Trend of Bootstrapping from 2009 to 2016

Paulin Boale Bomolo, Eugene Mbuyi Mukendi, Simon Ntumba Badibanga
Department of Computer Sciences and Mathematics
University of Kinshasa, Kinshasa, Democratic Republic of Congo

***Abstract*—The pedestal of fully homomorphic encryption is bootstrapping which allows unlimited processing on encrypted data. This technique is a bottleneck in the practicability of homomorphic encryption. From 2009 to 2016, the execution time of bootstrapping decreased from several hours to a few thousandths of a second for processing a logic gate on two encrypted bits. This paper makes a comparative study of the evolution of bootstrapping during the period. An implementation of multiplication on 16-bit integers on an Intel i7 architecture through three schemes whose libraries are respectively DGHV, FHEW and TFHE makes it possible to corroborate the trend that to date the best bootstrapping on bits is that of the TFHE which executes this processing in 29 seconds improving that of the FHEW 30 times despite the multiplication algorithm used.**

Keywords*—Bootstrapping; homomorphic encryption; binary multiplication; logic gates

I. Introduction

Encryption is said to be probabilistic if a plaintext message is encrypted in several ciphertexts. This feature is found by adding a random value during the encryption operation. It is said to be homomorphic if it allows performing processing on ciphertexts with corresponding results on plaintexts. If it performs only additions [1, 2], multiplications [3] or binary operations [4] it is called partial homomorphic otherwise if it performs additions and multiplications in a limited number then it is somewhat homomorphic [5, 6, 7]. On the other hand, if it performs processing on unlimited number it is said to be fully homomorphic [5, 6, 7, 8, 9, 10, 11].

Fully homomorphic encryption was a breakthrough made by Gentry [6] in his thesis in response to the conjecture state in [12]. This breakthrough is based on the bootstrapping technique that evaluates its own decryption circuit to refresh noise in the ciphertexts thus allowing an unlimited number of processing in the encrypted domain. Initially, it requires the squashing technique and an additional security assumption to reduce the complexity of the decryption circuit [9, 10, 11, 13, 14]. With the advent of fully homomorphic encryption schemes based on the difficult problem of LWE [15] which belongs a low-complexity decryption algorithm, squashing was eliminated in bootstrapping [8, 9,10, 11,13, 14].

The removal of squashing allows in [9,10] to use a second homomorphic encryption scheme in the homomorphic evaluation in the decryption algorithm. This consideration improved performance of the homomorphic processing of gate NAND on two-bits encrypted in less than a second [9]. This processing was improved to 13 milliseconds by [10, 11].

N-bits arithmetic operations such as multiplication or addition can be built from the universal gate NAND. While knowing that an addition or multiplication operation performed on encrypted bits can respectively multiple or raise to the power the noise by the number of operations. We seek to know in this paper whether the performance of bootstrapping on a multiplication on two encrypted integers of 16 bits through an implementation carried out with three libraries of the comparative schemas that each marked a period in this trend is in the same order of processing as on the encrypted bits.

Roadmap. Section II presents the literature review of the encryption scheme from gentry's breakthrough to the period under review. Section III establishes the criteria for comparison through bootstrapping with a focus on the concepts behind them and presents a comparative study based on the concepts between the relevant algorithms of each period. Section IV shows three multiplication algorithms on two 16-bit integers and a shifter. Finally, section V extends the bootstrapping processing of said algorithms to integers of 16 to perform homomorphic multiplication. Discussions close this comparison.

II. Literature Review

In 2009, Gentry published the first homomorphic encryption scheme based on ideals lattices. This scheme is characterized by too large parameters and additional security assumptions. Two schemes improved respectively the reduction of the size of encrypted keys and messages and the removal of the additional security assumptions [16, 17, 18, 19]. The majority of schemes in this category are unusable in everyday applications.

To facilitate an understanding of gentry blueprint, an integer-based homomorphic encryption scheme was published in 2010[7]. It is based on the difficult assumptions of Approximate Greatest Common Divisor [20]. Several integer-based schemes have been proposed to improve the efficiency of DGHV. These improvements could be achieved based on different variants of the AGCD security assumptions to reduce the size of the public key in security parameter and the expansion of this schema in [21, 22, 23,24 ,25].

The hardness of implementing homomorphic encryption schemes based on hard problems mentioned above have steered the research towards another security problem. Thus, the first complete homomorphic encryption scheme based its security on the assumptions of LWE that removes squashing was presented by Brakerski and Vaikuntanathan [13]. Said scheme is based on two procedures which are the relinearization and the reduction of the module or dimension. Relinearization is a technique that reduces the size of ciphertexts from $n^2$to $n + 1$. It starts from a quadratic function

in a secret key $s$to a linear function in a secret key $t$dependent on s. Reducing the module or dimension naturally reduces the complexity of the decryption function and also reduces the size of the digits. Then, Brakerski and Vaikuntanathan also proposed a version of their schema based on the assumptions RLWE [14,26,27]. Many homomorphic schemes by levels or complete was introduced [8, 27]. Each new scheme brings techniques aimed essentially at reducing the size of the parameters and increasing the multiplicative depth. But relinearization is still a bottleneck for the majority of these schemes.

Of all these schemes, [8] stood out. It relies on assumptions of approximate vectors to perform a homomorphic operation of gate NAND on encrypted messages. Encrypted messages are square matrices, addition and homomorphic multiplication is addition and multiplication matrix respectively. The author in [8] removes relinearization which is an expensive technique used in other LWE schemes in favor of the eigenvector approximation technique. The author in [9] uses a variant of the [8] to improve its bootstrapping based on gate NAND in less than a second. The author in [10, 11] introduces the external product in a variant of the [8] to reorganize bootstrapping of [9] and achieve 30 times better performance.

Our goal is to establish the criteria for comparing bootstrapping in three schemes which are the DGHV scheme [7], the Ducas Micciancio [9] scheme and the scheme in [10, 11]. In addition, use the libraries that implement them to corroborate the trend of bootstrapping in a 16-bit binary multiplication.

