THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 16, Special Issue 2015, pp. 329-338

COMPARISON-BASED APPLICATIONS FOR FULLY HOMOMORPHIC ENCRYPTED DATA

Mihai TOGAN[1], Luciana MOROGAN[2], Cezar PLESCA[3]

[1,3]certSIGN, RD Department, Bucharest, ROMANIA
[2]Military Technical Academy, Bucharest, ROMANIA
E-mail: mihai.togan@certsign.ro

The cloud computing providers need to offer security warranties. As we all know, one of the critical points is the confidentiality and access to customer data which, these days, is migrated and managed in cloud environments. In this sense, one solution is based on encrypting data before its upload in cloud. But this approach sets a limit regarding data processing. In this article we present a practical application of the homomorphic encryption schemes, namely the problem of finding maximum/minimum from a collection of encrypted integers. First, we present our algorithm that can be run directly in cloud without the need for an intermediate data exchange with the client. Second, our experimental results show the time resources necessary to evaluate the proposed algorithm.

Key words: fully-homomorphic encryption, cloud processing, maximum problem.

1. INTRODUCTION

There is a traditional solution for ensuring the confidentiality of the data which is migrated in cloud. It resides in encrypting the data at the source and keeping the encrypted data on servers in cloud. Access to the decryption keys is granted only to authorized users. The traditional model consists in three parts: i) the cloud provider manages data storage services; ii) the data owners encrypt their data files before the upload in the cloud platforms (the encrypted data is stored in cloud or shared with other users that have access rights to it); iii) the operations performed on the data (searches, updates, modifications...) are done by the cloud users which have to download the data in encrypted format and use the necessary keys for decoding these files.

However, this approach has its weaknesses. An important one resides in the fact that such a model, let us call it static, does not allow processing the data directly from the cloud infrastructure. The main advantage of cloud migration, namely the large computing power provided to the customers, cannot be used in this case because the data is not accessible to the cloud. Such a solution limits the cloud infrastructure only to the data storage service. This approach leads to a decrease in scalability and flexibility. There are others shortcomings of the model presented above, but we chose not get into details in these directions as these topics do not represent the subject for the present paper. What is important for the perspective of the present paper is the possibility of being able to process encrypted data directly in cloud. A solution much closer to the model that cloud services are able to offer to their clients involves renting also the processing capabilities.

The idea of using a fully-homomorphic encryption scheme that would work in reasonable performance conditions allow *data to be processed in encrypted format*. The server knows only the processing algorithm. Data processing is done by the server using this algorithm, involving a series of mathematical operations on the data. In this case, the formal scenario implies an entity (e.g. a cloud client) that sends information under an encrypted format to a third party server (e.g. a cloud computing service that cannot be considered trustworthy) in order to store and process data. The server must be able to perform computations on the data according to the algorithm, without having access to the decryption keys (thus, neither to the data). The result of this encrypted format processing is sent back to the client entity that holds the decryption keys.

The scenario we are speaking of was first demonstrated to be possible in 2009 by the first fully-homomorphic encryption scheme. This new scheme, introduced by C. Gentry in [1], allows performing two operations (i.e. addition and multiplication) with encrypted data. The result obtained from these operations

represent precisely the encryption of the result of some operations performed over the data in clear. This entire context of migrating data in cloud aims to perform operations with encrypted data in cloud. From a practical point of view, the functionality and the efficiency of the theoretical model would provide an excellent mechanism for data protection, at least for ensuring confidentiality. With the data migrated by the clients under encrypted forms before reaching the cloud servers and with the homomorphic encryption scheme, the cloud servers could implement any computation on these data without having access to the customers data.

Our contribution, in the above presented context, resides in a solution for the issue of determining the maximum element within a collection of fully-homomorphic based encrypted values. In this regard, we proposed a logarithmic form of the *maximum algorithm* in order to be best adapted to a *leveled – FHE* scheme. Moreover, our algorithm can find the maximum of a given encrypted values array without the need to collaborate with the client. We argued the benefits of our approach in terms of *levels* costs. We made a practical implementation of the algorithm using a leveled-FHE scheme and we measured the involved costs.

The suggested routines can be part of an application running on an untrusted server (located in cloud). A first simple example of applicability that crossed our minds is a cloud server that holds two lists: one corresponding to the name in clear of the employees of a client company and a second one with encrypted values for their corresponding salaries. In this scenario, the client can get from the server the largest value for the amount of the revenue which is paid by the company, without the cloud server to find out this value or whose employee it belongs.

