf{BlindRotate}$ procedure is performed. In this case, one can use several test polynomials and thus, compute several functions in the same input. This is how TV_F is encoded in the bootstrapping of [10, 3, 14]. The main drawback is that output ciphertext noise depends on test polynomial coefficient values.

$\mathbf{\mathsf{TV}^{(0)}\left(\boldsymbol{X}\right) \cdot \boldsymbol{X}^{\boldsymbol{m}} \cdot \mathsf{TV}_{\boldsymbol{F}}^{(1)}\left(\boldsymbol{X}\right)}$ – Finally, we can split test polynomial TV_F into two factors, with a first-phase factor $TV^{(0)}$ and a second-phase factor $\mathsf{TV}_F^{(1)}\left(X\right)$ test polynomials. The first-phase factor $\mathsf{TV}^{(0)}$ does not depend on the evaluated function F. Thus, as in the previous case, using different second-phase test polynomials we are able to evaluate several functions on the same input. Another condition when performing the factorization is to obtain the second-phase factors with low-norm coefficients. This is needed in order to obtain small noise increase in output ciphertexts. We conclude that this new evaluation technique allows to leverage the best of the first two possibilities.

The test polynomial is specific to a function f we want to evaluate. As the phase m is a noised version of the message of the input $\mathbf{c}$, it should be rounded before function f is applied to. We have $F = f \circ \texttt{round}$, where the function F is a composition of a rounding function and the "payload" function.

In the next subsection, we give a possible way to factorize test polynomials. Afterwards, we examine an updated version of Algorithm 3 which implements a bootstrapping procedure where the test polynomials are split.

3.1 Test polynomial factorization

Hereafter, we examine the conditions a function F should verify and we introduce a "half-circle" factorization of the test polynomial.

Theorem 3.1. *Let $F : \mathbb{Z}_{2N} \to \mathbb{Z}_{2N}$ be a function to be evaluated in a bootstrapping procedure using relation* (1)*. Function F must satisfy relation $F(m+N) = -F(m)$ for $0 \le m < N$.*

Proof. Let $P(X)$ be a polynomial from $\mathbb{Z}_N[X]$. Multiplying it by X^N gives the initial polynomial with negated coefficients, i.e. $P(X) \cdot X^N \equiv -P(X) \in \mathbb{Z}_N[X]$. This is due to relation $X^N = -1$ defining cyclotomic polynomial $\Phi_{2N}(X)$, i.e. the negacyclic property of the ring $\mathbb{Z}_N[X]$. If we apply this observation to the left-hand side of equation (1) we have:

$$\mathsf{TV}_F(X) \cdot X^{(m+N)} \equiv -TV_F(X) \cdot X^m \mod \Phi_{2N}(X),\ 0 \leq m < N$$

Respectively, the right-hand side must satisfy the condition $F(m+N) = -F(m)$ for $0 \leq m < N$.

In what follows we restrict equation (1) to values of m belonging to $\mathbb{Z}_N$. In this way, the condition $F(m+N) = -F(m)$ is automatically verified.

Half-circle polynomial bootstrapping. Let TV_F be a test polynomial defined as $\mathsf{TV}_F = \sum_{i=0}^{N-1} t_i X^i$, where $t_0 = F(0)$ and $t_i = -F(N-i)$ for $1 \leq i < N$. Thus, TV_F equals to $F(0) - \sum_{i=1}^{N-1} F(i) \cdot X^{N-i}$. It is straightforward to see that the relation $\mathsf{TV}_F \cdot X^m = F(m) + R(X) \mod \Phi_{2N}(X)$ is satisfied for any $0 \leq m < N$.

The test polynomial TV_F must be factored into two polynomials such that the first one $\mathsf{TV}^{(0)}$ does not depend on the evaluated function F. We did not mentioned earlier but the factorization can be fractional. Let τ denote the least common multiple of the factorization such that $TV^{(0)}, TV_F^{(1)} \in \mathbb{Z}_N[X]$:

$$\tau \cdot TV^{(0)} \cdot TV_F^{(1)} \equiv TV_F \mod \Phi_{2N}(X)$$

We define the first-phase test polynomial as $TV^{(0)} = \sum_{i=0}^{N-1} X^i$ and $\tau = 1/2$. Let second-phase test polynomial be $TV_F^{(1)} = \sum_{i=0}^{N-1} t'_i \cdot X^i$. Polynomials $\mathsf{TV}^{(0)}$ and $\mathsf{TV}_F^{(1)}$ being factors of TV_F we have:

$$\sum_i t_i \cdot X^i \equiv 1/2 \cdot \sum_i t'_i \cdot X^i \cdot \sum_i X^i \mod \Phi_{2N}(X)$$