III. BOOTSTRAPPING

It is a technique that was introduced by Gentry [6] to solve the open problem stated in 1978 by [12] which consists in carrying out the processing on the encrypted data. Before Gentry's breakthrough, the noise increases with the circuit depth to be evaluated. The consequence is that decryption fails. The solution to this concern for inefficiency was through the encryption technique to reduce noise in the encrypted message and a homomorphic evaluation of the decryption algorithm.

A. The reencryption [6]

1) Definition of reencryption: reencryption is a noted function $\mathrm{Rencrypt}_\varepsilon$ that converts a message encrypted under a key $\mathrm{p_{k1}}$ into another message encrypted under an another key $\mathrm{p_{k2}}$ without revealing any information about the private key $\mathrm{s_{k1}}$ or the plain text m it is clear that $\mathrm{Rencrypt}_\varepsilon(\mathrm{c}) = \mathrm{Encrypt}_\varepsilon(\mathrm{p_{k2}}, \mathrm{m})$ where $\mathrm{c} = \mathrm{Encrypt}_\varepsilon(\mathrm{p_{k2}}, \mathrm{m})$ [2].

2) Reencryption algorithm: A reencryption can be evaluated in the following steps:

Generate a key pair (s_{ka}, p_{ka}) and (s_{kb}, p_{kb}) respectively belonging to A and B.

Evaluate A reencrypting key A between B and as follows $r_k = Encrypt_\varepsilon(p_{kb}, s_{ka})$.

Calculate a ciphertext $c = Encrypt_\varepsilon(p_{ka}, m)$ where m is plaintext.

Redefine the decryption function $Decrypt_\varepsilon$ as follows $f_c(s_{ka}) = Decrypt_\varepsilon(s_{ka}, c)$.

Evaluate the reencryption as follows $Evaluate_\varepsilon(p_{kb}, f_c, r_k) = Encrypt_\varepsilon\big(p_{kb}, f_c(s_{ka})\big) = Encrypt_\varepsilon(p_{kb}, m)$.

Reencryption allows to A to designate B by giving it the ability to encrypt the plaintext that it has encrypted with another key. B is called proxy.

3) One-way reencryption: Given a pair of keys (s_{k1}, p_{k1}) and (s_{k2}, p_{k2}). A one-way reencryption is a conversion from $Encrypt_\varepsilon(m, p_{k1})$ to $Encrypt_\varepsilon(m, p_{k2})$ by the evaluation of $Evaluate_\varepsilon(p_{k2}, f_c, r_k)$ where $r_k = Encrypt_\varepsilon(p_{k2}, s_{k1})$, not the inverse $Encrypt_\varepsilon(m, p_{k1}) = Evaluate_\varepsilon(p_{k1}, f_c, r_k)$ where $Encrypt_\varepsilon(m, p_{k2})$ and $r_k = Encrypt_\varepsilon(p_{k1}, s_{k2})$. The reciprocal of this assertion is false.

B. Hard Problems of Homomorphic Encryption

1) The problem of learning with error: The Problem of Learning With Error (LWE) was introduced by Regev in 2005 [15]. The Ring version of this problem called RingLWE, was introduced by Lyubashevsky, Peikert and Regev in 2010[28]. All variants are widely used nowadays in the construction of lattices-based homomorphic encryption schemes.

a) The Regev problem: For a security setting λ, Either $\mathrm{n} = \mathrm{n}(\lambda)$an integer dimension, an integer $\mathrm{q} = \mathrm{q}(\lambda) \geq 2$, and a distribution $\chi = \chi(\lambda)$ under $\mathbb{Z}$. The hard problem of $\mathrm{LWE}_{\mathrm{n,q},\lambda}$ the is to distinguish two following distributions:

In the first distribution, the sample $(\overrightarrow{a_i}, b_i)$ drawn uniformly from $\mathbb{Z}_q^{n+1}$;

In a second distribution, draw $\vec{s} \leftarrow \mathbb{Z}_q^{n+1}$and $(\overrightarrow{a_i}, b_i) \in \mathbb{Z}_q^n$ then a sample by drawing uniformly $\overrightarrow{a_i} \leftarrow \mathbb{Z}_q^n$ and $e_i \leftarrow \chi$ respectively, and initializing $b = \langle \overrightarrow{a_i}, \vec{s}_i \rangle + e_i$. The assumptions of $LWE_{n,q,\chi}$ are such that the problem of $LWE_{n,q,\chi}$ is hard.

b) The RLWE problem: For a secret $s \in R_q$, the RLWE distribution under $R_q \times R$ is drawn by respectively a uniform and random $a \in R_q$ and $e \leftarrow \chi$, and gives the output expression $(a, b = \langle s, a \rangle) + e \bmod q$.

RLWE is said to be decisional if given m independent samples $(a_i, b_i) \in R_q \times R_q$ where each sample is distributed either $A_{s,\chi}$ for random and uniform $s \in R_q$ (fixed for all samples) or the uniform distribution, distinguish which is the case (with a significant probability).

c) The General Problem of the LWE (GLWE) [26]: For a security parameter λ, that is $f(x) = x^d + 1$, where $d = d(\lambda)$ is a power of 2. Let be two integers respectively the modulus $q = q(\lambda)$ and the dimension n, let $R = \mathbb{Z}[x]/f(x)$ and $R_q = R/qR$. Let be $\chi = \chi(\lambda)$ a distribution on R. The problem with GAAE is to distinguish between the following two distributions:

The first distribution is a uniform sample $(a_i, b_i) \in R_q^{n+1}$;

In the second uniformly drawn distribution $a_i \leftarrow R_q^n$, $s \leftarrow R_q^n$ and $e_i \leftarrow \chi$, the second distribution is the sample $(a_i, b_i) \in R_q^{n+1}$ where $b_i = \langle a_i, s \rangle + e_i$. The assumption of GLWE is that the GLWE problem is hard.

If $d = 1$ then the LWE problem is that of the LWE problem. If n = 1 then the GLWE problem is that of the RLWE problem.

The author in [8] based on assumptions of LWE, it constructs its schema with a plaintext space $\mathbb{Z}_4 = \{0, 1, 2, 3\}$, an encrypted message space $\mathbb{Z}_q$ with an error or noise $E < \frac{q}{16}$ where q is the modulus that determines the key space from which the secret key s is taken and n is the encrypted message dimension.