This paper is organized as follows. In **Section 2** we briefly present some homomorphic encryption schemes along with a summary of the BGV [2] scheme. We chose the BGV scheme for the implementation of our solution of finding the maximum from a collection of encrypted integer values. **Section 3** briefly describes the issue of comparing two encrypted integers. We used it within the proposed solution which we described in **Section 4** along with our experimental results. Finally, **Section 5** outlines our conclusions.

2. HOMOMORPHIC ENCRYPTION SCHEMES

Homomorphic cryptography was first introduced by R. Rivest et al. [3] in the early 1978s. Gentry's idea [1] is that any processing algorithm over a set of data can be reduced to a multitude of addition and multiplication operations at the bit level (XORs and, respectively, ANDs). Thus, if those data are bitwise encrypted and the encryption scheme supports the application of as many homomorphic transformations in relation to the two mathematical operations above, then it becomes possible to perform any processing algorithm directly on the encrypted data. The result of the processing is the one that could be obtained if the same operations were applied over the data in clear format.

Still, a question remains. Is this idea efficient enough to be used for computing models applied over externalized data? The efficiency is analyzed in relation to (1) the time required for executing the encryption/decryption algorithms and the processing operations evaluation and (2) the required computing resources (memory, processing power etc.), both on the server and at the client. Gentry analyzed in [4] the time required to a Google search using a query containing only one word which is encrypted. The encryption is realized using techniques from [1]. Gentry estimated that this new kind of Google search is a trillion times longer than the existing time required for queries in unencrypted format of the same word. Even if deeper analyses were made on Gentry's proposal, his estimation offers quite an eloquent dimension of the inefficiency of these schemes at the moment.

Another challenge with these types of schemes is represented by the support in obtaining the highest possible flexibility at the level of the processing algorithm. This flexibility refers to the ability of the scheme to support a wider range of operations on the encrypted data, while also keeping the accuracy of the results and an acceptable level of resources required for accomplishing this processing. In this context of performance, regarding efficiency and flexibility, there are more issues that compromise Gentry's idea. Since 2009, a lot of research effort was focused on finding reliable solutions that meet the requirements of the scenario presented at the beginning of this section. In the recent years, there were many continuations to Gentry's work, either new schemes or new optimizations to the existing ones [2, 5 – 12].

2.1. BGV Encryption Scheme

The simplest and one of the most efficient fully homomorphic encryption scheme which is known to date was constructed in [6] and it was refined in [2]. The whole construction is based on the so called "learning with errors" (LWE) problem, first presented by Regev in [13] (see also [14]). The LWE assumption states that if $\mathbf{s} \in Z_q^n$ is an n dimensional (secret) vector, then any polynomial number of "noisy" random linear combinations of the coefficients of $\mathbf{s}$ are computationally indistinguishable from uniformly random elements in Z_q. More precisely:

$$\{\mathbf{a_i}, \langle \mathbf{a_i}, \mathbf{s} \rangle + e_i\}_{i=1}^{\text{poly}(n)} \overset{c}{\approx} \{\mathbf{a_i}, u_i\}_{i=1}^{\text{poly}(n)}, \tag{1}$$

where $\mathbf{a_i} \in Z_q^n$ and $u_i \in Z_q$ are uniformly random, and the "noise" e_i is sampled from a noise distribution that outputs numbers much smaller than q (for example, a discrete Gaussian distribution over Z_q with small standard deviation). The LWE problem is at least as hard as finding short vectors in any lattice (see [13] and [15]).

To encrypt a bit m, a random $a \in Z_q^n$ and a "noise" $e \in Z_q$ are chosen. It is computed $b \leftarrow m + 2e + \langle \mathbf{a}, \mathbf{s} \rangle$ with $\mathbf{s}$ representing the public/secret key. The cipher-text is $c \leftarrow (\mathbf{a}, b) \in Z_q^{n+1}$. For the decryption process, there has to be done the computation $b - \langle \mathbf{a}, \mathbf{s} \rangle$. The result is represented by $2e + m \pmod q$. Since e is chosen to be much smaller than q, it is obtained that $2e + m \pmod q = 2e + m$. Finally, it is computated $2e + m \pmod 2$ to obtain m.