Using the fact that $X^N = -1$, we obtain the following system of linear equations with N unknowns $t'_i,\ 0 \leq i < N$:

$$\sum_{0 \leq i \leq k} t'_i - \sum_{k < i < N} t'_i = 2t_k,\ 0 \leq k < N \qquad (2)$$

Theorem 3.2. *The system of linear equation* (2) *admits an analytical solution given by:* $t'_0 = t_0 + t_{N-1}$ *and* $t'_k = t_k - t_{k-1}$ *for* $k \geq 1$.

Proof. Observe that two consecutive t_{k-1} and t_k differ only by t'_k element sign. Computing their difference, we have $2 \cdot (t_k - t_{k-1}) = \sum_{0 \leq i \leq k} t'_i - \sum_{k < i < N} t'_i - \sum_{0 \leq i \leq k-1} t'_i + \sum_{k-1 < i < N} t'_i = 2t'_k$. The case for t'_0 is equivalently proved except that for t_0 and t_{N-1} only the sign of t'_0 is the same.

Property 1. Suppose that function F has the same output value for consecutive points $N-k$ and $N-k+1$, thus $F(N-k) = F(N-k+1)$. Observe that $t'_k = t_k - t_{k-1} = -F(N-k) - F(N-k+1) = 0$. We deduce that the second-phase test polynomial coefficient t'_k is zero in this case. More generally, this test polynomial has exactly s non-zero coefficients where s is the number of transitions of function F, i.e. $s = |\{F(k) \neq F(k+1) : 0 \leq k < N\}|$.

The test polynomial factorization introduced earlier can be graphically interpreted as follows:

1. The first-phase test polynomial divides the torus in two parts. The bootstrapping with test polynomial $\tau \cdot TV^{(0)}$ returns $+\tau$ for first half-circle $[0, 1/2[$ of torus and $-\tau$ for the other part.
2. The second-phase test polynomial builds a linear combination of such half-circles, thus the half-circles described in step 1 are rotated by X^i and scaled by t'_i.

Example. We give in Figure 2 an example over $\mathbb{T}$ of the previously explained procedure. We ignore the coefficient τ in this illustration. On the top torus circle are denoted values returned by first-phase test polynomial, i.e. test polynomial values projected on torus circle. The second-phase test polynomial has 3 terms and is equal to $t'_a X^a + t'_b X^b + t'_c X^c$. The 3 bottom torus circles denote the linear mapping performed by each monomial of second-phase test polynomial. Summing up these terms gives a torus circle values illustrated on the rightmost part of figure. Observe the negacyclic property of cyclotomic polynomial $X^N + 1$ on the torus circles from the fact that symmetric output values are negated.

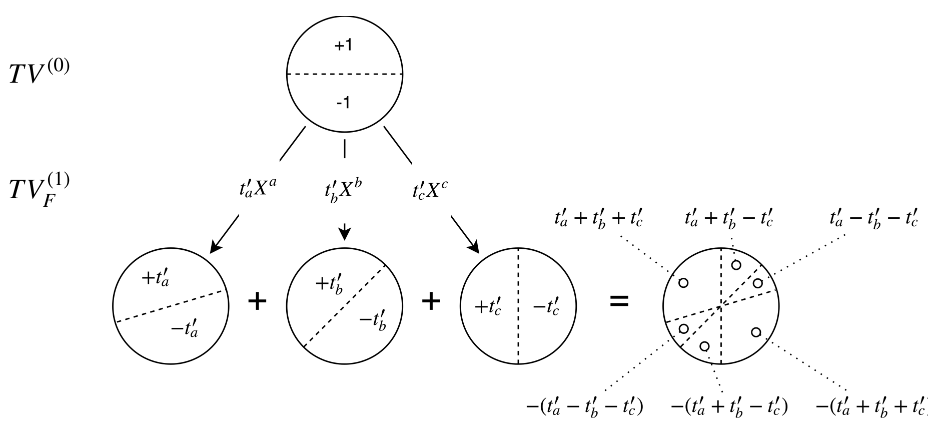

Fig. 2. Illustration of the high-level strategy for the multi-value bootstrapping

Function evaluation with rounding Let f be a function from $\mathbb{Z}_t$ to $\mathbb{Z}_q$ for $t < 2N$ and $q \leq 2N$. Let r be a rounding function which takes as input a message from

$\mathbb{Z}_{2N}$ and outputs a rounded message belonging to $\mathbb{Z}_t$. Function r is defined as $r(m) = \lfloor m \cdot {}^t\!/_{2N} \rceil$. This function corresponds to the rounding performed on TLWE ciphertext phase in order to obtain the plaintext message.