To encrypt a plaintext $m \in \mathbb{Z}_2 \subset \mathbb{Z}_4$, draw $a \leftarrow \mathbb{Z}_q^n$, $e \leftarrow \chi$ and output the ciphertext c as follows $LWE_s^{4/q}\left(m, \frac{q}{16}\right) = \left(a, a.s + \frac{2m}{4} + e\right) \in \mathbb{Z}_q^{n+1}$.

The authors in [10, 11] redefines the problem of LWE and RLWE on the real torus $T = \mathbb{R} \, mod \, 1$ and the torus of polynomials $T[X] = T[X] \, mod \, X^N + 1$ respectively. This redefinition produces three types of ciphertexts for this schema. It also generalized and improved the encryption scheme based on the [8] and several of its variants.

To encrypt a plaintext $m \in T$, pick a secret key $s \in \mathcal{B}^n = \mathbb{Z}_2^n$ and calculate $c = (a, b) \in T^{n+1}$ where $a \in T^n$ is a random mask, $b = a.s + \varphi$ and $\varphi = e + m$ where e is a parameter that is drawn in a Gaussian distribution.

To encrypt the plaintext $m \in T_N[X]$, draw a key $s \in \mathcal{B}_N[X]$ and calculate $c = (a, b) \in T_N[X]^2$ where a is a random mask and $b = s.a + e + m$ where $e \in T_N[X]$.

To encrypt the plaintext $m \in \mathbb{Z}_N[X]$, pick the secret key $s \in \mathcal{B}_N[X]$ as in the RLWE and calculate $c = Z + m.G_2 \in T_N[X]^{2l \times 2}$ where Z is a list of ciphertexts of type RLWE of 0 and G_2 is the matrix with $\begin{pmatrix} g & 0 \\ 0 & g \end{pmatrix} g^T = (2^{-1}, \dots\dots\dots\dots\dots\dots\dots, 2^{-l})$.

2) The problem of the Approximation of the Greatest Common Divisor (AGCD)[20, 29].

The AGCD's problem with the parameters (γ, η, ρ) is the problem of finding the secret integer p given several samples $x_i = pq_i + r_i$ of arbitrarily provided where:

The secret integer p has bits η;

The terms noises r_i are uniform samples from the interval $[-2^\rho + 1, 2^\rho - 1] \cap \mathbb{Z}$;

The terms q_i are uniform samples of $[0, 2^{\gamma-\eta}] \cap \mathbb{Z}$.

[7] is the first known scheme applying the AGCD problem in cryptography to produce a homomorphic encryption scheme. In its symmetric version, it encrypts the plaintext $m \in \{0, 1\}$, two random integers are drawn uniformly to evaluate the encrypted message as follows $c = pq + 2r + m$. In other words, a sample of AGCD is calculated by adding the even noise $2r$ to the product pq which is added to m.

C. Bootstrapping [6]

1) Fundamental properties: In Gentry construction, bootstrapping is based on three fundamental properties that belong a partial or somewhat homomorphic encryption scheme that make it fully homomorphic encryption. These properties are listed and noticed below:

The complexity of the decryption algorithm is greater than that of the circuits to be evaluated. Given d the maximum degree of the decryption algorithm $Decrypt_\mathcal{E}$ and p the maximum degree of the function or polynomial to be evaluated by scheme. If $d < p$ then the decryption algorithm $Decrypt_\varepsilon$ is been useful in homomorphic evaluations. If $d > p$ then the complexity of this algorithm is reduced to $Decrypt'_\varepsilon$ for homomorphic evaluations hence $f_c(s_k) = Decrypt'_\varepsilon(s_k, c)$ where $c = Encrypt_\varepsilon(p_k, m)$.

Bootstrappability is a critical property of an encryption scheme that allows you to homomorphically evaluate your own decryption algorithm under an encrypted decryption key. Given an encryption scheme $\mathcal{E}$, $\mathcal{E}$ is said to be bootstrappable if $Evaluate_\mathcal{E} = (p_{k1}, f_c, e_k) = Encrypt_\mathcal{E}(p_{k1}, m)$ where $f_c(s_k) = Decrypt_\mathcal{E}(c, s_k)$, $c = Encrypt_\mathcal{E}(m, p_k)$ and $e_k = Encrypt_\mathcal{E}(p_{k1}, s_k)$ it is obvious that $\mathcal{E}$ evaluates homomorphically its decryption algorithm.

Circular security is a property that an asymmetric (symmetric) encryption scheme has to encrypt one's private key securely (secretly) by its corresponding public (secret) key. A homomorphic encryption scheme $\mathcal{E}$ has the circular security property if for a couple of given keys, (s_k, p_k) the bootstrapping key is evaluated as follows $e_k = Encrypt_\mathcal{E}(s_k, p_k)$: it is obvious that the private key is securely encrypted by its public key.

2) Definition of bootstrapping: Bootstrapping is a technique for reducing noise in the ciphertext c and getting noise b' in a refreshed ciphertext c' such as $b' < b$ where $b' \supset Encrypt_\varepsilon(p_k, m) \leftarrow Evaluate_\varepsilon(p_k, f_c, e_k)$ and b is the original noise in the ciphertext c by the homomorphic evaluation its own decryption circuit $f_c(s_k) = Decrypt_\varepsilon(s_k, c)$ on a decryption key called bootstrapping key$e_k = Encrypt_\varepsilon(p_k, s_k)$.

3) Bootstrapping algorithm: Given two pairs of keys (p_{k1}, s_{k1}) and (p_{k2}, s_{k2}) generated by a homomorphic encryption scheme ε.

Let be two ciphertexts c_1 and c_2 evaluate as follows: $c_1 = Encrypt_\varepsilon(p_{k1}, m_1)$ and $c_2 = Encrypt_\varepsilon(p_{k1}, m_2)$ where $m_1 \, and \, m_2$ are plaintexts.

The bootstrapping key e_k is calculated as follows $e_k = Encrypt_\varepsilon(p_{k2}, s_{k1})$. And the decryption function $Decrypt_\varepsilon$ is redefined in the following way $f_{c_1,c_2}(s_k) = NONET\big(Decrypt_\varepsilon(s_k, c_1), Decrypt_\varepsilon(s_k, c_2)\big)$ where is the private key s_k.