This scheme is homomorphic in respect to addition, until too much noise accumulates. To make it homomorphic in respect to multiplication it is needed the re-linearization, introduced in [2, 6]. This method allows the multiplication by encrypting the resulted product under a new secret key. By posting a "chain" of L secret keys, it may be performed up to L levels of multiplications. This new construction produces a leveled fully homomorphic encryption scheme without using Gentry's bootstrapping procedure (see [1]).

3. COMPARISON OF *FH*-ENCRYPTED INTEGERS

For the algorithm which we propose in the next section of this paper, we needed the $>$ comparison operator. In this manner, we make a short review of our previous implementation [16] of a homomorphic evaluation for the comparison operators applied to encrypted integer values.

In [16], we bring in an $O(log_2(n))$ solution that evaluates the comparison $X > Y$. This comparison evaluates two *fully-homomorphic*–based encrypted integers. Long story short, this approach uses the binary representations of the integers $X = \overline{x_{n-1}x_{n-2}\cdots x_0}$ and $Y = \overline{y_{n-1}y_{n-2}\cdots y_0}$. In the particular case of the one-bit length integers, x and y, the comparison operators can be expressed using the following polynomial relations in Z_2. The addition and the multiplication operations represent the bitwise XOR and AND:

$$x > y \Leftrightarrow xy + x = 1 \tag{2}$$

$$x = y \Leftrightarrow x + y + 1 = 1 \tag{3}$$

For the general case of n-bit length integers, the polynomial relation which can evaluate the comparison $X > Y$ are recursively constructed using the following decomposition ($l = \lceil n/2 \rceil$):

$$\overbrace{x_{n-1}\cdots x_l}^{msb(X)}\overbrace{x_{l-1}\cdots x_0}^{lsb(X)} > \overbrace{y_{n-1}\cdots y_l}^{msb(Y)}\overbrace{y_{l-1}\cdots y_0}^{lsb(Y)} \Leftrightarrow$$
$$\big(msb(X) > msb(Y)\big) \vee \big(msb(X) = msb(Y)\big) \wedge \big(lsb(X) > lsb(Y)\big) \tag{4}$$

The relations (2) and (3) are generalized by the recurrences presented below (choosing each time $l = \lceil j/2 \rceil$). This generalisation is possible making use of (4) and the definitions of the two functions t and z (see bellow):

$$t_{i,j} = \begin{cases} x_i y_i + x_i, & j = 1 \\ t_{i+l,j-l} + z_{i+l,j-l} t_{i,l}, & j > 1 \end{cases} \quad (5)$$

$$z_{i,j} = \begin{cases} x_i + y_i + 1, & j = 1 \\ z_{i+l,j-l} z_{i,l}, & j > 1 \end{cases} \quad (6)$$

The two functions t and z are defined as follows:

1. $t_{i,j}$ represents the boolean value corresponding to the truth value of the expression $\overline{x_{i+j-1} \cdots x_i} > \overline{y_{i+j-1} \cdots y_i}$
2. $z_{i,j}$ represents the boolean value corresponding to the truth value of the expression $\overline{x_{i+j-1} \cdots x_i} = \overline{y_{i+j-1} \cdots y_i}$.

The evaluations of $X > Y$ and $X = Y$ can be obtained by computing $t_{0,n}$ and $z_{0,n}$, respectively. Their definitions led to an implementation based on divide-and-conquer approach. The main benefit consists in the depth of the equivalent boolean circuits which are exactly $\lceil log_2(n) \rceil + 1$ in the case of $t_{0,n}$ and $\lceil log_2(n) \rceil$ for $z_{0,n}$. The practical implementation (followed by the corresponding experimental results) described in [16] is built on top of HElib library [17]. It consists in coding the corresponding ***compute_t*** and ***compute_z*** recursive functions (C/C++ code). In this manner, we used the ***leveled*** version of the BGV FHE scheme [2] (embedded in the 2014 version of HElib). The reported time for the comparison of two 8-bit integers, $X > Y$, is $\approx 12 seconds$ (for 128 bits of the claimed security and using one core of an Intel(R) Xeon(R) E5-1620 at 3.6 GHz).

In the literature, there is another approach of the same issue of the encrypted integers comparison, which can be found in [18]. This one is based on integers substraction and evaluation of the resulted sign-bit. As both methods, [18] and [16], have similar efficiency, we chose to use [16] as it represents the result of our previous work.

