Selecting the Best Parameters: How to Accelerate Somewhat Homomorphic Encryption for Cloud Auditing

Louis Tajan Moritz Kaumanns Dirk Westhoff
Hochschule Offenburg University
Offenburg, Germany
{louis.tajan,moritz.kaumanns,dirk.westhoff}@hs-offenburg.de

***Abstract*—In a Semi-autonomic cloud auditing architecture we weaved in privacy enhancing mechanisms [14] by applying the public key version of the Somewhat homomorphic encryption (SHE) scheme from [4]. It turns out that the performance of the SHE can be significantly improved by carefully deriving relevant crypto parameters from the concrete cloud auditing use cases for which the scheme serves as privacy enhancing approach. We provide a generic algorithm for finding good SHE parameters with respect to a given use case scenario by analyzing and taking into consideration correctness, performance and security of the scheme. Also, to show the relevance of our proposed algorithms we apply it on two predominant cloud auditing use cases.**

***Index Terms*—Somewhat Homomorphic Encryption, Software Acceleration, Cloud Auditing.**

I. Introduction

In the field of computation on encrypted data ideally one would like to perform both, additively and multiplicatively arithmetic operations on encrypted data in its most flexible way. This is what FHE [6] achieves. However, for most application fields FHE comes with a non-acceptable performance degradation, both with respect to runtime and with respect to ciphertext versus cleartext data size. For the above reasons, in the work at hand we are investigating the usage of a SHE scheme [4] which we want to carefully adjust to concrete privacy enhancing cloud auditing use cases.

One may argue that the most recently proposed Leveled Fully Homomorphic Encryption (LFHE) [3] scheme which is based on the building blocks of SHE [4] scheme may be a more advanced choice. However, we argue that for performance reasons this is surely not the case. Additional costs for the LFHE extension of the SHE mainly arise due to a freshness function which is applied subsequently to each addition operation and multiplication operation on ciphertext polynomials with the two objectives to weave in i) a fresh key as well as ii) a new modulus. Whereas to our understanding the first aims at what we coined "perfect forward computation" for computation on encrypted data (e.g. in similarity to "perfect forward secrecy" for communication protocols), a new modulus per computation level aims to reach "fully" homomorphic in particular for the limited operation in SHE, namely the multiplication. This way the LFHE compared to the FHE is surely performance saving, however compared to the SHE for the above denoted reasons still more complex. Since we are aiming for concrete use cases, specifically aiming for a homomorphic encryption scheme which supports in a cost efficient manner many addition operations but only few multiplicative operations, still the original SHE scheme is a promising candidate for our setting.

It is in our pre-dominant interest to accelerate arithmetic operations on encrypted data, such that they can be performed also on a relatively large number of data. It turns out that this is a cross-layer effort, since parameters from i) the use case that shall be supported need to be mapped to the performed number of arithmetic operations and size of the required cleartext value space, ii) from the encoding layer (plaintext polynomials), and finally iii) from the encryption layer (ciphertext polynomials). Since many of these parameters turn out to be mutually dependent a proper configuration algorithm is required, which orchestrates them to apply SHE in a speed-optimized and secure way for a given setting. We will independently analyze the parameters regarding the correctness, the performance and the security aspects of the scheme and we will merge them in order to draw this proper configuration algorithm. Exactly this is the value of the work at hand. We examplarily show the benefits of our approach by choosing two relevant use cases from the arena of privacy-preserving cloud auditing.

II. Background

A. The Ring Learning with Errors Problem

The Ring Learning with Errors (RLWE) problem, presented by Lyubashevsky *et al.* in [10], corresponds to the larger Learning with errors (LWE) problem specialized to polynomial rings over finite fields. The solution to the RLWE problem may be reducible to the NP-Hard Shortest Vector Problem (SVP) in Lattice. The RLWE assumption is characterized by multiple parameters, rings $R := \mathbb{Z}[x]/\langle f(x)\rangle$ and $R_q := R/qR$ for some degree n integer polynomial $f(x) \in \mathbb{Z}[x]$, a prime integer $q \in \mathbb{Z}$ and an error distribution χ over R. The ring R_q thus represents the ring of degree n polynomials modulo $f(x)$ with coefficients in $\mathbb{Z}_q$. With these configuration, addition in this ring is done component-wise in their coefficients and multiplication is simply polynomial multiplication modulo $f(x)$ and q. Let $s \xleftarrow{\$} R_q$ be a uniformly random ring element. The assumption says that given any polynomial number of samples of the form $(a_i, b_i = a_i \cdot s + e_i) \in (R_q)^2$, where a_i is uniformly random in R_q and e_i is drawn from the error distribution χ, the b_i's are computationally indistinguishable from uniform in R_q. This specific case of the RLWE assumption where ring elements are represented polynomials has been highlighted by Brakerski and Vaikuntanathan in [4] and referenced as the Polynomial Learning with Errors (PLWE) problem.

B. The SHE Scheme

As we argued in the Section I, the studied encryption scheme here is the somewhat homomorphic encryption scheme presented by Brakerski and Vaikuntanathan in 2011 [4]. This scheme requires to firstly encode the message into a polynomial representation subsequently to encrypt it with the respective encryption function resulting in a couple of polynomials. Figure 1 illustrates the steps to encrypt a message.

