THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 16, Special Issue 2015, pp. 375-384

HOMOMORPHIC EVALUATION OF SPECK CIPHER

Mihai TOGAN[1], Cristian LUPASCU[2], Cezar PLESCA[3]

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

In this paper we make some considerations regarding the homomorphic evaluation of the SPECK *lightweight* cipher. Firstly, we show a reduced form (in terms of the homomorphic circuit-depth) for the corresponding boolean circuit of the 32-bits integers adder. This result leads to the possibility of performing more efficient homomorphic implementations for other applications which are based on the 32-bits integers addition. Secondly, we present a bit-sliced implementation for the homomorphic evaluation of the SPECK cipher encryption round. The proposed implementation offers good results in terms of time costs and memory required to process the algorithm's rounds. With our implementation we completed 11 of 22 rounds of the SPECK32/64. From our knowledge, this is the first research of its kind for this cipher.

Key words: fully-homomorphic encryption, lightweight cipher, SPECK evaluation.

1. INTRODUCTION

Outsourcing data processing in the cloud in good security conditions offers the use of cloud services a major advantage. Homomorphic encryption could be a solution in this regard, especially if it can be combined with more efficient protection methods, such as symmetric encryption algorithms. The homomorphic cryptography was first introduced by R. Rivest et al. [1] in the early 1978s. The first fully-homomorphic encryption scheme was first demonstrated to be possible (at least in theory), much later in 2009, by C. Gentry [2]. Using fully-homomorphic encryption schemes allows data to be processed by untrusted parties in encrypted format. Since 2009, a lot of research effort was focused on finding better solutions that meet resonable requirements in terms of efficiency. In the last years, there were many continuations to Gentry's work, either new schemes or new optimizations to the existing ones [3 – 10].

It is well known that type FHE encryption schemes (even SWHE) are heavy consumers of computing resources (CPU time, required memory, output data size). Their applicability, at least the theoretical one, is undeniable in the context of securing data in the cloud. But the immense consumption of resources limits their effectiveness on all levels: *a)* when encrypting the data to the client before they are uploaded in the cloud; *b)* when transmitting the data on the communications channel between the client and the server; *c)* when processing the data, in encrypted format, on the server (in cloud). In terms of memory size, the requirements are of the GBs order even for small input data set (with a reasonable security of 80/100 bits). In this regard, the model based on the afore mentioned flow could be modified. The aim is to reduce the effort both for the client and during the communication. A solution could be based on the following alternative model [11]:

On the client:

a) The data are encrypted in a traditional manner (using a symmetric block cipher, e.g. AES), $SymEnc(K, data)$;

b) The symmetric encryption key is FHE encrypted, $FHEEnc(K)$;

c) The client sends to the server the symmetrically encrypted data and the FHE encrypted key, $[SymEnc(K, data), FHEEnc(K)]$.

On the server:

d) The server performs a FHE encryption of the $SymEnc(K, data)$, using the FHE public key, and gets $FHEEnc(SymEnc(K, data))$;

e) The server performs the symmetrical decryption of data using a homomorphic evaluation of the symmetric decryption function. The server gets symmetrically decrypted, but FHE encrypted data, $HomEvalSymDec(FHEEnc(SymEnc(K, data)), FHEEnc(K)) \rightarrow FHEEnc(data)$;

f) The server can now apply processing functions (homomorphic evaluations) on the FHE encrypted data getting results that are then sent to the client.