4. THE MAXIMUM OF A FH-ENCRYPTED ARRAY

4.1. A naive approach

A rather naive approach for the maximum problem was already suggested in [16]. The method requires the existence of a messages exchange protocol between a server and a client. The messages are represented by FH-encrypted values. In short, we supposed that the server holds a set of N encrypted numbers $\{Cx_1, Cx_2, \ldots, Cx_N\}$, with Cx_i representing the FH-encryption of the value x_i for $i = \overline{1 \ldots N}$. The algorithm to find the maximum value, that we proposed in [16], is described below:

• For each index i found in the $\{1, \ldots N\}$ interval, the server makes the following computations:

- For all j with $j \neq i$, the server computes the $F(Cx_i, Cx_j)$ evaluation. The F function is the result of calling a function ***compute_s*** that evaluates the result of $\geq$ operator. The result of the evaluation of $F(Cx_i, Cx_j)$ is $Enc(1)$ if and only if $x_i \geq x_j$;

- The server computes the product $P(i) = \prod_{j \neq i} F(Cx_i, Cx_j)$. $P(i)$ is $Enc(1)$ if and only if x_i is the maximum in the set $\{x_1, x_2, \ldots, x_N\}$ ($x_i \geq x_j$ for any $j \neq i$).

• The server sends the value of $P(i)$ to the client

• The client decrypts it and if the obtained value is **1**, then the protocol stops with the result that the x_i value is the maximum element

• If the client decrypts **0**, then it will request the server to go for the next i in the index set $\{1,\dots N\}$.

The above solution has some disadvantages. For each i^{th} iteration of this message exchange, the client has indeed the possibility to know whether or not the maximum of the array is represented by the x value (indexed by i in the set of encrypted values). It is obvious that, in the worst case, in order to find the maximum over a set of N values, it is necessary to know N intermediate results computed and sent by the server to the client. Furthermore, for the computation of each of these values, there are necessary $N-1$ evaluations of the comparison operator $\geq$ (which is also the most expensive of the three operators that can be evaluated $>,=,\geq$). Another disadvantage is due to the fact that the server can find the information regarding the position i of the maximum element, even if it does not know its value. Indeed, this can be a problem considering the context of certain applications which involve the usage of the solution for the maximum.

Resuming the simple example we presented in the first chapter of this paper, we try to exemplify this disadvantage. Suppose, again, the server in cloud that holds the two lists: one containing the employees of a client company (storing the names of the employees in clear, on the server) and one composed by the encrypted values for their salaries. In the context of the client requesting the employee (meaning the index in the list) with the highest income, the server can find out the person with the maximum revenue. In order to eliminate this problem, the protocol above needs to be continued so as to fully browse the list of indexes up to step $i = N$, even if the client already learned the maximum in an intermediate step. In this case, all $N-1$ comparison evaluations are performed on the server, although at the client some of the N decryptions may not be necessary.

4.2. The new proposal

We continue our previous work (started in [16]), where we used the functions of homomorphic evaluation of the comparison operators that we proposed. In the present section we describe a better implementation and an improvement of the algorithm for finding the maximum value of a set of FH-encrypted integers. Some experimental results for sustaining our results are also summarized. Our new solution, that we are about to present, adapt the well known classical algorithm for finding maximum of an array (presented in *Algorithm 1*) in order to support the homomorphic evaluations.

Algorithm 1 Classical algorithm for maximum

```
1: function GETMAX(int V [ ], int N) , where N is the number of elements of V
2:     int max = v[0];
3:     for (int i = 1; i < N; i++)
4:         if (v[i] > max)
5:             max = v[i];
6:     return max;
7: end function
```

To allow the homomorphic evaluation of the above algorithm we have to rewrite some parts of it (the decision section) using polynomial forms. First of all, we use an additional function that makes the selection of the maximum of two values. The result of this selector depends on the result of the comparison between its input values. We define the selection function as follows:

$$sel : \mathbb{Z}\times\mathbb{Z}\times\mathbb{Z}_2 \to \mathbb{Z}, sel(a,b,t) = \begin{cases} a, & \text{if } t = 1 \\ b, & \text{if } t = 0. \end{cases} \quad (7)$$

It is easy to see that the function **sel** (which is in fact the maximum selection operator of two integer values a and b, depending on the value of t) can be written under a polynomial form:

$$sel(a,b,t) = at + (1-t)b \tag{8}$$

In [16], we defined the function **compute_t** which returns the boolean value of the comparison $a > b$ evaluation. We replaced the comparison component of the *Algorithm 1* with the function **compute_t** and the decisional component with the selection function **sel**. In this way, if we have an array of FH bitwise-encrypted integers and we are using the polynomial implementation of **compute_t**, then we can run the function described by the *Algorithm 2* to find the maximum value of an array containing FH-encrypted integers. The new form of the maximum algorithm is presented in *Algorithm 2*.