1) Message Space: The plaintext polynomial space corresponds to $R_t = \mathbb{Z}_t[x]/(x^n + 1)$ with two parameters t and n. The polynomial degree n limits the amount of coefficients while parameter t bounds the coefficients size.

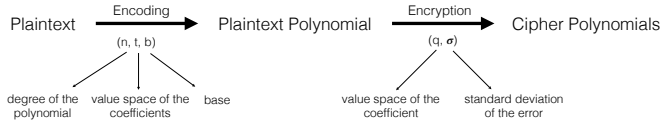

Figure 1. Encoding and encryption path for a message.

In [2], Bieberstein considers one additional parameter comparing to the presentation of this scheme by Brakerski *et al.* : the base b. This parameter represents the base of all the polynomials. In other words, when encoding and decoding the messages, we have $x = b$. In a classical manner we consider this parameter equal to 2 (as it is presented in [8]). For example, if we have a message equal to 89 and parameter $t = 2$ and $n = 8$, we encrypt it by using his binary representation $89_{10} \equiv 1011001_2$ and $m = 0 + 2^6 + 0 + 2^4 + 2^3 + 0 + 0 + 1 = 0x^7 + 1x^6 + 0x^5 + 1x^4 + 1x^3 + 0x^2 + 0x^1 + 1x^0$. In this classical configuration, we are encoding the message with coefficient $c_i \in \{0, \ldots, b-1\}$. Bieberstein proposes that b could be different than 2 or t. In particular he is specifying that $b = 10$ is the easiest encoding. However, he did not propose anything regarding t while in [8], t should be prime and less than q. We are now evaluating the choice of parameter t over parameter b.

a) Positive Messages only: $b, t \geqslant 2$

- $t < b$: we cannot represent all the values.
- $t = b$: fits perfectly, $M = \{0, \ldots, t^n - 1\}$ and $\#elements = t^n$.
- $t > b$: multiple representations for some values, $M = \{0, \ldots, (t-1)(\frac{1-b^n}{1-b})\}$ and $\#elements = t^n$.

b) Positive and Negative Messages: $b \geqslant 2$, $t \geqslant 3$ and $\frac{-t}{2} < c < \frac{t}{2}$ with $c \in \mathbb{Z}$ the coefficients of the polynomial.

For t odd:

- $t \leqslant b$: we cannot represent all the values.
- $t = b + 1$: fits perfectly, $M = \{-(\frac{t}{2} - 1)(\frac{1-b^n}{1-b}), \ldots, (\frac{t}{2} - 1)(\frac{1-b^n}{1-b})\}$ and $\#elements = (t-2)(\frac{1-b^n}{1-b}) + 1$.
- $t > b + 1$: multiple representations for some values, $M = \{-(\frac{t}{2} - 1)(\frac{1-b^n}{1-b}), \ldots, (\frac{t}{2} - 1)(\frac{1-b^n}{1-b})\}$ and $\#elements = (t-1)^n$.

For t even:

- $t < b$: we cannot represent all the values.
- $t = b$: fits perfectly, $M = \{-(\lfloor \frac{t}{2} \rfloor - 1)(\frac{1-b^n}{1-b}), \ldots, (\lceil \frac{t}{2} \rceil - 1)(\frac{1-b^n}{1-b})\}$ and $\#elements = 2(\lceil \frac{t}{2} \rceil - 1)(\frac{1-b^n}{1-b}) + 1$.
- $t > b$: multiple representations for some values, $M = \{-(\lfloor \frac{t}{2} \rfloor - 1)(\frac{1-b^n}{1-b}), \ldots, (\lceil \frac{t}{2} \rceil - 1)(\frac{1-b^n}{1-b})\}$ and $\#elements = t^n$.

2) Distribution:

a) Uniform Distribution: When we write $d \xleftarrow{\$} S$ that means d chosen from the uniform distribution over some finite set S. In the latter sections we specify it to the ring R_q.

b) Gaussian Distribution: We let the distribution $\chi = D_{\mathbb{Z}^n,\sigma}$ to indicate the n-dimensional discrete Gaussian distribution. To sample a vector $x \in \mathbb{Z}^n$ from this distribution, sample $y_i \in \mathbb{R}$ from the Gaussian of standard deviation σ and set $x_i := \lfloor y_i \rceil$, where $\lfloor \cdot \rceil$ represents rounding to the nearest integer. Using the isomorphism mentioned above, we treat χ as the error distribution over integer degree n polynomials defined by the probability density function in [8]:

$$\forall \mathbf{e} \in \mathbb{Z}^n : \Pr[\mathbf{e} \leftarrow D_{\mathbb{Z}^n,\sigma}] = \frac{e^{-\pi \|\mathbf{e}\|^2/\sigma^2}}{\sum_{e \in \mathbb{Z}^n} e^{-\pi \|\mathbf{e}\|^2/\sigma^2}} \tag{1}$$

We let χ' be a noise distribution like χ, only with larger standard deviation σ' as it is presented in [4].

$$\sigma' \geq 2^{\omega(log\ n)} \cdot \sigma \tag{2}$$

3) Key Generation: We are considering here the public-key version of the scheme such that a key pair is being generated. The secret key is set as $sk = s \in R_q$ where we sample a small ring element $s \xleftarrow{\$} \chi$. The public key corresponds to a RLWE instance: $pk = (a_0, b_0 = a_0 s + te_0)$. $a_0 \xleftarrow{\$} R_q$ is uniformly randomized and $e_0 \xleftarrow{\$} \chi$ a small error.