A homomorphic evaluation of f_{c_1,c_2} on c_1 *and* c_2 is carried out as follows: $Evaluate_{\varepsilon}(p_{k2}, f_{c_1,c_2}, e_k) = Encrypt_{\varepsilon}\left(p_{k2}, NONET\left(Decrypt_{\varepsilon}(s_{k1}, c_1), Decrypt_{\varepsilon}(s_{k1}, c_2)\right)\right) = Encrypt_{\varepsilon}(p_{k2}, NONET(m_1, m_2)) = NONET(Encrypt_{\varepsilon}(p_{k2}, m_1), Encrypt_{\varepsilon}(pk_2, m_2)) = NONET(c_1', c_2')$ where c_1' *and* c_2' are refreshed ciphertexts of c_1 *and* $c_2$whose noise $b' \ll b$.

4) Squashing: Squashing is a procedure that consists of expressing the decryption algorithm $Decrypt_{\varepsilon}$ into a polynomial or function $p_c(s_k)$ whose variables are the ciphertext c and the secret key sk. $p_c(s_k)$ is equivalent to a shallow circuit.

In [3], the decryption algorithm is expressed by the function $c^d \bmod N$. The complexity of the operation of exponentiation does not make it possible to rewrite this function into an equivalent function of low degree.

In the [7], the decryption algorithm is expressed by the expression $c \bmod p \bmod 2$ (1) which is not a low complexity. To do this, it is transformed into a circuit of expression $[c]_2 \oplus \left[\left\lfloor c.\left({}^{1}/_{p}\right)\right\rfloor\right]_2$ (2). ${}^{1}/_{p}$ is replaced in the evaluation (2) by the expression $\sum_{i=1}^{\Theta} s_i z_i$ which represents the sum of the subsets where $s_i = {}^{u_i}/_{2^{\kappa}}$. Evaluation (1) becomes $\left[c - \sum_{i=1}^{\Theta} s_i z_i\right]_2$ (3). (3) is the equivalent function of (1). (3) is an expression that has a low complexity.

5) Concept of Bootstrapping from 2015 [8 9, 10, 11]: Bootstrapping of scheme based on the problem of assumptions of LWE and its variants removes squashing. The decryption algorithm has a complexity that allows it to be evaluated homomorphically in the reencryption. This reencryption is carried out by a homomorphic accumulator which makes it possible to refresh the encrypted message into an equivalent encrypted message containing a small noise.

A homomorphic accumulator is a quadruplet of algorithms $Encrypt_{\varepsilon}$, Init, Incr and msbExtract. $Encrypt_{\varepsilon}$ is an encryption scheme that uses a key and is different from the first. It is called an internal scheme.

Init is the algorithm that initializes the contents of the accumulator. More briefly, this operation is written as follows: $ACC \leftarrow Encrypt_{\varepsilon}, (v)\ pour\ ACC \leftarrow Init(Encrypt_{\varepsilon}, (v))$.

Incr is the algorithm that allows you to add a value to the contents of the accumulator. This operation is written as follows: $ACC \overset{+}{\leftarrow} Encrypt_{\varepsilon}$ for $, (v) Incr\left(ACC,\ Encrypt_{\varepsilon}, (v)\right)$.

$msbExtract$ calculates with high probability from the contents of the homomorphic accumulator to produce a valid number. This operation is summarized by the expression $c \leftarrow msbExtract(ACC)$ with $c \in LWE_s^{t/q}\left(msb(v), e(l)\right)$ where e is the noise.

6) Type of bootstrapping: There are two types of bootstrapping that bootstrapping by squashing or by homomorphic accumulator.

A bootstrapping is said by squashing if a new security assumption is added in the reduction of the complexity of the decryption algorithm to ensure optimal security in the encryption scheme during the refresh of the noisy message.

Refreshing the ciphertext c with the addition of the assumption of the sum of subsets to re-encrypt c using the encrypted secret key $\sum_{i=1}^{\Theta} s_i z_i$ of $\frac{1}{p}$ which is used to obtain the ciphertext $c^* = \left[c - \sum_{i=1}^{\Theta} s_i z_i\right]_2$ [7].

A bootstrapping is said by homomorphic accumulator if a homomorphic accumulator is used to refresh a ciphertext in the reencryption operation.

a) Homomorphic accumulator in [9]: In [9], the homomorphic accumulator is based on the encryption scheme [8] defined under the assumptions of the Ring LWE. Let be a message m and the key $z \in \mathbb{Z}$, $Encrypt_{\varepsilon_z}(m)$ encrypts as described below:

Pick Randomly and uniformly the vector $a \in \mathcal{R}_Q^{2dg}$ and $e \in \mathcal{R}_Q^{2dg}$ into a Gaussian distribution χ of parameter ζ where $\mathcal{R}_Q^{2dg} = \mathbb{Z}_Q^{2dg}, N = 2^k$;

Calculate $Encrypt_{\varepsilon,z}(m) = [a, a.z + e] + uY^m G\ de\ \mathcal{R}_Q^{2dg\times 2}$ where m is encoded as the root of the unit $Y^m \in \mathcal{R} = \frac{\mathbb{Z}_N}{X^N+1}$ of where $N = 2^k$.

To upload the accumulator with the ciphertext $v \in \mathbb{Z}_q$, the function $Init(ACC \leftarrow v)$ uploads the content of accumulator with v as follows $ACC := uY^v G\ de\ \mathcal{R}_Q^{2dg\times 2}$;

To add an ciphertext to the contents of the accumulator, a decomposition of $u^{-1}.ACC$ in the base B_{dg} is performed as follows: $u^{-1}.ACC = \sum_{i=1}^{d_g} B_g^{i-1} D_i$ where the $D_i \in \mathcal{R}^{2dg\times 2}$ with the coefficients $\left\{\frac{1-B_g}{2}, \ldots\ldots, \frac{B_g-1}{2}\right\}$ and then perform $Incr(ACC \overset{+}{\leftarrow} C)$ where $ACC, C \in \mathcal{R}_Q^{2dg\times 2}$ to output $ACC := \left[D_1 \ldots\ldots D_{dg}\right]$.