Algorithm 2 The maximum algorithm

```
1: function GETMAX(int V [ ], int N), where N is the number of elements of V
2:     int max = v[0];
3:     for (int i = 1; i < N; i++)
4:     {
5:         t = compute_t (v[i], max);
6:         max = sel (v[i], max, t);
7:     }
8:     return max;
9: end function
```

We obtained the polynomial form of the function **sel** having all operations in Z_2 (the multiplication and the addition are realized using the AND and XOR bit-operators) by using the binary representations for $a = \overline{a_{n-1}, a_{n-2}, ...a_0}$ and $b = \overline{b_{n-1}, b_{n-2}, ...b_0}$. We also needed to define $T = \overline{t_{n-1}, t_{n-2}, ...t_0}$ as the integer value having all bits 1 or 0 (representing the n-duplicated result of the **compute_t** function). We used the definition of **sel** under the following polynomial form:

$$sel : Z_2^{1..n} \times Z_2^{1..n} \times Z_2^{1..n} \to Z_2^{1..n} \tag{9}$$

$$sel(a,b,T) = \sum_{i=0}^{n-1} a_i t_i + (1+t_i) b_i \tag{10}$$

For the implementation of the **sel** function, which evaluates the polynomial relation in (10), we worked with bit-level encrypted values (therefore, all operations are done in Z_2, where additions and multiplications mean XORs and ANDs, respectively). For the FHE needed crypto-primitives (*KeyGen*, *Enc*, *Dec*, *add* and *mul* operators) we used the features provided by HElib [17]. We also adapted the finding maximum function (GETMAX) to a form that uses the divide-and conquer technique. As the *Algorithm 2* uses a linear iteration of the input array, in the context of the homomorphic encrypted values, it is not optimal. The non-optimal feature arises from the $N-1$ iterations (comparisons and selections) with an $O(N)$ equivalent homomorphic circuit depth (the number of consecutive multiplications which have a negative impact for the growth of the curve of the cipher-text noise). The divide-and-conquer approach is more appropriate in this case. Overall, it does not decrease the total number of iterations (therefore, neither the total number of involved multiplications), but it reduces the homomorphic circuit depth to $O(log_2(N))$ instead of $O(N)$.

The *Algorithm 3* contains our final recursive implementation. We are working with bit-level encrypted integers. The encryption of each integer results in a vector of ciphertexts and, in this case, the input array of the maximum algorithm is an array of vectors of encrypted bits. Using this approach, the server knows the algorithm required for it to deliver the maximum element in the array. What is of great importance is the fact that the server does not know neither the value of the maximum, nor its position (its index) in the input collection of elements. This hidden information occurs as a result of the truth values of the comparisons

which are not known to the server (the ***ct_t*** value is also encrypted). In this case, the selection of the maximum of two values occurs invisibly for the server. Additionally, each intermediate maximum value ***ct_max*** (the encrypted maximum of two integers) is modified as a result of the mathematical computations which are required for its homomorphic evaluation. Finally, after completing all iterations, the server is unable to identify the maximum of the array.

Algorithm 3 The maximum algorithm working on encrypted values

```
1: function GETMAX(vector < vector < Ctxt >> V [ ], int start, int N)
2:     if (N == 1)
3:         return V [start]
4:     vector<Ctxt> ct_max_1 = GETMAX (V, start, ⌊N/2⌋);
5:     vector<Ctxt> ct_max_2 = GETMAX (V, start + ⌊N/2⌋,⌈N/2⌉);
6:
7:     vector < Ctxt > ct_t = compute_t (ct_max_1, ct_max_2);
8:     vector < Ctxt > ct_max = sel (ct_max_1, ct_max_2, ct_t);
9:     return ct_max;
10: end function
```

5. EXPERIMENTAL RESULTS

For the experimental part of the *Algorithm 3*, we used the HElib-based implementation of the comparison operator (>) made within [16]. For n -bits length integers, each iteration of the algorithm needs $log_2(n)+1$ levels for evaluation of the **compute_t** function (comparison). The **sel** function (selection) contains only one multiplication and, therefore, it needs one additional level. We have already argued that the circuit depth is $log_2(N)$. This means a total of $log_2(N)(log_2(n)+2)$ levels needed for the evaluation of the maximum finding algorithm for an array of N encrypted integers. This will conduct to a limitation of using the leveled version of the [2] scheme. In our case, for an initial setup measuring L levels, we can keep the correctness of the algorithm for at most $2^{L/(log_2(n)+2)}$ numbers. There is well known that the performance of the scheme is affected by the bound $N = \phi(m)$, modulus chain length (fixed number of levels) and the security parameter k.