4) Encryption: Given the public-key $pk = (a_0, b_0)$ and a message $m \in R_q$, elements $v, e' \xleftarrow{\$} \chi$ and $e'' \xleftarrow{\$} \chi'$ are set and the ciphertext is defined as $ct = (c_0, c_1) = (b_0 v + te'' + m, -(a_0 v + te'))$.

5) Decryption: Given the secret key $sk = s$ and a ciphertext $ct = (c_0, c_1)$, we first compute $\widetilde{m} = c_0 + c_1 s \in R_q$. Secondly we output the decrypted message $m \equiv \widetilde{m} \mod t$. In case of a ciphertext with more than two elements, the generic decryption formula is:

$$\widetilde{m} = \sum_{i=0}^{d} c_i \cdot s^i \tag{3}$$

6) Addition: Given two ciphertexts $ct = (c_0, \ldots, c_d)$ and $ct' = (c'_0, \ldots, c'_d)$, the addition of the two corresponds to the following ciphertext:

$$ct_{add} = ct + ct' = (c_0 + c'_0, \ldots,\ c_d + c'_d) \tag{4}$$

Namely, addition is done by coordinate-wise vector addition of the ciphertext vectors. If the two ciphertexts have a different length, we should pad the shorter ciphertext with zeroes on the most significant bits. The output of the addition of two ciphertexts $ct = (c_0, \ldots, c_\delta)$ and $ct' = (c'_0, \ldots, c'_\gamma)$ contains $\max(\delta + 1, \gamma + 1)$ ring elements. Therefore addition does not increase the number of elements in the ciphertext vector.

7) Multiplication: Given $ct = (c_0, c_1)$ and $ct' = (c'_0, c'_1)$, the multiplication computes $ct_{mult} = (c_0 c'_0, c_0 c'_1 + c'_0 c_1, c_1 c'_1)$. Homomorphic multiplication increases the size of the ciphertext. The output of the multiplication of the two ciphertexts $ct = (c_0, c_1, \ldots, c_\delta)$ and $ct' = (c'_0, c'_1, \ldots, c'_\gamma)$ contains $\delta + \gamma + 1$ ring elements and will be represented as:

$$\left(\sum_{i=0}^{\delta} c_i v^i\right) \cdot \left(\sum_{i=0}^{\gamma} c'_i v^i\right) = \sum_{i=0}^{\delta+\gamma} \tilde{c}_i v^i \tag{5}$$

III. Discussion of the SHE Parameters

The parameters should be chosen according to guarantee i) the correctness, ii) an acceptable performance and iii) an appropriate security level.

A. Correctness Considerations

To guarantee correctness of the resulting value after having computed a certain amount of additions and multiplications on the ciphertexts, we have to ensure three aspects of the ciphertext evolution. The growing of the information's degree, the growing of the information's coefficient and finally the growing of the error. Indeed, even if the ciphertext is not growing by itself because of the q and $x^n + 1$ moduli reductions, these aspects are growing inside of it. In Figure 2 we can see the two ways the information is spreading. The choice of the parameters (n, t and q) will determine the maximal value that could take the degree and the coefficients. By performing computations on the ciphertexts, the plaintext information weaved into them will

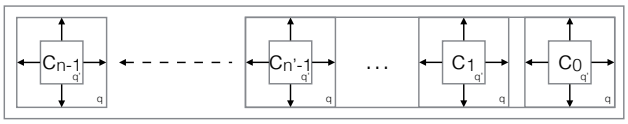

Figure 2. The two dimensional growing of the information in the ciphertext polynomials

be able to grow up to these limitations. We have seen previously that addition and multiplication could be performed directly on ciphertexts. In addition to consider the amount of executions of these two types of operations, we should also be aware of the order of execution of the operations. Indeed, this will impact the plaintext's maximal possible value and thus the correctness of the final result. Let's define $\|ct\|_\infty := \max(\mid c_0 \mid, \mid c_1 \mid, \ldots, \mid c_\delta \mid)$ the infinite norm of a ciphertext which correspond to the maximal size of its coefficients in the plaintext dimension. For example, we first encode a plaintext value $m \in \mathbb{Z}_\mathcal{N}$ into a message polynomial p and we choose an encoding method where the coefficient values are minimized. Then we get $p = (c_0, c_1, \ldots, c_{n-1})$ with $\forall i \in \{0, \ldots, n-1\},\ 0 \leqslant c_i < b$, so $\|p\|_\infty = b-1$. This polynomial is then encrypted by the current SHE scheme into a polynomial $ct = \{ct_0, ct_1\}$ and we get $\|ct\|_\infty = q-1$. Let us now suppose that we are performing an encrypted addition between the ciphertexts ct and ct' similarly produced ($\|ct'\|_\infty = q-1$). Since the encrypted addition is an arithmetic addition performed coefficient wise as presented in Section II-B6, the resulting ciphertext ct_{add} will have the same norm due to the modulo, $\|ct_{add}\|_\infty = q-1$. However, in the plaintext space, the resulting polynomial will have its norm doubled $\|p_{add}\|_\infty = 2b+2$. We could then generalize to: $\|Add(p,p')\|_\infty = \|p\|_\infty + \|p'\|_\infty$. To be able to optimize the choice of parameter t (i.e. choosing it as small as possible while still ensuring security and correctness), the best solution would be to predict the maximal retrieved coefficient after performing the computations circuit on the plaintexts. Parameter t will have to be greater than the infinite norm of the resulting corresponding polynomial.
We also have to consider limitations from the initial work [4] in the Key Dependent Messages (KDM) scenario:

- $t > \sigma\sqrt{n}$ (i.e. a sample from χ resides in R_t with all but negligible probability).
- $t < 2^{-\omega(log n)} \cdot \sigma^{-d} \cdot q$ with $ct = (c_0, \ldots, c_d)$.