Finally, use the msbExtract function with two entries that are a switch key $\mathfrak{R}$, a test vector $t = -\sum_{i=0}^{{}^{q}/_{2}-1} \vec{Y}_i$ to find $c \in LWE_s^{4/q}\left(m, \frac{q}{16}\right)$.

In [10], bootstrapping by accumulator is performed on the one-bit encrypted message $m \in \mathcal{B}$, $(a, b) \in T^n \times T = LWE_q^s(m, e)$ where $\mathcal{B} = \{0, 1\}$ and $e < \frac{1}{4}$ for valid decryption. Said message is first rounded to $\left(\bar{a}, \bar{b}\right) \in \mathbb{Z}_{2N}^n \times \mathbb{Z}_{2N}$ where $\bar{b} = \lceil 2Nb \rfloor$ and $\overline{a_i} = \lceil 2Na_i \rfloor$.

Given a test vector $testv = (1 + X + \cdots\ldots\ldots + X^{N-1}).X^{N/2}.u'$ where $u' = \frac{m}{4} \in T$, the result of the expression $X^{\bar{b}}.(0, testv)$ is loaded into ACC: $ACC \leftarrow (0, X^{-b}, testv)$. The evaluation of the

expression $[h + (X^{-\bar{a}_i} - 1)] \odot ACC$ update the content of ACC: $ACC \leftarrow X^{\bar{b}-\bar{a}s}.testv$.

An extraction is performed with the function $SampleExtract$ that receives as input the contents of: $ACCX^{\bar{b}-\bar{a}s}.testv$. It extracts the terms of said polynomial in a sample $msg((a', b'))$ where $(a', b') = (coefs(a''(X)), b'') \in T^n \times T$ where $coefs(a''(X))$ is the coefficient of the vector $a'' \in T_N[X]$ and $b_0'' \in T$ is the constant term of the polynomial $b'' \in T_N[X]$.

Key switching allows you to find a sample $TLWE(a, b) \in T^n \times T$ of the message $\frac{m}{2} \in T$ under the secret key s. It receives as input the result of the expression $msg(u) = u' + msg(SampleExtract(ACC))$.

7) Processing bootstrapping [9, 10]: There are two types of processing bootstrapping which are logic gate bootstrapping and logic circuit bootstrapping.

It is said that a homomorphic encryption scheme supports logic gate processing bootstrapping if a refresh is performed after each logic gate it is obvious that $Evaluate_\varepsilon(p_k, f_c, e_k)$ where f_c is a logic gate of the type AND, OR, NOT, ...

In [8], the homomorphic NAND gate is defined by $HomNAND: LWE_s^{4/q}(m_0, {}^{q}/_{16}) \times LWE_s^{4/q}(m_1, {}^{q}/_{16}) \rightarrow LWE_s^{2/q}(m_0 \overline{\wedge} m_1)$ where $m_0 \overline{\wedge} m_1 = 1 - m_0 m_1$ and $c_i = LWE_s^{4/q}(m_i, {}^{q}/_{16})$ with $i \in \{0, 1\}$. The refresh is performed on the result as follows: $LWE_s^{2/q}(m, {}^{q}/_{4}) \longrightarrow LWE_s^{4/q}(m, {}^{q}/_{16})$.

It is said that a homomorphic encryption scheme supports circuit processing bootstrapping if a refresh is performed after each logic circuit it is obvious that $Evaluate_\varepsilon(p_k, f_c, e_k)$ where f_c is a circuit that includes more than one logic gate of the type AND, OR, NOT.

In [8], let be a circuit for calculating the retention in an n-bit adder of two numbers a and b and an incoming retention c_0= 0, the expression (2). $c_i = (a_i \oplus c_{i-1}).(b_i \oplus c_{i-1}) \oplus c_{i-1}$ where $\oplus$ is XOR logic gate. From the Table I which is table of truth below of this expression a bootstrapping by circuit can be performed from a function majority with three variables noted $Maj(m_1, m_2 m_3)$ gives a value equal to 1 if the majority of bits is 1 otherwise 0.

Specifically, given three encrypted messages; Expression (1) can evaluate these three ciphertexts and produce a resulting ciphertext. Being calculated modulo 4, this makes it possible to homomorphically process the majority function described above. $c_1, c_2\ et\ c_3$.

This addition modulo 4 of the encrypted messages makes it possible to find the encrypted $Encrypt_\varepsilon(m)$, $m \in \{2,3\}$ with if majority is equal to 1 or if $m \in \{0,1\}$ the majority is equal to 0. An affine transform of $\frac{9q}{8}$ is performed to find the majority function in $\mathbb{Z}_4$. The circuit retained out of three is illustrated with the majority function noted maj as follows:

$$Maj\left(LWE_s^{4/q}(m_0, {}^{q}/_{16}), LWE_s^{4/q}(m_1, {}^{q}/_{16}), LWE_s^{4/q}(m_2, {}^{q}/_{16})\right) \rightarrow LWE_s^{2/q}(m, {}^{q}/_{4}).$$

The refresh is carried out on the result of the circuit as follows: $LWE_s^{2/q}(m, {}^{q}/_{4}) \longrightarrow LWE_s^{4/q}(m, {}^{q}/_{16})$.

8) Bootstrapping: Analysis and comparison of algorithms: Table II shows that TFHE bootstrapping performs better than bootstrapping performed and executed in the other two schemes. This fact is due to the removal of the decomposition step in any basis of the vector a of assumptions LWE [10, 11].