Test polynomial $TV_{f \circ r} = \sum_i t_i$ for the composed function $f \circ r$ is defined as: $t_0 = f \circ r(0)$ and $t_k = -f \circ r(N-k)$ for $1 \leq k < N$. Building the system of linear equation (2) and using explicit solution given in Theorem 3.2 we can deduce the coefficients for second-phase test polynomial.

Proposition 1 (Second-phase test polynomial norm). *Let f be a function from $\mathbb{Z}_s$ to $\mathbb{Z}_q$ and let $TV^{(1)}_{f \circ r}$ be the corresponding second-phase test polynomial. The squared norm of this polynomial is given by:*

$$\left\| TV^{(1)}_{f \circ r} \right\|_2^2 \leq s \cdot (q-1)^2$$

Proof. (Number of non-zero coefficients) From the definition of the rounding function r we have $r(k) = l$ for any k such that $l \cdot {}^{2N}\!/_t \leq k < (l+1) \cdot {}^{2N}\!/_t$. Without loss of generality we suppose here that t divides $2N$. Composed function $f \circ r$, denoted by F, has the same output value for ${}^{2N}\!/_t$ consecutive input messages from $\mathbb{Z}_{2N}$, i.e. $F(k) = f \circ r(k) = f(l)$ for $l \cdot {}^{2N}\!/_t \leq k < (l+1) \cdot {}^{2N}\!/_t$. Using Property 1 we deduce that the $TV^{(1)}_{f \circ r}$ polynomial is sparse and has exactly s non-zero coefficients. Let S, $|S| = s$, be the set of indexes of non-zero coefficients, we have $TV^{(1)}_{f \circ r} = \sum_{i \in S} t'_i X^i$.

(Coefficient range) Each non-zero coefficient t'_i, $i \in S$, is defined as the difference between consecutive output values of function $f \circ r$, or equivalently function f. Refer to Theorem 3.2 and $TV_{f \circ r}$ definition. We have $(t'_i)^2 \leq (f(k) - f(k'))^2$ for any $k, k' \in \mathbb{Z}_t$. As function f is defined over $\mathbb{Z}_q$ relation $0 \leq f(.) \leq q-1$ is verified. We deduce $(t'_i)^2 \leq (q-1)^2$

Combining these results we obtain the bound expression:

$$\left\| TV^{(1)}_{f \circ r} \right\|_2^2 = \left\| \sum_{i \in S} t'_i X^i \right\|_2^2 = \sum_{i \in S} (t'_i)^2 \leq s \cdot (q-1)^2$$

3.2 Optimized multi-value bootstrapping

In this subsection we focus on multi-value bootstrapping procedure for Torus FHE where the $2N$-th cyclotomic polynomial $X^N + 1$ defines TRLWE samples. We assume that first and second phase test polynomials, $TV^{(0)}, TV^{(1)}_F \in \mathbb{Z}_N[X]$, together with scale factor τ verifying condition (3) are given.

$$\tau \cdot TV^{(0)}(X) \cdot X^m \cdot TV^{(0)}_F(X) \equiv F(m) + R(X) \mod \Phi_{2N}(X) \qquad (3)$$

Algorithm 4 illustrates the steps of optimized bootstrapping procedure using split test polynomials. It takes as input a ciphertext encrypting a message ${}^m\!/_{2N}$, $m \in \mathbb{Z}_{2N}$, and outputs a ciphertext encrypting $F(m) \in \mathbb{Z}_{2N}$. Test polynomial $TV^{(0)}$ belongs to $\mathbb{Z}_N[X]$. It is mapped to $\mathbb{T}_N[X]$ by multiplication with ${}^1\!/_{2N} \in \mathbb{T}$

and with scale factor τ (algorithm step 2). There is not need to map second-phase test polynomial to $\mathbb{T}_N[X]$ because in step 4 a linear transformation of ACC by $TV_F^{(1)}$ is performed.