The main benefit of the model described above is that it reduces the size of the data transferred in the cloud (we are talking about GBs of data). Furthermore, it reduces the amount of resources required on the client side. The symmetric encryption (e.g. AES) ensures a very good efficiency in terms of speed (CPU processing time), required memory and the output size. The size of $SymEnc(K, data)$ is the same with the input $data$ size. The described model replaces (at the client) the FHE data encryption with the FHE encryption of the symmetric key (its size is much reduced, compared to the data size). This model's drawback is the additional effort for the server prior to the actual data processing. The server needs to go through the additional $FHEEnc$ and $HomEvalSymDec$ homomorphic functions. This effort could be compensated by the much bigger processing power of servers in the cloud, as compared to the one available at the client. Furthermore, these additional computations can be performed once and then multiple processings can be performed on the data. The greatest benefit occurs at the level of data transmission between client and server. Sending data of the order of GBs (or TBs) through network environments is totally inefficient and it leads to very high costs. Considering these facts, the implementation of homomorphic evaluations for symmetric algorithms becomes very attractive. These evaluations can be full or partial (decryption, or encryption only). One option (may be more optimally) is to expand the symmetric key K to the client (before being FHE encrypted). The client expands the round keys K_i and then encrypts them using FHE. A trade-off can be made in this case between speed gains which could occur on the server – the evaluation step for the round keys expansion algorithm is avoided – and the additional data volume that needs to be sent from the client to the server.

Our contribution. In this paper we make a first attempt to perform a homomorphic evaluation of the SPECK [12] cipher. To the best of our knowledge, no such work exists yet. In the terms of homomorphic computation effort, the costliest operation of the SPECK is the integers addition (modulo 2^n). In this regard, we propose a reduced form of the boolean circuit required by the 32-bit integers adder. This allows homomorphic evaluation of the 32-bit addition in 5 levels at most. The most optimal depth that we found in similar approaches for this operation is 10 levels [13]. This leads to the possibility of performing more efficient implementations for other applications (e.g. homomorphic evaluations) which are based on the 32-bits integer additions.

Using this, we propose a homomorphic implementation of the SPECK cipher round. To get a good level of efficiency, we chose to use the BGV [3] leveled-FHE scheme. We applied and tested our implementation on a reduced form of the SPECK32/64 cipher. More precisely, we homomorphically computed 11 of the full 22 rounds, keeping a reasonable level of security and efficiency (in terms of processing time and memory costs).

This paper is organized as follows. In **Sections 2** and **3** we briefly present the SPECK *lightweight* cipher and the BGV leveled–FHE scheme, respectively. **Section 4** contains the considerations on the 32-bits integers addition and our homomorphic implementation details along with the experimental results. A few similar works are shortly described in **Section 5**. Finally, **Section 6** outlines our conclusions.

2. SPECK BLOCK CIPHER

In 2013, the U.S. National Security Agency (NSA) released the SPECK and SIMON encryption algorithms [12]. These were proposed as two families of *lightweight* block ciphers. The ciphers were designed to provide optimal software and hardware performances in the context of limited computing environments (specially on microcontrollers, ASICs, etc.). The SPECK cipher is based on a Feistel structure and allows few variants of data block and key sizes. Each variant supports data words of n-bit which implies data block sizes of $2n$-bits and m-word keys leading to mn -bits key sizes. In general, a variant of the cipher is referred to as SPECK $2n/mn$ and can be instantiated with values of $n = 16/24/32/4\ 8/64$ and $m = 2/3/4$.

Algorithm 1. SPECK Encryption

```
Input: $x, y$ - Plaintext
Input: $k[T-1], \cdots, k[1], k[0]$ - Round–keys sequence
Output: $x, y$ - Ciphertext
for $i = 0 \cdots T-1$ do
    $x \leftarrow (S^{-\alpha}x + y) \oplus k[i]$
    $y \leftarrow S^{\beta}y \oplus x$
end for
```

Algorithm 2. SPECK round–keys expanding

```
Input: $l[m-2], \cdots, l[0], k[0]$ - Initial key sequence
Output: $k[T-1], \cdots, k[1], k[0]$ - Expanded keys sequence (round–keys)
for $i = 0 \cdots T-2$ do
    $l[i+m+1] \leftarrow (k[i] + S^{-\alpha}l[i]) \oplus i$
    $k[i+1] \leftarrow S^{\beta}k[i] \oplus l[i+m+1]$
end for
```