TABLE I. TRUTH TABLE OF THE SELECTED FUNCTION OF THE THREE BITS

a_i	b_i	c_{i-1}	$a_i \oplus c_{i-1}$(1)	$b_i \oplus c_{i-1}$(2)	1 ⊕ 2(3)	3 ⊕ c_{i-1}	Maj
1	1	1	0	0	0	1	3
1	1	0	1	1	1	1	2
1	0	1	0	1	0	1	2
1	0	0	1	0	0	0	1
0	1	1	1	0	0	1	2
0	1	0	0	1	0	0	1
0	0	1	1	1	1	0	1
0	0	0	0	0	0	0	0

TABLE II. ANALYSIS AND COMPARISON OF ALGORITHMS

Encryption scheme	Type of homomorphy	hard problem	Type of bootstrapping	Homomorphic operations	Squashing	Security parameter size (bits)	Complexity of the decryption algorithm	Bootstrapping key size	Bootstrapping execution time(s)
DGHV	fully	AGCD	By squashing	$+\ et\ \times$	Yes	72	great	NA	660
FHEW	fully	LWE RLWE	By accumulator	NAND	No	88	low	2.4 GB – 1 GB	0.63
TFHE	fully Leveled	TLWE	By accumulator	NAND, AND, ...	No	110	low	24 MB	0.052 0.0013

IV. Applications: Binary Multiplication

The operation of multiplying two integers is described in Fig. 1, for any calculation basis (binary, decimal, etc.) by the following two steps:

The calculation of partial products;

The sum of the partial products obtained.

The product of two numbers of n digits can be given by a number of $2n$ digits. In the binary system, the gate AND is used to generate the partial products $a_i b_i$ between each bit of the two multiplicandes. A binary addition is performed on each column of partial products.

				a_3	a_2	a_1	a_0
				b_3	b_2	b_1	b_0
				a_3b_0	a_2b_0	a_1b_0	a_0b_0
			a_3b_1	a_2b_1	a_1b_1	a_0b_1	
		a_3b_2	a_2b_2	a_1b_2	a_0b_2		
	a_3b_3	a_2b_3	a_1b_3	a_0b_3			
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Fig. 1. Example of Multiplying Two Numbers at 4 Bits.

A. The Classic Multiplication Algorithm[30]

Let a and b be two numbers of k bits, expressed as a basis:$\beta = 2$

$$a = (a_{n-1}a_{n-2} \dots \dots \dots \dots a_0) = \sum_{i=0}^{n-1} a_i \beta^i \quad (1)$$

$$b = (b_{n-1}b_{n-2} \dots \dots \dots \dots b_0) = \sum_{i=0}^{n-1} b_i \beta^i \quad (2)$$

Where the and a_i are b_i in the interval $[0, 1]$. The classical algorithm of multiplication of a and b consists in calculating partial products by multiplying the b_i of the multiplier by b the whole number a and then adding these partial products in order to obtain the final product p which is a number of $2n$ bits.

Note p_{ij} the pair (carry, Sum) obtained from the partial product $a_i b_j$. Fig. 1 illustrates the results p_{ij} of multiplication of a and b at 4 bits.

The last rank denotes the total sum of the partial products which is also the product $a\ by\ b$ represented by a number of $2k$ bits.

Algorithm 1: Classical Multiplication MC

Input: a, b
Output: p = ab
Initialize $p_i \coloneqq 0$ for $i = 0, 1, \dots \dots, 2n-1$
for $i = 0\ to\ n-1$
r
for $j \coloneqq 0\ to\ n-1$
$(r, s) = p_{i+j} + b_i a_j + r$
$p_{i+j} = s$
End For
$p_{i+n} \coloneqq r$
End for
Return $(p_{2n-1}p_{2n-2} \dots \dots \dots \dots p_0)$

This algorithm requires $O(n^2)$ bit-level operations to multiply two n bit encrypted numbers.

B. Horner's Algorithm

It was originally introduced to effectively evaluate the value of a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ for a given value α. It is based on the following rewrite:

$$p(x) = a_0 + a_1 x + \cdots \dots + a_n x^n \quad (3)$$

$$= a_0 + x\left(a_1 + x\left(a_2 + \cdots \dots \dots \dots . . + x\left(a_{n-1} + x(a_n)\right) \dots \dots .\right)\right) \quad (4)$$

The expressions below evaluate a polynomial $p(x)$ at a given point α by performing n multiplications and n additions to calculate $p(\alpha)$.

$$ab = a.\sum_{i=0}^{n-1} b_i 2^i = ab_0 2\left(ab_1 + 2\left(ab_2 + \cdots . . + 2\left(ab_{n-2} + 2(ab_{n-1})\right) \dots \dots \dots \dots\right)\right) \quad (5)$$

The equation below can be written in the following recursive form:

$$p_0 = 0$$

$$p_i = 2p_{i-1} + b_{i-1}a \quad (6)$$

From these equations, Horner's algorithm (2) for multiplying binary integers is written as follows:

Input: $a_0, a_1, \dots \dots \dots, a_{n-1}$ and $b_0, b_1, \dots \dots \dots \dots \dots \dots, b_{n-1}$
Output : $p = ab$
$p_0 \coloneqq 0$
For $i = n - 1\ to\ 0$
Do
$p_i \coloneqq 2p_{i-1} + b_{i-1}a$
End do
End for
Return p

This algorithm has the same complexity as the classical multiplication algorithm is $O(n^2)$.

C. Karatsuba's Algorithm

The Karatsuba algorithm is a recursive algorithm introduced by the Russian mathematician Karatsuba in 1962. This algorithm requires $O\left(n^{\log_2 3}\right)$ to multiply two numbers of n bits. Its complexity is reduced by method of the divide-and-conqueror which uses fewer multiplications than the classical algorithm.

Let a and b be two integers of n bits and $l = \lceil n/2 \rceil$. Karatsuba initially breaks down $a\ and\ b$ into two equal parts:

$$a = 2^l a_1 + a_0\ , b = 2^l b_1 + b_0 \quad (7)$$

Such as a_1 is the l high-weight bits of a and a_0 is the l low-weight bits of a. Note that the 2^lvalue thus constitutes the basis of the representation β.

1) Naïve recursion method: The naïve recursion method reduces the multiplication of a and b multiplication of their components $a_1, a_0, b_1\ et\ b_0$ including the size of the initial integers as shown in the following equation:

$$p = a.b = (2^l a_1 + a_0)(2^l b_1 + b_0)$$

$$= 2^{2l}(a_1b_1) + 2^l(a_1b_0 + a_0b_1) + a_0b_0$$

$$= 2^{2l}p_2 + 2^lp_1 + p_0 \quad (8)$$

Said formulation reveals that the multiplication of two numbers of k bits require 4 multiplications of $l = \frac{k}{2}$ bits. Its complexity is not far from that of a classical algorithm.