Let us now consider the encrypted multiplication. We define the degree function $deg(p) := i$ such that i is the degree of polynomial p. If we perform an addition between two ciphertexts, since the addition is computed on the respective coefficients, the information will not spread to higher coefficients. On the contrary, when we perform a multiplication of some ciphertexts, the information will affect higher degree coefficients. When we multiply two polynomials, p and p' we can denote $deg(p_{mult}) = deg(p) + deg(p')$ if $deg(p), deg(p') > 1$. In the ciphertext space, the degree of the encrypted polynomial will not increase because of the $x^n + 1$ modulo and at any time $deg(ct) \leqslant n-1$. The use of multiplication on the ciphertext will result in a limitation of parameter n. Indeed, if after too many multiplications, $deg(p_{mult}) > n-1$ then the modulo wrapping will remove information and the final decryption will not be correct. For that reason, with the degree of the "fresh" encoded polynomials and the computations circuit, we will have to predict the degree of the resulting polynomial such that parameter n remains less than this degree.

B. Performance Considerations

In [8], Lauter *et al.* state for a reduction of the parameter's size to improve performance. In the following section we aim to analyze how these parameters are manageable and concretely what are their limitations. We should determine which form could be the best with respect to the objective to reduce the computation cost of the cryptographic functions. We face two opposite strategies: firstly, having a high polynomial degree with small coefficients and secondly having a low maximal degree with significant coefficients.
We recall that parameter b is the base that is used to encode the message into a polynomial. If we choose a small base, by definition, we could get a small message space reduction t but that will imply a use of a large polynomial degree n. Vice versa, a choice of a larger base b will increase the lower bound of t since the "fresh" coefficients will be larger and will grow faster with computations. A choice of a larger encoding base could also reduce the lower bound of n and thus help reducing the homomorphic operation complexities.

The choice of the base b and how we decide to encode the message polynomial have to be consider. Regarding b we could consider two opposite strategies of encoding the messages into polynomials. First we could encode any message with coefficients $c_i \in \{0, \ldots, b-1\}$. With this strategy we will get a maximal degree for the encoding polynomial. The second strategy will be to have on the contrary a minimal degree for the message polynomial. We then could get the first coefficient such that $c_0 = m$ (for m the complete plaintext message) and $c_i = 0$ for all $0 < i < n$. By this usage we get an encoding polynomial of degree 0 and therefore parameter n will not have to be that large.

To be able to represent a specific message space, we have to adjust the three parameters namely n, t and q, the polynomial degree, the value spaces of the coefficients of the cleartext and ciphertext. As message spaces, we should not consider only the initial message space but also the final message space. This corresponds to the possible values of the messages after being computed in the ciphertext space. In [8], the authors consider that a wise strategy regarding performance would be to reduce parameters t, q or n. They suggest to encode the messages as a list of bits. In other terms, as a polynomial with coefficients equal to $\{0,1\}$. Such a strategy will directly require the largest parameter n for a given value space. Indeed, the minimal value of n should be equal to the logarithm of the maximal encoded message. Also, this technique implies a use of the base (parameter b) equal to 2, otherwise not all the values could have been encoded (see Section II-B1). As we will show later, this seems to be not the most optimal strategy to set the parameters.

We are now investigating how reducing the size of the polynomials will impact the performance of the encryption and computations of the ciphertexts. In a first step we will see how a variation of parameter n impacts the performance. Subsequently we do this for parameter q. We are expressing the cryptographic functions in terms of elementary operations to see how they are connected to parameter n. We consider here fresh ciphertexts, (i.e. ciphertext with two polynomials) and polynomials multiplication performed with the FFT algorithm:
$\mathbf{1\ Enc_{data}} \rightarrow 3\ Sample_{poly} + 3\ Add_{poly} + 2\ Mult_{poly} + 3\ scMult_{poly} \rightarrow 3n\ Sample_{coef} + (2n\log n + 7n - 8)\ Add_{coef} + (2n\log n + 7n - 8)\ Mult_{coef} + (4n\log n + 13n - 20)\ Mod_{coef}$
$\mathbf{1\ Add_{data}} \rightarrow 2\ Add_{poly} \rightarrow 2n\ Add_{coef} + 2n\ Mod_{coef}$
$\mathbf{1\ Mult_{data}} \rightarrow 4\ Mult_{poly} + 4\ Red_{poly} + 1\ Add_{poly} \rightarrow (4n\log n + 9n - 16)\ Add_{coef} + (4n\log n + 8n - 16)\ Mult_{coef} + (21n - 40)\ Mod_{coef}$
$\mathbf{1\ Dec_{data}} \rightarrow 3\ Mult_{poly} + 1\ Add_{poly} \rightarrow (3n\log n + 7n - 12)\ Add_{coef} + (3n\log n + 6n - 12)\ Mult_{coef} + (6n\log n + 16n -$

30) Mod_{coef}

The results show that the complexity of the four cryptographic functions is linearithmic in term of n.