Algorithm 4 Multi-value bootstrapping algorithm

Input: A TLWE sample $\underline{\mathbf{c}} = (\underline{\mathfrak{a}}, \underline{\mathfrak{b}}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}},\underline{\eta}}(\mu)$ where $\mu = {}^{m}/_{2N}$, $m \in \mathbb{Z}_{2N}$

Input: First, second phase test polynomials $TV^{(0)}, TV_F^{(1)} \in \mathbb{Z}_N[X]$ and scale factor τ

Input: A bootstrapping key $\mathsf{BK}_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\alpha} = \left(\mathsf{BK}_i \in \mathsf{TRGSW}_{\bar{K},\alpha}(\underline{\mathfrak{K}}_i)\right)_{i \in [\![1,n]\!]}$ where $\bar{K}$ is the TRLWE interpretation of $\bar{\mathfrak{K}}$.

Output: A TLWE sample $\bar{\mathbf{c}} \in \mathsf{TLWE}_{\bar{\mathfrak{K}},\bar{\eta}}({}^{F(m)}/_{2N})$

1: Let $b = \lfloor 2N\underline{\mathfrak{b}} \rceil$ and $a_i = \lfloor 2N\underline{\mathfrak{a}_i} \rceil \in \mathbb{Z}_{2N}$ for each $i \in [\![1, n]\!]$
2: Let $v \leftarrow \mathsf{TV}^{(0)} \cdot {}^{1}/_{2N} \cdot \tau \in \mathbb{T}_N[X]$
3: $\mathsf{ACC} \leftarrow \mathsf{BlindRotate}((\mathbf{0}, v), (a_1, \ldots, a_n, b), (\mathsf{BK}_1, \ldots, \mathsf{BK}_n))$
4: $\mathsf{ACC} \leftarrow \mathsf{TV}_F^{(1)} \cdot \mathsf{ACC}$
5: Return $\bar{\mathbf{c}} = \mathsf{SampleExtract}(\mathsf{ACC})$

Theorem 3.3. *Given a* TLWE *input ciphertext* $\underline{\mathbf{c}}$ *of message* $\mu = {}^{m}/_{2N}$, $m \in \mathbb{Z}_{2N}$, *first-phase* $\mathsf{TV}^{(0)} \in \mathbb{Z}_N[X]$, *second-phase* $\mathsf{TV}_F^{(1)} \in \mathbb{Z}_N[X]$ *test polynomials, factorization factor* τ *verifying condition* (3) *and a valid bootstrapping key* $\mathrm{BK}_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\alpha} = (\mathsf{BK}_i)_{i \in [\![1,n]\!]}$, *Algorithm 4 outputs a valid* TLWE *ciphertext* $\bar{\mathbf{c}}$ *of message* ${}^{F(m)}/_{2N}$ *with error distribution variance verifying:* $\mathit{Var}(\mathit{Err}(\bar{\mathbf{c}})) \leq \left\|\mathsf{TV}_F^{(1)}\right\|_2^2 \left(n(k+1)\ell N\beta^2\vartheta_{\mathsf{BK}} + n(1+kN)\epsilon^2\right)$ *where* ϑ_{BK} *is the variance of bootstrapping key* $\mathit{Var}(\mathit{Err}(\mathsf{BK}_{\underline{\mathfrak{K}} \leftarrow \bar{\mathfrak{K}},\alpha})) = \alpha^2$.

Proof. (Correctness) The first 3 lines of Algorithm 4 compute a TRLWE ciphertext of message $X^{b - a\underline{\mathfrak{K}}} \cdot TV^{(0)} \cdot {}^{1}/_{2N} \cdot \tau$. Line 4 applies a linear transformation to it and message ${}^{\tau}/_{2N} \cdot X^{b - a\underline{\mathfrak{K}}} \cdot TV^{(0)} \cdot TV_F^{(1)}$ is obtained. Input message μ is a multiple of ${}^{1}/_{2N}$ on the torus so we have $b - a\underline{\mathfrak{K}} = \mu \cdot 2N$. Recall that $\tau \cdot TV^{(0)} \cdot TV_F^{(1)} \cdot X^m \equiv F(m) + \ldots$ for any $m \in \mathbb{Z}_{2N}$ and $m = \mu \cdot 2N$. Thus, ACC at line 5 contains an encryption of a polynomial whose zero-degree coefficient is ${}^{F(m)}/_{2N}$. The $\mathsf{SampleExtract}$ function from the last line extracts from ACC a TLWE sample of message ${}^{F(m)}/_{2N}$.