The encryption, decryption and key–expanding functions of SPECK are based on the following operations:

- bitwise XORs
- additions modulo 2^n bits
- left and right rotations by α bits (denoted as S^{α} and $S^{-\alpha}$, respectively)

For a $k \in GF(2)^n$, the k–th encryption round of a SPECK $2n$ instance is defined by the map: $R_k : GF(2)^n \times GF(2)^n \to GF(2)^n \times GF(2)^n$ and is implemented by the relations:

$$R_k(x, y) = ((S^{-\alpha}x + y) \oplus k, S^{\beta}y \oplus (S^{-\alpha}x + y) \oplus k). \tag{1}$$

The *Algorithm 1* presents the pseudo-code of the SPECK encryption routine. The key–scheduling function inputs the chipher key $K = l_{m-2}, l_{m-1}, \cdots, l_1, k_0$ and generates the T–rounds keys sequence $k_0, k_1, \cdots, k_{T-1}$ using the steps presented by the *Algorithm 2*.

3. BGV SCHEME

The simplest and one of the most efficient fully homomorphic encryption scheme which is known to date was constructed in [5] and it was refined in [3]. The whole construction is based on the so called "learning with errors" (LWE) problem, first presented by Regev in [14] (see also [15]). 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:

$$\{\mathbf{a_i}, \langle \mathbf{a_i}, \mathbf{s} \rangle + e_i\}_{i=1}^{\mathrm{poly}(n)} \overset{c}{\approx} \{\mathbf{a_i}, u_i\}_{i=1}^{\mathrm{poly}(n)}, \tag{2}$$

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 [14] and [16]). 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 $2e + m$ (mod q). Since e is chosen to be much smaller than q, it is obtained that $2e + m$ (mod q) $= 2e + m$. Finally, it is computated $2e + m$ (mod 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 [3, 5]. 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 [2]).

4. HOMOMORPHIC EVALUATION OF SPECK

4.1. FHE addition over $Z_{2^{32}}$

The integers addition modulo 2^n is required by the SPECK functions. This operation involves high costs in the context of the homomorphic evaluations. This is because of the large number of multiplications required by the carry bits computations. Considering two 32-bit integers $x = \overline{x_{31} \cdots x_1 x_0}$ and $y = \overline{y_{31} \cdots y_1 y_0}$, to compute $s = x + y$ modulo 2^{32} we need to design a boolean circuit for the adder that computes the sum and carry bits. The circuit will be homomorphic evaluated and that requires to propose its smallest form in terms of depth-levels. Using a trivial approach, the sum and carry bits can be expressed as follows:

$$\begin{aligned}
&s_0 = x_0 \oplus y_0 \\
&s_1 = x_1 \oplus y_1 \oplus c_1, \text{where } c_1 = x_0 y_0 \\
&s_2 = x_2 \oplus y_2 \oplus c_2, \text{where } c_2 = (x_1 y_1) \vee (x_1 c_1) \vee (y_1 c_1) = (x_1 y_1) \vee ((x_1 \vee y_1) c_1) \\
&\cdots \\
&s_{31} = x_{31} \oplus y_{31} \oplus c_{31}, \text{where } c_{31} = (x_{30} y_{30}) \vee (x_{30} c_{30}) \vee (y_{30} c_{30}) = (x_{30} y_{30}) \vee ((x_{30} \vee y_{30}) c_{30})
\end{aligned} \tag{3}$$

We can apply the De Morgan law to evaluate the OR operator by using of Z_2 additions and multiplications operations (XORs, ANDs):

$$x_i \vee y_i = \overline{\overline{x_i} \wedge \overline{y_i}} = ((x_i \oplus 1)(y_i \oplus 1)) + 1. \tag{4}$$

Working with this design, in the case of the first carry bit c_1 we consume one level, while for the rest of the carry bits we need 2 more additional levels for each of c_2 to c_{31}. That means a total number of 61 levels that should be consumed in order to implement the homomorphic evaluation of the adder, in this manner.