2) Karatsuba algorithm: Its algorithm improves the performance of equations in (1). By reducing the number of multiplications to three but adding four additional additions. A rearrangement of the terms of the product $p = a.b$ makes it possible to obtain:

$$p_0 = a_0b_0 (9)$$

$$p_1 = (a_0 + a_1)(b_0 + b_1) - p_0 - p_2 \quad (10)$$

$$p_2 = a_1b_1 \quad (11)$$

Of these equations, a remark is made of the presence of three multiplications, two bits n and $n + 1$ one bit. The karatsuba algorithm requires $O(n^{1.59})$ operations to give the product of two numbers.

Algorithm 3: Karatsuba multiplication.MK

Input: a, b, k
Output : $p = a.b$
If (is small) then k
Return: Call the classic algorithm.$MC(a, b)$
Finsi
$l := k/2$
$a_0 := a/2^l$
$a_1 := a \ mod \ 2^l$
$b_0 := b/2^l$
$b_1 := b \ mod \ 2^l$
$p_0 := MC(a_0, b_0)$
$p_1 := MC(a_1, b_1)$
$temp := MC(a_0 + a_1, b_0 + b_1)$
$p_1 := temp - p_0 - p_2$
Return $2^{2l}p_2 + 2^lp_1 + p_0$

The version of the algorithm that has been implemented in this paper is iterative. It performs operations on 8-bit encrypted integers.

D. The Shifter

A shifter is formed of n + 1 inputs d_1, d_2, ..., d_n, c and n outputs s_1, s_2, ..., s_n and operates an offset of 1 bit on the inputs if c = 1, it is an offset to the right and if c = 0 then it is an offset to the left.

Algorithm 4: shifting to left or right.

```
Input: a: n-bit encrypted integer, right or left Boolean: offset direction
Positions: Number of offset positions
b: encrypted integer shifted by offset over n bits of positions.
cx1, cx2 two null encrypted integers of n bits
i: integer counter
flag: A Boolean integer that determines the offset direction.
if flag = 0 then
 right = 1;
 left = no(right)
otherwise
 right = 0;
 left = no(right)
finsi
for i of 1 to positions
do
 for k from 0 to n – 1
do
 if k > 0 and k < n – 1 then
 cx1k = and(ak-1, left);
 cx2k = and(ak+1, right);
 bk = or(cx1k , cxk2);
 otherwise
 if k = 0 then
 bk = and(ak+1, right)
 finsi
 if k == n then
 bk = and(ak-1, left);
 finsi
 finish
end do
return b
```

Algorithm 4 has a complexity of $O(p \times n)$ where p is the number of offset positions and n is the bit size of the number to be shifted.

V. Implementation and Interpretation of Results

A. Implementation

The implementations were tested on the Intel® core™ i7-5500 CPU @2.4 GHZ processor of a laptop with a cache memory of 4019 kilobytes, a clock clock clock of 1100 MHZ and a volatile memory of 8 Gigabytes that supports extensions of the following instruction sets: MMX, SSE, SSE2, SSE4_1, SSE4_2, FMA, AVX and AVX2.

The DGHV code was implemented in Python with Sage and GMP (GNU Multi Precision). These two libraries provide machine compiled mathematical libraries that are fast in their executions. We have not been optimal to work with these tools in the implementation of multiplication.

The FHEW library that is written in C/C++ language. An optimization to quickly perform convolution was achieved by an implementation of the Fourier transform FFTW3 to process bootstrapping. Functions useful for performing multiplication have been added to the FHEW.cpp source file [31].

The TFHE library is written in C/C++ language and an optimization has been implemented for the fast processing of bootstrapping with the data parallelism of fused-multiply add and as an Advanced Vector eXtensions assembler through a SQLIOS fast Fourier transform parameterized in either AVX or FMA. Useful functions have been added to the cloud file.c and alice.c [32].

Synthesis and comparison:

In Table III, the columns represent the circuit type used in the implementation of multiplication operations and the type of logic gates. As for the rows, they represent the implementation of different types of multiplication. The intersection between the row and the column gives the number of circuits or gates implemented to achieve each type of multiplication.

TABLE III. Circuit used in each Type of Multiplication

	Adder	Subtractor	Shifter	And	Multiply	Weighting
Horner	1	0	1	1	0	N
Classic	2	0	0	1	0	N^2
Karatsuba	4	4	4	1	3	1

The implementation of Karatsuba is less expensive in circuits and logic gates than the other two implementations are about three offsets respectively of 8 bits on 8 bits and 16 bits on 16 bits, two subtractors on 8 bits which represents the modulo 2^8, four additions respectively two on 8 bits and two on 16 bits and three multiplications on 8 bits. And on the other side, the classic implementation takes 512 complete additions on one bit and 256 multiplications with the door and on one bit. And in the same proportion as Horner's is 16 offsets of 1 bit by 16 bits, 256 multiplications on 1 bit with the door and 16 additions on 16 bits.

B. Interpretation of Results

In Table IV, the columns represent the implementation library and the rows represent the type of multiplication implemented. The intersection is the second execution time of a type of multiplication of two 16-bit numbers with one of the column libraries.

TABLE IV. PERFORMANCE TABLE OF MULTIPLICATION BY DGHV, FHEW OR TFHE

	DGHV	FHEW	TFHE
Horner	NA	671	41
Classic	NA	649	39
Karatsuba	NA	483	29

The library implemented for the DGHV did not provide results in a reasonable time to be taken into account in this paper. As for the FHEW and TFHE libraries, the theoretical results corroborated the theoretical hypotheses in memory and time complexity. It appears that the choice made in the design and implementation of the TFHE makes its bootstrapping more efficient.

VI. DISCUSSION

TFHE bootstrapping improves 15 times that of FHEW for this homomorphic multiplication on two 16-bit encrypted integers. This multiplication deteriorates the performance of the TFHE compared to the FHEW by halving the starting assumptions for a logic gate on ciphertexts bits. But in practice, this improvement is negligible if we consider that a binary multiplication of two 16-bit on plaintext numbers on the same architecture is carried out in less than 1 nanosecond. The ratio of improvement of the TFHE by adding the decryption time of the result is close to zero. This observation is also valid for the FHEW.