(Error Analysis) The error analysis for this method follows from the error analysis of the TFHE gate bootstrapping. It adds one multiplication by a constant polynomial $TV_F^{(1)}$ and gives the following variation of error distribution: $\mathsf{Var}(\mathsf{Err}(\bar{\mathbf{c}})) \leq \left\|TV_F^{(1)}\right\|_2^2 \left(n(k+1)\ell N\beta^2\vartheta_{\mathsf{BK}} + n(1+kN)\epsilon^2\right)$.

Theorem 3.4. *Under the same hypothesis as in Theorems 2.8 and 3.3, when given a correct input ciphertext* $\underline{\mathbf{c}}$ *of message* μ, $m = \mu \cdot 2N \in \mathbb{Z}_{2N}$, *the multi-value bootstrapping followed by a key-switching outputs a ciphertext* $\bar{\mathbf{c}}$ *of message*

$^{F(m)}/_{2N}$ *with error distribution variance:*

$$\mathit{Var}(\mathit{Err}(\overline{\boldsymbol{c}})) \leq \left\| \mathsf{TV}_F^{(1)} \right\|_2^2 \left(n(k+1)\ell N \beta^2 \vartheta_{\mathsf{BK}} + n(1+kN)\epsilon^2 \right) + n t \vartheta_{\mathsf{KS}}^2 + n 2^{-2(t+1)} \quad (4)$$

where ϑ_{BK} *and* ϑ_{KS} *are respectively the variances of bootstrapping and key-switching keys error distributions.*

Multi-output version In many cases one needs to evaluate several functions over the same encrypted message. The naive way is to execute bootstrapping Algorithm 4 several times for each function. Remark that for equal first-phase test polynomials $TV^{(0)}$ algorithm 4 performs the same computations up to line 3. Thus, until second-phase test polynomial integration into the accumulator. By repeating steps 4-5 for several second-phase test polynomials $\mathsf{TV}_{F_1}^{(1)}, \ldots, \mathsf{TV}_{F_q}^{(1)}$ the bootstrapping algorithm outputs encryptions of messages $F_1(m), \ldots, F_q(m)$. Figure 3 is a schematic view of the bootstrapping procedure which evaluates several functions over same input message.

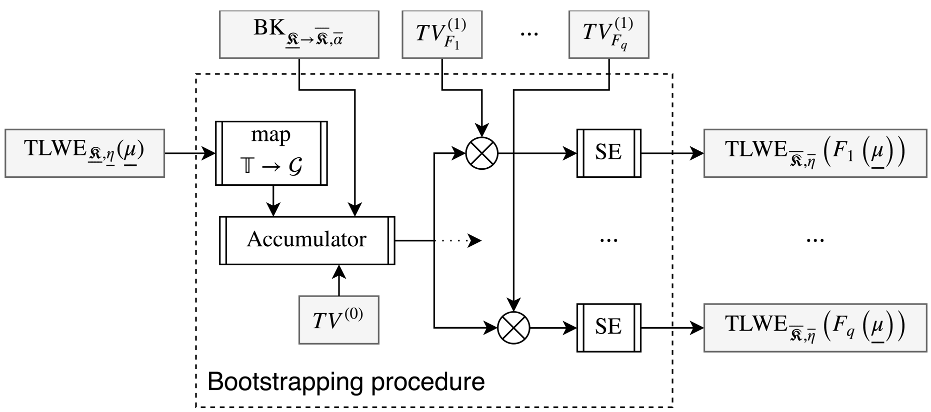

Fig. 3. Multiple output multi-value bootstrapping overview. Test polynomials $\mathsf{TV}_{F_1}^{(1)}, \ldots, \mathsf{TV}_{F_q}^{(1)}$ correspond to q functions evaluated over message $\underline{\mu}$ encrypted in the input ciphertext.

4 Homomorphic LUT

In this section, we show how to use the multi-value bootstrapping introduced earlier to homomorphically evaluate r-bit LUT functions over encrypted data. Afterwards, we describe how to compose them in a circuit and give implementation details for the case $r = 6$.