For $N = 1 + L/(log_2(n)+2)$, the linear completion of the *Algorithm 2* conducts to a threshold which can be completely inefficient in real applications. In order to illustrate this fact, let us consider the example of a configuration of $L = 50$ levels. *Algorithm 2* limits the array to a maximum of $N = 11$ numbers (8-bits integers). In this proper case, *Algorithm 3* allows a processing of 1025 numbers representing a threshold which is much more closer to the needs of the real applications. As the parameter L drastically affects the performance and the security of the scheme, its value can not increase unconditionally. Though, it can be established a trade-off between N, the number of values that are processed by a real application, and the values of m and L which provide an acceptable level for the security and the efficiency. If N exceeds the value for L (which is acceptable in terms of the scheme security and efficiency), the solution can be based on the *bootstrapping technique*. Even using the leveled version under the *bootstrapping optimization* of the BGV scheme [2], the tree approach which we proposed in *Algorithm 3* is more efficient. This efficiency derives from the fact that the *recrypt* operation (which is time consuming) will be rarely used (as its rate of applicability is logarithmic instead of the linear one that is used by the *Algorithm 2*).

We conducted experiments testing the time consumption for the evaluation of our algorithm. We tried various combinations of the m and L parameters. These parameters define the level of security and they affect the efficiency of the FHE scheme that we used. For the security level we worked with the HElib estimation (in accordance with [19], equation number 8, for $\sigma = 1$). In **Table 1** we resume the costs (in terms of both time and memory consumption) that we measured for the homomorphic evaluation of the *GETMAX* function.

Table 1

The GETMAX evaluation costs (in terms of time and memory) of different configurations.

Params	Sec(k)	getmax	comp_t	select	L_f	Enc	Dec	Mem
m = 21845, L = 22, N = 16	48 bits	**279.1 sec**	10.4 sec	8.3 sec	1	1.5 sec	0.7 sec	**~ 800MB**
m = 28679, L = 22, N = 16	106 bits	**346.9 sec**	12.8 sec	10.2 sec	1	1.6 sec	0.9 sec	**~ 1.0GB**
m = 53261, L = 40, N = 16	140 bits	**1295 sec**	47.3 sec	38.9 sec	18	5.7 sec	3.2 sec	**~ 3.8 GB**

The conducted tests involved an workstation with an x64 of openSUSE 12.1 distribution (Intel i7-4710HQ processor running at 3.5 GHz, one core and 8GB RAM). Working with the bit-level encryption, we used $p = 2$ as the base for plain-space specific parameter of the HElib and we used 23 bits per level. **Table 1** contains the needed time costs for the homomorphic evaluation of the *GETMAX* function for an array of $N = 16$ integer values (of $n = 8$ bits length). From the table, one can observe that for an 140 bits security level, the maximum element can be found in 21 minutes. The average time consumption values at each iteration of the algorithm are also presented in **Table 1**. This time consumption is implied by two basic operations: the comparison (*ct_t*) and the selection (*ct_max*). The L_f value denotes the base level where the maximum element is found at the end of the evaluation.

In the case of HElib implementation for an initial fixed L, the values of L_f are consistent with the computations above, considering that the base level of a recent encrypted number is $L-2$. Resuming to our case, for $N = 16$ and $n = 8$ there are taking place a number of $L = 5$ modulus-switching operations at each of the $log_2(N) = 4$ necessary levels for the circuit completion. The table contains (indicative) both the average time required for the encryption (*Enc*) and the decryption (*Dec*) of an 8-bit integer and the average memory needed for the evaluation (the array of the encrypted numbers is loaded entirely into the memory). It is obvious that the memory requirement is quite high (3.8 GB for $L = 50, N = 16$ numbers). A more optimal solution may be considered as making use of some software engineering tricks (e.g. keeping into memory, when needed, of only a subset of the array which enters or is about to enter into the processing iteration).