We now analyze the complexity at a deeper level. We consider the coefficients encoded with the *two's complement* encoding technique. We use the notation *qBits* which represents the size of the parameter q in bits, in other terms, $\log q$.

$\mathbf{1\ Enc_{data}} \rightarrow 3n\ Sample_{coef} + [(2n \log n + 7n - 8)(10\ qBits^2 - 14\ qBits + 8) + (4n \log n + 13n - 20)(5\ qBits - 2)]\ Add_{bit} + [(2n \log n + 7n - 8)(3\ qBits^2 - 2qBits + 1) + (4n \log n + 13n - 20)\ qBits]\ Mult_{bit}$

$\mathbf{1\ Add_{data}} \rightarrow 4n(5\ qBits - 2)\ Add_{bit} + 4n\ qBits\ Mult_{bit}$

$\mathbf{1\ Mult_{data}} \rightarrow [(4n \log n + 9n - 16)(10\ qBits^2 - 14\ qBits + 8) + (21n - 40)(5\ qBits - 2)]\ Add_{bit} + [(4n \log n + 9n - 16)(3\ qBits^2 - 2qBits + 1) + (21n - 40)\ qBits]\ Mult_{bit}$

$\mathbf{1\ Mult_{data}} \rightarrow [(3n \log n + 7n - 12)(10\ qBits^2 - 14\ qBits + 8) + (6n \log n + 16n - 30)(5\ qBits - 2)]\ Add_{bit} + [(3n \log n + 7n - 12)(3\ qBits^2 - 2qBits + 1) + (6n \log n + 16n - 30)\ qBits]\ Mult_{bit}$

We see here that the complexity is quadratic in term of the size *qBits*. Or in term of parameter q, it is polylogarithmic. We could then affirm that a reduction of n is the predominant parameter saving performance even if that should imply a growing of q. This should be the guiding principle that leads our aimed algorithm.

C. Security Considerations

We are now taking security aspects into account by integrating security measurement against a specific attack. As in [8], we consider the distinguising attack presented in [9]. In this work, the authors analyze the security of a lattice-based encryption scheme based on the LWE problem. This scheme is an instance of an abstract system described by Micciancio [11] that generalizes all the schemes of [13], [12], [7]. This cryptosystem could be assimilated to the one we are currently using, since it is based on the same mathematical problem and both of them are characterized by the same parameters.

In this section we will consider this attack and express the robustness of the scheme according to the selected parameters. As it is also done in [8], we express the security of the scheme through the logarithm of the running time (in seconds) of the attack itself with each specific parameters configuration. This amount of security depends on three parameters n, $log(q)$ and l. Parameter l corresponds to the size of the messages that have to be encrypted and expressed in bits.

$$log(T) = \frac{1.8\ (2n + l)^2}{n\ log(q)} - 110 \tag{6}$$

As it is presented in the Appendix of [5], the security of this cryptosystem could also be reduced to the hardness of the RLWE assumption (defined in II-A). The parameter σ, the standard deviation of the used Gaussian distribution, relies on the security of such problem. This parameter should be chosen such that it could avoid combinatorial style attack. An attack of this kind is presented in [1] which breaks LWE in time $2^{O(\|e\|_\infty^2)}$ with high probability, where e is the LWE error vector. Since e is chosen by the discrete Gaussian distribution with standard deviation σ, if we pick σ large enough then this attack should be prevented. Thus, according to [5] choosing $\sigma \geqslant 8$ will ensure that it is large enough to avoid combinatorial attacks. Also we define the second standard deviation $\sigma' = 2^{log(n)} \cdot \sigma$ to get slightly stronger overall security by making the public key better-protect than any ciphertext.

IV. Strategy to Reduce Computation Costs

We are now getting to the main idea of this work which is to integrate the three previously presented considerations correctness, performance and security in order to obtain algorithms that generate the perfect parameters to a suitable usage. These algorithms allow a user, respectively in our case a database administrator to select appropriate and performance saving parameters for their own use case. The algorithm's arguments T, M_i and λ correspond respectively to the arithmetic tree of computations, the set of the value spaces of the computed data and the security parameter. In other words, M_i corresponds to the value spaces of the T's leaves. The overall protocol is presented in Algorithm 1 and we describe here each of its steps in details:

Set n: As we have seen in Section III-B *Performance Considerations*, we should choose parameter n a power of 2 as little as possible. Thus we start with $n = 2$.

Set b: To initialize the encoding base, we consider parameters n and M_i and we apply the formula $b \geqslant \lceil \sqrt[n]{Max(M_i)} \rceil$.

Compute p_{result}: In the objective to select parameter t, we firstly have to compute a prediction of the resulting polynomial after decryption phase (see Algorithm 2). In that way, we will select parameter t such that the *t-reduction* during the decryption will not remove any information as we have seen in Section III-A *Correctness Considerations*. For this step the concrete UC's arithmetic tree T is necessary to consider the order of the homomorphic operations and to which data they are processing. Then we verify that $deg(p_{result}) \geqslant n$ if not, parameter b should be set greater in order to reduce p_{result}'s degree.

Set t: We choose t prime such that $t > \|p_{result}\|_\infty$. In other words, t should be greater than the maximal possible value of a coefficient from the resulting polynomial.

Set σ and σ': As we have seen in Sections II-B2b and III-C, to avoid combinatorial style attacks we set the standard deviations $\sigma = 8$ and $\sigma' = 2^{\log n} \cdot \sigma$.