4.1 Homomorphic LUT evaluation

A boolean LUT is a function defined as $f : \mathbb{Z}_2^r \to \mathbb{Z}_2^q$. It takes an r-bit word as input and outputs a q-bit word. At first we focus on single-output LUTs, i.e. the case $q = 1$. Afterwards we show how to efficiently evaluate multi-output LUTs. It is straightforward to see an equivalent formulation for f over the ring of integers modulo 2^r, in particular $F : \mathbb{Z}_{2^r} \to \mathbb{Z}_2$. This formulation is obtained using the linear mapping $\phi(m_0, \ldots, m_{r-1}) = \sum_{j=0}^{r-1} m_j \cdot 2^j$ from $\mathbb{Z}_2^r$ to $\mathbb{Z}_{2^r}$. We have $F \circ \phi(m_0, \ldots, m_{r-1}) \equiv f(m_0, \ldots, m_{r-1})$ for any $(m_0, \ldots, m_{r-1}) \in \mathbb{Z}_2^r$.

The multi-value bootstrapping is used to evaluate LUT function F. The bootstrapping procedure is instantiated as follows. We encode integers over the torus as multiples of $1/2^{r+1}$. Only the first half-circle of torus is used for input and output message spaces. In this way any function can be evaluated using bootstrapping procedure, refer to restrictions from Theorem 3.1. Full message space is used for the input $j/2^{r+1}$ for $j \in \mathbb{Z}_{2^r}$ and only the first 2 elements are used for the output messages $j/2^{r+1}$ for $j \in \mathbb{Z}_2$. Test polynomial factorization described in previous section is used. Recall, the first-phase test polynomial $\mathsf{TV}^{(0)}$ is $\sum_i X^i$ and scaling factor is $\tau = 1/2$. The second-phase test polynomial is computed using Theorem 3.2 for LUT function F composed with a rounding function. From Proposition 1 this test polynomial norm verifies relation $\left\| TV^{(1)}_{F \circ r} \right\|_2^2 \leq 2^r$.

4.2 LUT circuits

A naive solution for multi-output LUT evaluation is to map $\mathbb{Z}_2^q$ to $\mathbb{Z}_{2^q}$. Doing so, we would be able evaluate functions $F : \mathbb{Z}_{2^r} \to \mathbb{Z}_{2^q}$ where $q \leq r$. The drawback of this method appears when we need to compose LUTs into a circuit and evaluate it. A reverse mapping from $\mathbb{Z}_{2^q}$ to $\mathbb{Z}_2^q$ is indeed needed. It will be an overkill to use another function to extract bits from $\mathbb{Z}_{2^q}$ messages, because it implies to use another multi-value bootstrapping.

Let $F^{(\ell)} : \mathbb{Z}_{2^r} \to \mathbb{Z}_2$ be a multi-value input function computing the ℓ-th output bit of LUT function $f : \mathbb{Z}_2^r \to \mathbb{Z}_2^q$, $\ell = 1, \ldots, q$. Each of these functions, $F^{(1)}, \ldots, F^{(q)}$, is evaluated as described in the last subsection. Note that the expensive blind rotate part from the bootstrapping Algorithm 4 is performed once. Only the multiplication by second-phase test vector and sample extract is done for each evaluated function.

Figure 4 illustrates intermediary steps for interfacing LUTs. Firstly, ciphertexts encrypting messages $m_1, \ldots, m_r \in \mathbb{B}$ obtained from several bootstrapping procedures are combined together into a multi-value message m using the linear transformation ϕ. Note that this transformation is performed in the output key space of the bootstrapping procedure under the secret key $\overline{\mathfrak{K}}$. Next, a key-switching procedure is performed and a ciphertext of the same message m under the secret $\underline{\mathfrak{K}}$ is obtained. This ciphertext is fed into the next bootstrapping and the process can be repeated.

It is possible to reorder the linear mapping evaluation and the key-switching, i.e. perform key-switching directly after the bootstrapping and evaluate the lin-

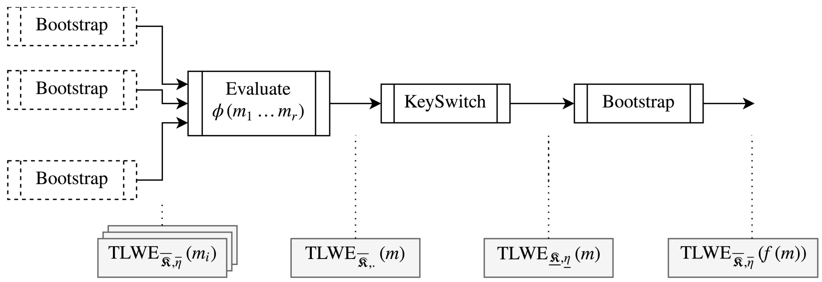

Fig. 4. LUT composition into circuits. On top are shown executed algorithms and at the bottom obtained ciphertexts.

ear mapping afterwards. Besides the fact that r times more key-switching procedures are performed the noise increase will also be larger. Actually, the linear map evaluation noise increase is multiplicative compared to the additive key-switching noise. In the next subsection, we describe implementation in more details.