A much better approach can be found in [13]. It has been proposed in the context of the homomorphic evaluation for the SHA256 algorithm, which involves also 32-bit addition operations. The authors presented in [13] a solution for the 32-bit adder and use within their design the look-ahead technique. Finally, they estimated that the 32-bit adder can be homomorphic evaluated using 10 levels. These are needed to compute the most significant carry c_{31}.

In the following, we propose an optimized version for the addition over $Z_{2^{32}}$, which aims to reduce the corresponding boolean circuit of homomorphic multiplications to a maximum depth-5 circuit. In this regard, we use also a carry look-ahead adder approach. We start the design for our circuit defining the sum and carry bits of the adder respectively as follows:

$$s_i = x_i \oplus y_i \oplus c_i \tag{5}$$

$$c_{i+1} = c_i \oplus (x_i \oplus c_i) \wedge (y_i \oplus c_i). \tag{6}$$

If we make the notations $G_i = x_i y_i$ and $P_i = x_i \oplus y_i$, then we can rewrite the relation (6):

$$c_{i+1} = x_i y_i \oplus c_i (x_i \oplus y_i) = G_i \oplus c_i P_i . \tag{7}$$

The expression (7) can be decomposed as follows:

$$\begin{aligned} c_1 &= G_0 \\ c_2 &= G_1 \oplus c_1 P_1 = G_1 \oplus (G_0 P_1) \\ c_3 &= G_2 \oplus c_2 P_2 = G_2 \oplus (G_1 P_2) \oplus (G_0 P_1 P_2) \\ &\dots \\ c_{31} &= G_{30} \oplus c_{30} P_{30} = G_{30} \oplus (G_{29} P_{30}) \oplus (G_{28} P_{29} P_{30}) \oplus \cdots \oplus (G_0 P_1 P_2 \cdots P_{30}). \end{aligned} \tag{8}$$

In terms of consumed levels (consecutive multiplications) and total operations, the carry c_{31} is the most expansive element of the 32-bit adder. We see that the last term of c_{31} contains a total number of 30 multiplications but all of these are based on independent factors and thus can be implemented using a tree approach. In this way, the last term can be computed using a 5-depth circuit of homomorphic multiplications. The other terms are less expansive and can be computed using smaller circuits (depths of 1 to 5 levels). Finally, we can implement the entire homomorphic evaluation for c_{31} using circuits which consume maximum 5 levels. For other carries of the adder, the computation is analogue and does not exceed 5 levels. In conclusion, in terms of consumed levels, the homomorphic evaluation for the 32-bit adder can be implemented with our design in the limit of 5 levels.

Our solution eliminates the OR functions used by [13] for the initial expressions that define the c_i values. This is the reason which has led to a reduction of the homomorphic circuit for c_{31} from 10 levels (in their case) to 5 levels (in our case).

Using this approach, the total number of multiplications required to be computed in order to complete the homomorphic evaluation for c_{31} is $1+2\cdots+31=496$ multiplications, but these are not cumulative in terms of ciphertext noise. Relations described by (8) allow further optimization in terms of computation time. The values calculated for intermediate terms (shorter terms) of each c_i could be cached and reused for the computation of the longer terms of c_i or for the terms of the next carry c_{i+1}. In this way, the terms of c_i can be computed using programming techniques based on tree structures and having a set of the involved sub-trees already pre-computed in the previous steps. There is a penalty which regards the required memory with this approach, but the total number of multiplications will be lower and the computation time required by the homomorphic evaluation of the adder will be shorter.

4.2. FHE implementation of SPECK

For performance reasons, in what follows we focus on SPECK32/64. This variant has a block size of $2n = 32$ bits, a 64-bit secret key and a complete encryption flow needs 22 rounds. In this case, each round requires additions modulo 2^{16} bits. In the case of 16–bit adder and using the design of the corresponding circuit, described above, that means we need 4 levels for each round. In our representation, the other operations required by the SPECK rounds (bitwise XORs and bitwise rotations) are free in terms of levels consumption.