VII. CONCLUSION

Bootstrapping is the basis of unlimited homomorphic processing on encrypted data. This study compared bootstrapping through three patterns to identify its evolution from 2009 to 2016. It emerges from this comparison that the best design and implementation is that of the TFHE which is based respectively on the problem of the LWE on the real torus modulo 1, the bootstrapping by accumulator, on the fast Fourier transform coupled with the parallelism of FMA and AVX data. One avenue to explore is to study the performance of the implemented FHEW with a rapid transform based on the stockham algorithm, optimized throttle calculation and data parallelism.

REFERENCES

[1] Pascal Paillier, Public Key cryptosystem based on composite degree residuosity classes, In Stern 97, pages 223-238. 27, 29, 51, 53, 55.

[2] Taher El Gamal, A public key cryptosystem and a signature scheme based on discrete logarithms. In GR Blakey and David Chaum, editors, CRYPTO 1984, volume 196 of Lectures Notes in computer Sciences, pages 10-18, Springer 1984.

[3] R. L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public key cryptosystems. Common of the ACM, 21:120-126, 178.

[4] Shafi Goldwasser and Silvio Micali, probabilistic encryption, J. Computer. System. Sci, 28(2): 270-299, 1984. 6, 33.

[5] Dan Boneh, Eu-Jin Goh and Kobbi Nissim, Evaluating 2-DNF formulas on ciphertexts, In Joe Killian, editor, TCC 2005, Volume 3378 of Lectures Notes in Computer Science, Pages 325-341, Springer, 2002. 2, 31, 66, 67, 97, 98, 99.

[6] Craig Gentry. "A fully homomorphic encryption scheme". crypto.stanford. edu/craig. PhD thesis. Stanford University, 2009.

[7] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. "Fully Homomorphic Encryption over the Integers". In: EUROCRYPT 2010. Ed. By Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 24–43.

[8] Craig Gentry, Amit Sahai, and Brent Waters. "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based". In: CRYPTO 2013, Part I. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 75–92. doi: 10.1007/978-3-642-40041-4_5.

[9] Léo Ducas and Daniele Micciancio. "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second". In: EUROCRYPT 2015, Part I. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 617–640. doi: 10.1007/978-3-662-46800-5_24.

[10] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast Fully Homomorphic Encryption Library over the Torus. https: / / github . com /tfhe/tfhe. 2016.

[11] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds". In: ASIACRYPT 2016, Part I. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 3–33. doi: 10.1007/978-3-662-53887-6_1.

[12] R. L. Rivest, L. Adleman, and M. L. Dertouzos. "On Data Banks and Privacy Homomorphisms". In: Foundations of Secure Computation, Academia Press (1978), pp. 169–179.

[13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping". In: ITCS 2012. Ed. by Shafi Goldwasser. ACM, Jan. 2012, pp. 309–325.

[14] Brakerski Z. and Vaikuntanathan V., Fully Homomorphic Encryption from RingLWE and security for key dependent messages, LNCS, vol. 6841, Springer Verlag, Proceedings of CRYPTO, pp. 505-524, 2011.

[15] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: 37th ACM STOC. Ed. by Harold N. Gabow and Ronald Fagin. ACM Press, May 2005, pp. 84–93.

[16] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In CRYPTO, pages 138–154, 2010.

[17] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, Public Key Cryptography, volume 6056 of Lecture Notes in Computer Science, pages 420–443. Springer, 2010.

[18] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In Kenneth G. Paterson, editor, EUROCRYPT, volume 6632 of Lecture Notes in Computer Science, pages 129–148. Springer, 2011.

[19] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. Cryptology ePrint Archive, Report 2011/279, 2011. http: //eprint.iacr.org/2011/279.

[20] N. Howgrave-Graham. Approximate integer common divisors. in J. Silverman (ed), Cryptography and Lattices, Springer LNCS 2146 (2001) 51−66.

[21] Jean-Sebastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully Homomorphic Encryption over the Integers with Shorter Public Keys. Cryptology ePrint Archive, Report 2011/441, 2011. http://eprint.iacr.org/.

[22] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. 9 In Public-Key Cryptography–PKC 2014, pages 311–328. Springer, 2014.

[23] JungHee Cheon, Jean-Sébastien Coron, Jinsu Kim, MoonSung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch Fully Homomorphic Encryption over the Integers. In Thomas Johansson and PhongQ. Nguyen, editors, Advances in Cryptology – EUROCRYPT 2013, volume 7881 of Lecture Notes in Computer Science, pages 315–335. Springer Berlin Heidelberg, 2013.

[24] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. In David Pointcheval and Thomas Johansson, editors, EUROCRYPT, volume 7237 of Lecture Notes in Computer Science, pages 446–464. Springer, 2012.

[25] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Batch Fully Homomorphic Encryption over the Integers. Cryptology ePrint Archive, Report 2013/036, 2013. http://eprint. iacr.org/.

[26] Brakerski Z. and Vaikuntanathan V., Efficient Fully Homomorphic Encryption from standard LWE, Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, pp. 97-106, IEEE Computer Society, 2011.

[27] BLLN13. Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In Martijn Stam, editor, IMA Int. Conf., volume 8308 of Lecture Notes in Computer Science, pages 45–64. Springer, 2013.

[28] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors overs rings. In In Proc. Of EUROCRYPT, Volume 6110 of LNCS, pages 1-23. Springer, 2010.

[29] Yuanmi Chen and Phong Q. Nguyen. Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic Encryption Challenges over the Integers. In David Pointcheval and Thomas Johansson, editors, EUROCRYPT, volume 7237 of Lecture Notes in Computer Science, pages 502–519. Springer, 2012.

[30] Kassem Kalach, Implementation of the multiplication of large numbers by FFT in the contexts of cryptographic algorithms, August 2005, dissertation, Université de Montréal.

[31] P. Boale Bomolo, S. Ntumba Badibanga, E. Mbuyi Mukendi, Implementation of homomorphic arithmetic operations on integers, volume 5, number 2, May 2021, pp 125-137, IJISR.

[32] P. Boale Bomolo, S. Ntumba Badibanga, E. Mbuyi Mukendi, performance of Adder Architectures on encrypted integers, volume 10, issue-6, august 2021, IJEAT.