Set q: We set the modulo q regarding others parameters. q should be a sub-exponential prime such that $q \equiv 1 \pmod{2n}$. To be able to decrypt correctly, q should also fulfill $q > 2 \cdot \|p_{result}\|_\infty \cdot (t\sigma n^{1,5})^{D+1}$ with Algorithm 3 that enables to compute D.

Compute $log(T)$: We compute the security level of the generated parameters and verify if it fulfills the security parameter λ. If not, we start over Algorithm 1 with a greater parameter n.

Algorithm 1 Generating SHE Parameters.

Input: M_i, T, λ
Output: n, b, t, q, σ, σ'

1: Set $n \leftarrow 2$
2: Set $b \leftarrow \lceil \sqrt[n]{Max(M_i)} \rceil$
3: **if** $b < 2$ **then**
4: $\quad b \leftarrow 2$
5: **end if**
6: Compute $p_{result} \leftarrow F(n, b, T.root)$
7: **if** $deg(p_{result}) \geqslant n$ **then**
8: $\quad$ go back to step 2 with a greater b
9: **end if**
10: Set t prime s.t. $t > \|p_{result}\|_\infty$
11: Set $\sigma \leftarrow 8$
12: Set $\sigma' \leftarrow 2^{\log n} \cdot \sigma$
13: Compute $D \leftarrow CD(C)$
14: Set q prime s.t. $q \equiv 1 \pmod{2n}$ and $q > 2 \cdot \|p_{result}\|_\infty \cdot (t\sigma n^{1,5})^{D+1}$
15: Compute $log(T) \leftarrow (1.80\ (2n + \lceil log(Max(M_i)\rceil))^2)/(n \cdot log(q)) - 110$
16: **if** $log(T) < \lambda$ **then**
17: $\quad n \leftarrow n^2$ and go back to step 2
18: **end if**
19: return σ, σ', b, n, t and q

Algorithm 2 Function F

Input: n, b, C
Output: p_{result}
function POSTORDER(n, b, no) ▷ a node no_i and its two respective children $S1_i$ and $S2_i$
  if $no = null$ **then**
    return
  end if
  POSTORDER($n, b, no.left\_son$)
  POSTORDER($n, b, no.right\_son$)
  if $no.left\_son = null$ AND $S2_i = null$ **then**
    $deg(no) \leftarrow \lfloor log_b(no.M) \rfloor$
    if $no.M < b$ **then**
      $\|no\|_\infty \leftarrow no.M$
    else
      $\|no\|_\infty \leftarrow b - 1$
    end if
  else
    if $no.type = (+)$ **then**
      $deg(no) \leftarrow max(deg(no.left\_son), deg(no.right\_son))$
      $\|no\|_\infty \leftarrow \|no.left\_son\|_\infty + \|no.right\_son\|_\infty$
    else if $no.type = (*)$ **then**
      $deg(no) \leftarrow deg(no.left\_son) + deg(no.right\_son)$
      $\|no\|_\infty \leftarrow 2 \cdot [min(deg(no.left\_son), deg(no.right\_son)) + 1] \cdot \|no.left\_son\|_\infty \cdot \|no.right\_son\|_\infty$
    end if
  end if
  $p_{result} \leftarrow no$
end function
return p_{result}

Algorithm 3 Function CD

Input: C
Output: $int\ D$
function POSTORDER(no) ▷ a node no_i and its two respective children $S1_i$ and $S2_i$
  $D \leftarrow 0$
  if $no.type = (+)$ **then**
    $D \leftarrow max(POSTORDER(no.left\_son), POSTORDER(no.right\_son))$
  else if $no.type = (*)$ **then**
    $D \leftarrow 1 + POSTORDER(no.left\_son) + POSTORDER(no.right\_son)$
  else
    return 0
  end if
  return D
end function

V. USE CASES

To illustrate the choice of the correct parameters we implement our algorithms and then, we select use cases and perform their execution by using an implementation of the SHE scheme. We run our solution for these specific use cases and verify that the generated parameters are indeed the best to use in a term of performance. We choose two specific use cases which are taken from cloud security auditing to illustrate our purpose. Nevertheless, the reader should understand that our algorithms could be applied to any scenario they may use as long as the SHE scheme is needed. The user will have to provide his arithmetic tree as well as the message spaces of the different values used during the scenario and precise the required level of security.

A. SHE Implementation.

We are using the initial work of A. Bieberstein. For his Master Thesis [2], he implemented the SHE scheme in *C++* with the use of the two arithmetic libraries *Flint* and *GMP*. His work was purely based on the symmetric version and we modified it such that we could use it as a public-key scheme. The need of the scheme with private/public keys was motivated by the use cases that could require the cloud infrastructure PAL SAaaS project. We are also planning to publish this implementation online in the near future. The measurements presented in Table I and Table II have been made with a CPU configuration of Intel Core i5 M520 2.40GHz x 4.