4.3 Implementation details and performance

We implement the previous method for $r = 6$. We take $n = 2^9, k = 1, h = 63$ (TLWE key non-zero coefficient count) and $B_g = 2^6$. Let $\mathcal{A}_{\text{mv-boot}}$ be the amplitude of the output of the bootstrapping and $\mathcal{A}_{\text{key-switching}}$ be the amplitude of the output part induced by the key-switching. We can derive a bound on $\mathcal{A}_{\text{mv-boot}}$ and on $\mathcal{A}_{\text{key-switching}}$ from their variance using Theorem 3.3 and Theorem 2.8 respectively. With ℓ equals to 2^3 and $N = 2^{14}$, we obtain bootstrapping key standard deviation $\leq 2^{-47}$. Now setting $t = 24$, and using the inequality from Theorem 2.8, we obtain that key-switching standard deviation $\leq {}^{1}/_{24 \cdot 2^{25}}$

We have implemented the multi-value bootstrapping technique proposed above on-top of the TFHE library [9] and a test implementation is available in the `torus_generic` branch. Several modifications were performed in order to support 64-bit precision torus. Approximate sample sizes are: TLWE 8kB, TRLWE 256kB and the TRGSW 4MB. As for the keys we have: multi-value bootstrapping key 2GB and the switching key 3GB. The key sizes can be reduced using a pseudo-random number generator as in [7]. Our experimental protocol consisted in: (i) a 6 bit multi-value message is encrypted, (ii) parameters (i.e. second-phase test polynomials) for several LUTs are generated randomly, (iii) the multi-value bootstrapping is executed on this encrypted message (several ciphertexts encrypting boolean messages are obtained), (iv) a weighted sum is used to build a new multi-value message ciphertext from 6 of the output boolean messages obtained previously, (v) finally a key-switching procedure is performed in order to regain the bootstrapping input parameter space.

We have executed the algorithms on a single core of an Intel Xeon E3-1240 processor running at 3.50GHz. The bootstrapping and switching keys are generated in approximatively 33 seconds. Multi-value bootstrapping on 6 bit words with 6 boolean outputs runs in $\approx$ 1.2 sec. with the bit combination plus key-switching phase and in under 1 sec. without the key-switching. For comparison the gate bootstrapping from TFHE library takes 15ms on the same machine. We did not observed a significant increase in the execution time when the number of LUT outputs augments. For example computing 128 different functions on the same input message increased the execution time only by 0.05 sec., almost for free ! We shall note that the combination and key-switching was performed a single time in this last experiment. To estimate the security, we ran the estimator from [18] which includes the recent attacks on small LWE secrets [1]. We found that our instances achieve about 100 bits of security which is comparable to the concrete security level of the 6-to-6 LUT implementation of [14].

5 Further applications

5.1 Improved circuit bootstrapping

Another application concerns the optimization of the circuit bootstrapping from [16, Sec. 4.1] which allows to compose circuits in a leveled mode by turning a TLWE sample into a TRGSW sample. The first step of the *circuit bootstrapping* consists to make ℓ calls to the TFHE gate bootstrapping on the same TLWE input sample. Here each bootstrapping call is associated to a different test polynomial. We can apply the multi-value bootstrapping (in particular the multi-output feature) to optimize this step: since the LWE input sample is the same, the idea is to perform the BlindRotate Algorithm 1 only once for the ℓ bootstrapping calls, to adapt the output using corresponding test polynomials $\mathsf{TV}_F^{(1)}$ as in Sub-section 3.2 and obtain the ℓ desired outputs. This allows to save a factor ℓ in one of the circuit bootstrapping phases. We let the implementation and the integration of this optimization in the TFHE project [9] for a future work.