The proposed solution can be considered relatively efficient in terms of time and the size of the processed data. Still, the memory consumption may lead to a limitation. For $m = 21845$ and $L = 40$, we tested $N = 101$ encrypted numbers. The result containing the maximum value we obtained in 3400 seconds and it required 4.8GB of memory. For comparative purposes we did an alternative implementation for the maximum algorithm. It is based on the FHEW library [20], which provides the FHE symmetric encryption scheme detailed by [12]. The FHEW library allows to encrypt single bit messages using a fast bootstrapping technique to supports the homomorphic evaluation of arbitrary boolean circuits on encrypted data. Using the FHEW, in the same hardware conditions and similar security level, finding the maximum of 16 integers took about 3000 seconds, at least twice the time consumed by HElib leveled implementation.

6. CONCLUSIONS

In this paper we proposed a solution for the homomorphic evaluation of the maximum problem in the case of an encrypted array. Our solution is based on the *bitwise* encryption of integer values and all the computations take place in Z_2. We built upon our previous work [16] that present a solution for the evaluation of the comparison operator applied to encrypted integers. The new proposal assures the confidentiality for both of the maximum value and its position into the array.

As simple as it is in the plain domain, the maximum problem rise some challenges when it comes to an array of encrypted values. First, we present a naive solution which has several security and efficiency issues. Second, in contrast with this approach, we proposed a new algorithm that does not need to exchange intermediate data with the client in order to identify the maximum. Another contribution of this paper concerns the adaptation of the proposed algorithm (i.e. the reduction of levels consumed) for leveled-FHE schemes with respect to efficiency. The usage of a leveled FHE schemes [2] removes the need of the *recrypt*

operation specific to the bootstrapping techniques and provides an acceptable degree of efficiency in the case of real applications.

Technical aspects of our implementation and some experimental results involving the time costs are described in the last part of the paper. For our implementation we used the latest version of HElib library [17]. At the moment, the simple usage of this library represents a challenge as it contains low-level crypto-primitives and an optimal configuration of its parameters is needed to be set and adjusted through experiments for each type of application.

We consider that efficiently solving the maximum/minimum problem in the encrypted domain is important as this could lead to solutions for other types of more complex techniques such as clusterization. The advantages of the FHE schemes are incontestable in the context of outsourcing data processing to the cloud. Actually, there is a trend to migrate known problems and algorithms to the homomorphic encryption field. We can find many recent attempts in the literature which target homomorphic evaluation of known problems like sorting of data, signal processing or cryptographic algorithms like AES, Simon or SHA2/3 [18, 19, 21, 24, 26]. The main problem still remains the lack of the efficiency, and the fact that each application must be adapted in a very customized mode to get reasonable performances with the actual implementation of the known FHE schemes.

ACKNOWLEDGMENTS

This research was partially supported by the Romanian National Authority for Scientific Research (CNCS-UEFISCDI) under the project PN-II-IN-DPST-2012-1-0086 (ctr. 9DPST/2013). All the authors contributed equally to this work.