B. UC1 - Access Control Auditing:

The first considered use case is taken from one of our work on cloud security auditing [14]. The cloud auditor is checking on behalf of a client that the CSP's access control has been performed correctly. Some computations will be performed directly on the evidences on the Evidence Store (ES) in the objective to retrieve some amounts of specific data. The considered evidences are partially encrypted with the SHE scheme. Each connection attempt could get the characteristic of being made from an authorized user or not and being successful or not by having an encryption of 1 to the respective field and an encryption of 0 otherwise. To compute a set, we add all of the encryption regarding one field in the ES database and we get the total amount of connection attempts with this characteristic. To get an intersection of sets we firstly multiply two characteristics for each connection attempt and do subsequently add all the products. We then get the total amount of connection attempts with the two characteristics. In Figure 3 we show a preview of the database stored in the ES simplified to two characteristics, along with the operation circuit of the computations that will have to be performed on the data.

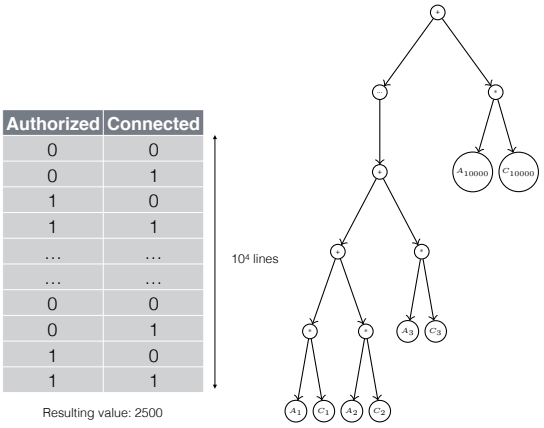

Figure 3. UC1 - Access Control Auditing database and arithmetic tree

We run our algorithms for UC1 and obtain the following set of parameters $\{t = 20011,\ b = 2,\ n = 4096,\ log(q) = 84\}$ for a standard level of security of 128 bits ($\lambda = 128$).

Independently of our resulting set of parameters, we will run our use case with several other sets of parameters to illustrate the accurateness of our algorithms in term of performance.

We suppose an evidence store with 10^4 data (connection attempts). The use case's algorithm will perform 10^4 times a multiplication between two polynomials which represent a simple encryption of 0 or 1.

Table I shows the different running times when we run the use case with the parameters obtained with our algorithms for n equals each power of two. The *Database* column corresponds to the phase during which the ES database is generated by the CSP and in this specific use case, that corresponds to 2.10^4 encryptions. The *Audit* field corresponds to the execution of the computations circuit by the evidence store. We reasonably get results showing that increasing the polynomial degree makes the running times relatively bigger. Also, the increase of parameter n is pushing us to choose a parameter q slightly bigger. We could also notice that the base b remains equal to 2 since the message space is $\{0, 1\}$.

Table I
UC1 RUNTIMES (IN MS) AND LEVEL OF SECURITY FOR DIFFERENT SHE PARAMETERS SETTINGS

t	b	n	$qBits$	Encryption	Addition	Multiplication	Decryption	Database	Audit	$log(T)$
20011	2	2	51	0.019	0.003	0.388	0.013	235	117	-109,5588
20011	2	4	54	0.019	0.003	0.425	0.013	324	182	-109,3250
20011	2	8	57	0.031	0.004	0.774	0.019	530	326	-108,8592
20011	2	16	60	0.052	0.006	1.363	0.031	933	596	-107,9581
20011	2	32	63	0.096	0.009	2.723	0.058	1765	1185	-106,2277
20011	2	64	66	0.188	0.016	5.925	0.123	3594	2535	-102,9087
20011	2	128	69	0.510	0.031	13.462	0.319	9530	5647	-96,5389
20011	2	256	72	0.999	0.059	30.96	0.677	19550	12920	-84,2999
20011	2	512	75	2.049	0.114	68.204	1.409	46605	132537	-60,7520
20011	2	1024	78	2.049	0.114	68.204	1.409	46605	132537	-15,3846
20011	2	2048	81	2.049	0.114	68.204	1.409	46605	132537	72,1333
20011	2	4096	84	2.049	0.114	68.204	1.409	46605	132537	241,1714
20011	2	8192	87	2.049	0.114	68.204	1.409	46605	132537	568,0414

Table II
UC2 RUNTIMES (IN MS) FOR DIFFERENT SHE PARAMETERS SETTINGS

t	b	n	$qBits$	Encryption	Addition	Multiplication	Decryption	Database	Audit	$log(T)$
187520027	294	2	152	0.021	0.004	0.491	0.012	663	776	-82,6211
10880029	18	4	137	0.031	0.006	0.837	0.016	1058	1399	-92,9723
4000037	6	8	136	0.055	0.01	1.524	0.027	1921	2777	-99,4118
640007	3	16	129	0.096	0.017	2.91	0.046	3536	5330	-101,9628
320009	2	32	129	0.18	0.03	5.97	0.09	6805	10468	-102,9107
320009	2	64	136	0.18	0.03	5.97	0.09	6805	10468	-102,3765
320009	2	128	142	0.18	0.03	5.97	0.09	6805	10468	-99,8592
320009	2	254	148	0.18	0.03	5.97	0.09	6805	10468	-94,2378
320009	2	512	154	0.18	0.03	5.97	0.09	6805	10468	-82,9766
320009	2	1024	160	0.18	0.03	5.97	0.09	6805	10468	-60,9950
320009	2	2048	166	0.18	0.03	5.97	0.09	6805	10468	-18,3735
320009	2	4096	172	0.18	0.03	5.97	0.09	6805	10468	64,1500
320009	2	8192	178	0.18	0.03	5.97	0.09	6805	10468	223,9556
320009	2	16384	184	0.18	0.03	5.97	0.09	6805	10468	533,6198