5.2 Homomorphic evaluation of neural networks

Our multi-value bootstrapping can also be used to homomorphically evaluate a neural network. We give a very brief overview on how it works and let the specification of the model and the detailed protocol analysis for another work. Assume neurons $x_1, \ldots, x_p$ inputs and output y are encrypted as TLWE ciphertexts. The computational neuron network functionality is defined by two functions, a linear function $f : \mathbb{T}^p \mapsto \mathbb{T}$ and an activation function $g : \mathbb{T} \mapsto \mathbb{T}$. The result is a TLWE sample of $y = g(f(x_1, \ldots, x_p))$. Function f is usually implemented as an inner-product. We can compute the inner-product between p neuron inputs and a fixed weight vector using a functional key-switch, and afterwards, extract the TLWE encryption from key-switch output (a TRLWE sample). Note that the public functional key-switch allows to compute up to N inner-products. Thus, using a single key-switch procedure we can compute all the linear functions of a whole neural network layer! Afterwards, using our multi-value bootstrapping, we compute a TLWE sample of $g(.)$. Here, g is not an arbitrary function. Usually a threshold function is used. In this particular case, the multi-value bootstrapping can be more efficiently instantiated than for arbitrary functions.

Conclusion

In this paper, we introduced a bootstrapping procedure based on TFHE scheme with split test polynomials. This bootstrapping procedure can be used to evaluate multi-value functions and to increase the evaluation efficiency of multi-output functions. We note that this method (the test polynomial split trick) can be easily adapted to other FHEW-based bootstrapping algorithms. We also studied several associated tools and the applicability of the multi-value bootstrapping to execute arbitrary LUT functions on encrypted data. The evaluation of a 6 input, 1 output LUT takes under 1.2 seconds. The evaluation of additional outputs on the same input comes at virtually no cost. We also introduced some ideas on how to optimize the circuit bootstrapping used to compose circuits in a leveled mode and how to evaluate a neural network system.

References

1. M. R. Albrecht. On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. In *In Proc. of EUROCRYPT, volume 10211 of LNCS*, pages 103–129. Springer, 2017.
2. J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In *Crypto*, pages 297–314, 2014.
3. J. F. Biasse and L. Ruiz. FHEW with efficient multibit bootstrapping. In *Proc. of Latincrypt 2015*, LNCS 9230, pages 119–135. Springer-Verlag, 2015.
4. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.
5. Z. Brakerski and R. Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In *Crypto'2016*, volume 9814, pages 190–213, 2016.
6. J. H. Cheon and D. Stehlé. On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. In *In Proc. of EUROCRYPT, volume 9057 of LNCS*, pages 513–536. Springer, 2015.
7. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Asiacrypt 2016, Part I 22*, pages 3–33. Springer, 2016.
8. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. A homomorphic lwe based e-voting scheme. In *Post-Quantum Cryptography*, pages 245–265. Springer, 2016.
9. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast fully homomorphic encryption library. https://tfhe.github.io/tfhe/, August 2016.
10. L. Ducas and D. Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Eurocrypt*, pages 617–640, 2015.
11. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. https://eprint.iacr.org/2012/144, 2012.
12. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

13. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Crypto*, pages 75–92, 2013.
14. L. D. Guillaume Bonnoron and M. Fillinger. Large FHE gates from tensored homomorphic accumulator. In *Proc. of Africacrypt 2018*, LNCS 10831, pages 217–251. Springer-Verlag, 2018.
15. S. Halevi and I. V. Shoup. Helib - an implementation of homomorphic encryption. https://github.com/shaih/HElib/, September 2014.
16. M. G. Ilaria Chillotti, Nicolas Gama and M. Izabachène. Improving tfhe: faster packed homomorphic operations and efficient circuit bootstrapping. In *Proc. of Asicacrypt 2017*, LNCS 10624, pages 377–408. Springer-Verlag, 2017.
17. T. Lepoint. FV-NFLlib: Library implementing the Fan-Vercauteren homomorphic encryption scheme. https://github.com/CryptoExperts/FV-NFLlib, May 2016.
18. R. P. Martin Albrecht and S. Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology, ePrint Archive 2015/046*, 2015.
19. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
20. SEAL. Simple encrypted arithmetic library. https://sealcrypto.codeplex.com/.
21. C. P. Vadim Lyubashevsky and O. Regev. On ideal lattices and learning with errors over rings. In *In Proc. of EUROCRYPT, volume 6110 of LNCS*, pages 1–23. Springer, 2010.
22. C. P. Z. Brakerski, A. Langlois, O. Regev, and D.Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584, 2013.