During the implementation we used a *bit-sliced* encoding of the plaintext/ciphertext. In this way, the *state* of a SPECK$2n$ instance (we see it as a $2n$-bits length array) is encrypted using a set of $2n$ ciphertexts. More exactly, the bits-array *state* $(x, y) = (x_0, x_1, \cdots, x_{n-1}, y_0, y_1, \cdots, y_{n-1})$ is represented (in FHE encrypted space) by $2n$ ciphertexts $(c_0, c_1, \cdots, c_{n-1}, c_n, \cdots, c_{2n-1})$, where each c_i encrypts the x_i bit and each c_{i+n} encrypts the y_i bit, for each $0 \le i < n$.

This type of representation allows us to implement in a easy way the operations involved during the SPECK rounds. The specific Feistel swapping can be achieved easily and with no costs. It consists in modifying the indexes of the ciphertext in encrypted *state* array, $c_i \leftrightarrow c_{i+n}$. The left and right rotations also consist in modifying the positions of ciphertexts in the working *state* array. All these permutations of the encrypted elements are free and not implies costly AND operations. The XORs operations with round keys or between the *state* halves are also free in terms of AND operations or consumed levels. The only operation that involves costs remains the n-bit addition. In terms of levels, it requires $log_2(n)$ levels. Thus, per total, in the case of SPECK32/64 (16-bit additions) we need 4 levels to complete an encryption round.

For our experimental part we used the last release of the HElib [17] library. This is a NTL-based C++ library that implements the BGV [3] leveled-FHE scheme. The HElib provides the basic FHE functionalities to perform our needed computations. We worked with Z_2 base and used XOR and AND implemented operations on FHE–ciphertexts. Based on these functionalities, we implemented the homomorphic evaluation of the 16-bit adder circuit using the form described above. In our implementation, we used also the proposed optimization based on the partial terms caching technique during the carries computations. This reduced the total number of multiplications from 605 (without caching) to 169 (with caching).

In our experiments the round keys were expanded also before, and then were FHE encrypted. We conducted tests using two configurations for BGV parameters. To keep a reasonable level of efficiency and security, for both configurations, we made a setup for the BGV scheme configured with a modulus chain allowing a maximum of 45 levels-depth computations ($L = 45$). With this setup in place, we have completed 11 from 22 rounds of the SPECK32/64. We presented the results in terms of timing and memory costs. The results (in terms of timing and memory costs) are presented in the *Table 1*.

Table 1

The SPECK evaluation costs

BGV params	**Sec(k)**	***l* slots**	**Enc Time**	**Time/block**	**RAM**
$m = 27303$, $\phi(m) = 21168$, $L = 45$	< 80 bits	756	**1772 sec**	2.34 sec	≈ **5.8 GB**
$m = 46063$, $\phi(m) = 45360$, $L = 45$	110 bits	1008	**3624 sec**	3.60 sec	≈ **15.6 GB**

In conclusion, the half of the SPECK encryption took us 3624 sec for $m = 46063$, $\phi(m) = 45360$ and $L = 45$ (these lead to a security level of 110 bits). In this setting, we have 1008 slots. Using the SIMD [7] capabilities of HElib we get an amortized time of 3.6 sec per SPECK block.

5. RELATED WORK

In the literature it was recently proposed a series of homomorphic evaluations [13, 18-22] over different primitives and cryptographic algorithms. Part of them focus on AES algorithm or other symmetric algorithms like lightweight ciphers.