C. UC2 - Billing Service:

This use case represents a billing process for cloud service's usage executed by an auditor on encrypted data from a database. It is a similar use case to UC1 but in addition to the two fields from UC1 (*Authorized* and *Connected*) we require two extra columns in the database namely *Time* and *Price*. For each client the cloud auditor multiplies the service usage with the cost and with the two fields *Authorized* and *Connected* which represent markers. Finally, the auditor sums up all values together and sends the final sum to the client. This use case represents a charging for cloud service's usage. One difference compared to UC1 is the size of the data which are multiplied and added. These values belong no longer to the set $\{0, 1\}$ but represent also some time duration, costs or amounts of data. Therefore, the message space of the message polynomials will be much larger The arithmetic tree of computations is similar to the one from UC1. For each line we do a multiplication between time and cost and another one between the two marker fields. Subsequently we multiply the two resulting products. Finally the summation of all of these 10^4 products is performed. Important to note: since we are multiplying products, we get a multiplication depth of 3.
As in UC1, we show in Figure 4 a preview of the database stored in the ES, along with the operation circuit of the computations that will have to be performed on the data.

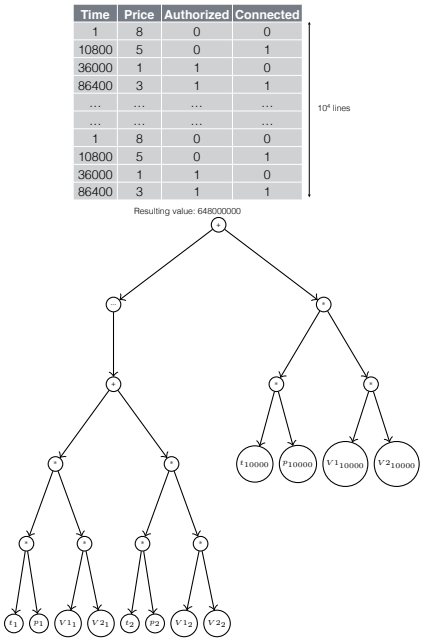

Figure 4. UC2 - Charging Service database and arithmetic tree

We run our algorithms for UC2 and obtain the following set of parameters $\{t = 32009,\ b = 2,\ n = 8192,\ log(q) = 178\}$ for a standard level of security of 128 bits ($\lambda = 128$).

In Table II we show, as for UC1, the different running times. We notice that we could get very small parameter n thanks to an increasing of the base, but to fulfill an acceptable level of security againdt the distinguishing attack we must consider large polynomial degree as $n = 8192$ for a security of 223 bits.

VI. CONCLUSION AND OPEN ISSUES

After analyzing the SHE scheme from Brakerski and Vaikutanathan [4] and performing some tests on its implementation with respect to promising cloud auditing use cases, we established a way of generating the scheme's parameters in the objective to reduce as much as possible the computation cost of the scheme's functions as the encryption or the multiplication of ciphertexts. Our solution relies on the observation that reducing first and foremost the polynomial degree provides us the most significant improvements regarding performance. To this end and despite what the authors claim in [8], the encoding base should be as high as possible. With a relatively high encoding base, the degree of the fresh encoded polynomials will be low and then the parameter n could be reduced. We present a complete algorithm that could generate all the scheme's parameters considering the value spaces of the different encrypted data and the circuit of the computations that will have to be performed directly on the ciphertexts.

REFERENCES

[1] Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Proceedings of the 38th International Colloquim Conference on Automata, Languages and Programming - Volume Part I. ICALP'11, Springer-Verlag, Berlin, Heidelberg (2011)

[2] Bieberstein, A.: An implementation of SomeWhat Homomorphic Encryption scheme from the Ring Learning with Errors. Master's thesis, Hochschule Furtwangen University, Furtwangen, Germany (2014)

[3] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. ITCS '12, ACM, New York, NY, USA (2012)

[4] Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Rogaway, P. (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA. Proceedings. Lecture Notes in Computer Science, vol. 6841. Springer (2011)

[5] Damgard, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535 (2011)

[6] Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University (2009), crypto.stanford.edu/craig

[7] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the fortieth annual ACM symposium on Theory of computing. pp. 197–206. ACM (2008)
[8] Lauter, K., Naehrig, M., Vaikuntanathan, V.: Can homomorphic encryption be practical? Cryptology ePrint Archive, Report 2011/405 (2011)
[9] Lindner, R., Peikert, C.: Better key sizes (and attacks) for lwe-based encryption. In: Kiayias, A. (ed.) Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Lecture Notes in Computer Science, Springer (2011)
[10] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 6110, pp. 1–23. Springer (2010)
[11] Micciancio, D.: Duality in lattice cryptography. In: Public key cryptography. p. 2 (2010)
[12] Peikert, C., Vaikuntanathan, V., Waters, B.: A Framework for Efficient and Composable Oblivious Transfer, pp. 554–571. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
[13] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM 56(6), 34:1–34:40 (Sep 2009), http://doi.acm.org/10.1145/1568318.1568324
[14] Tajan, L., Westhoff, D., Reuter, C.A., Armknecht, F.: Private information retrieval and searchable encryption for privacy-preserving multi-client cloud auditing. In: 11th International Conference for Internet Technology and Secured Transactions, ICITST 2016, Barcelona, Spain, December 5-7, 2016. pp. 162–169. IEEE (2016)