REFERENCES

1. C. GENTRY, *A Fully Homomorphic Encryption Scheme*, PhD Thesis, Stanford University, http://crypto.stanford.edu/craig, 2009.
2. Z. BRAKERSKI, C. GENTRY, V. VAIKUNTANATHAN, *Fully Homomorphic Encryption without Bootstrapping*, Innovations in Theoretical Computer Science Conference, pp. 309–325, 2012.
3. R. RIVEST, L. ADLEMAN, M. DERTOUZOS, *On Data Banks And Privacy Homomorphisms*, Foundations of Secure Computation, **4**, *11*, pp. 169–180, 1978.
4. C. GENTRY, *Computing Arbitrary Functions of Encrypted Data*, Communications of the ACM, **53**, *3*, pp. 97–105, 2010.
5. M. VDIJK, C. GENTRY, S. HALEVI, V. VAIKUNTANATHAN, *Fully Homomorphic Encryption Over The Integers*, Advances in Cryptology–Eurocrypt 2010, Lecture Notes in Computer Science, 6110, pp. 24–43, 2010.
6. Z. BRAKERSKI, V. VAIKUNTANATHAN, *Efficient Fully Homomorphic Encryption From (Standard) LWE*, IEEE 52nd Annual Symposium on Foundations of Computer Science, pp. 97–106, 2011.
7. Z. BRAKERSKI, V. VAIKUNTANATHAN, *Fully homomorphic encryption from ring-LWE and security for key dependent messages*, Advances in Cryptology – Crypto 2011, Lecture Notes in Computer Science, **6841**, pp. 505–524, 2011.
8. N.P. SMART, F. VERCAUTEREN, *Fully Homomorphic SIMD Operations*, IACR Cryptology ePrint Archive: Report 2011/133, http://eprint.iacr.org/2011/133.pdf, 2011.
9. Z. BRAKERSKI, *Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP*, Advances in Cryptology – CRYPTO 2012, Lecture Notes in Computer Science, **7417**, pp. 868-886, 2012.
10. D. BONEH, C. GENTRY, S. HALEVI, F. WANG, D.J. WU, *Private database queries using somewhat homomorphic encryption*, ACNS 2013, Lecture Notes in Computer Science, **7954**, pp. 102-118, 2013.
11. J.-S. CORON, T. LEPOINT, M. TIBOUCHI, *Batch fully homomorphic encryption over the integers*, IACR Cryptology ePrint Archive: Report 2013/036, https://eprint.iacr.org/2013/036.pdf, 2013.
12. L. DUCAS, D. MICCIANCICIO, *FHE Bootstrapping in less than a second*, IACR Cryptology ePrint Archive: Report 2014/816, http://eprint.iacr.org/2014/816.pdf, 2014.
13. O. REGEV, *On Lattices, Learning With Errors, Random Linear Codes And Cryptography*, ACM Symposium on Theory of Computing, pp. 84–93, 2005.
14. V. LYUBASHEVSKY, C. PEIKERT, O. REGEV, *On Ideal Lattices And Learning With Errors Over Rings*, Advances in Cryptology – Eurocrypt 2010, Lecture Notes in Computer Science, **6110**, pp. 1–23, 2010.
15. C. PEIKERT, *Public-key Cryptosystems From The Worst-Case Shortest Vector Problem: Extended Abstract*, ACM Symposium On Theory of Computing, pp. 333–342, 2009.

16. M. TOGAN, C. PLESCA, *Comparison-based computations over fully homomorphic encrypted data*, 10th International Conference on Communications (COMM), pp. 1–6., 2014.
17. S. HALEVI, V. SHOUP, *The HElib library*, https://github.com/shaih/HElib, 2015.
18. C. AGUILAR-MELCHOR, S. FAU, C. FONTAINE, G. GOGNIAT, R. SIRDEY, *Recent Advances in Homomorphic Encryption*, IEEE Signal Processing Magazine, **30**, *2*, pp. 108–117, 2013.
19. C. GENTRY, S. HALEVI, N. P. SMART, *Homomorphic Evaluation of the AES Circuit (Updated Implementation)*, IACR Cryptology ePrint Archive: Report 2012/099, https://eprint.iacr.org/2012/099.pdf, 2015.
20. L. DUCAS, D. MICCIANCIO, FHEW, *A Fully Homomorphic Encryption library*, https://github.com/lducas/FHEW, 2014.
21. R.L. LAGENDIJK, Z. ERKIN, M. BARNI, *Encrypted Signal Processing for Privacy Protection: Conveying the utility of homomorphic encryption and multiparty computation*, IEEE Signal Processing Magazine, **30**, *1*, pp. 82–105, 2013.
22. Y. DOROZ, Y. HU, B. SUNAR, *Homomorphic AES Evaluation Using NTRU*, IACR Cryptology ePrint Archive: Report 2014/039,
23. T. LEPOINT, M. NAEHRING, *A Comparison of the Homomorphic Encryption Schemes FV and YASHE*, IACR Cryptology ePrint Archive: Report 2014/062, https://eprint.iacr.org/2014/062.pdf, 2014.
24. B. CARMER, D.W. ARCHER, *Block Ciphers, Homomorphically*, Galois, Inc., 2014.
25. Y. DOROZ, A. SHAHVERDI, T. EISENBARTH, B. SUNAR, *Toward Practical Homomorphic Evaluation of Block Ciphers Using Prince*, IACR Cryptology ePrint Archive: Report 2014/233, https://eprint.iacr.org/2014/233.pdf, 2014.
26. S. MELLA, R. SUSSELA, *On the Homomorphic Computation of Symmetric Cryptographic Primitives, Cryptography and Coding*, Lecture Notes in Computer Science, **8308**, pp. 28–44, 2013.