In [18] is presented a homomorphic evaluation of AES–128 cipher. First version of this research was published in 2012 and updated then in 2015. The implementation is based on the BGV [3] scheme (*leveled*–FHE scheme) and was built up on HElib FHE–library [17]. During their work, the authors proposed multiple optimizations for the BGV scheme. The aim was to adjust the scheme and its related techniques (key-switching, modulus-switching) to the specific needs of the AES circuit. These optimizations are now included in the baseline version of the HElib, being considered as useful to evaluate other circuits too. The authors used both variants of the BGV scheme (with and without boostrapping) for evaluation. There were proposed three approaches for encoding the plaintext/ciphertext : *a) packed*, where a single ciphertext is used to encrypt the entire AES state matrix; *b) byte-sliced*, in this case 16 ciphertexts are used, each one corresponding to encryption of one byte (F_{2^8}) from the AES state matrix; and *c) bit-sliced*, which operates with 128 ciphertexts, each corresponding to a bit from the plaintext. In terms of circuit depth, each AES round is evaluated at a cost of 4 levels (the S-box lookup requires approximately 3.5 levels; other 0.5 levels being required by the Mix-Column transformation). In terms of time consumption, the complexity comes from key-switching operations (20 operations/round) which are required after each multiplication and automorphism (the automorphisms are free in terms of level consumption). The all 10 rounds of AES-128 are evaluated for a total cost of 40 levels. Packed implementation without boostrapping was tested using the parameters $m = 53261$, $\phi(m) = 46080$ (a equivalent security level of ≈ 150 bits). The homomorphic evaluation of the AES encryption function was completed in 252 seconds using an usual laptop and 3GB of RAM. Using the SIMD [7] processing technique they have been processing 120 AES blocks in parallel yielding an amortized rate of 2 seconds per block. Using the boostrapping and $m = 28679$, $\phi(m) = 23040$ (the security is roughly equivalent to 120 bits), the encryption of 180 AES blocks was accomplished in 18 minutes, which means about 6 seconds per block.

In [19] is presented also a *bit-sliced* homomorphic evaluation of the AES-128 circuit. The evaluation is based on a custom and optimized implementation of the ATV leveled-FHE encryption scheme [23] (based on the NTRU cryptosystem [24]). The authors evaluated homomorphically the full 10 rounds of the AES circuit (40 levels) in 31 hours working with 2048 message slots. This means an amortized evaluation time of 55 seconds per AES block.

In [20] is presented an evaluation of the PRINCE [25] *lightweight* block cipher. The leveled implementation is based on the NTRU cryptosystem. The work shows that the PRINCE encryption can be implemented using only 2 levels per round. Thus, to complete the 12 rounds of the algorithm, has been used a 24–level deep circuit. The batched implementation evaluated 1024 blocks in 57 minutes, with 3.3 seconds per block amortization. The authors make also a analysis regarding the depth of the circuits for other few *lightweight* ciphers (Present, HIGHT, SEA, KATAN-64, SIMON64/128, requiring circuits of 62, 96, 372, 254, 44 levels, respectively).

In [21, 22] there are homomorphic evaluations for different variants of SIMON [12] *lightweight* algorithm. In [21] is realized a *bit-sliced* implementation based on some FV and YASHE schemes. Using parameters that assure a security of 80 bits, a SIMON32/64 instance has been completed in 3062 seconds (FY) and 1029 seconds (YASHE) and having amortized times of 1.7 and 0.57 seconds per block, respectively. In [22] has been used the BGV leveled-FHE scheme and the HElib library. The resulted implementation evaluated all the 44 rounds of the SIMON64/128 (requires 1 level per round) in 112 minutes.

Other implementations (along with interesting considerations) on the homomorphic computations of cryptographic algorithms and primitives can be found in [13]. The authors take into discussion several algorithms (AES, SHA256, Salsa20, KECCAK). For each of them are presented various encoding techniques. For AES and KECCAK are presented implementations (these are based also on HElib) and experimental results in terms of timing costs and ciphertext sizes. This is probably the first implementation of this type for the KECCAK algorithm (SHA-3 winner). The SHA256 and Salsa20 algorithms involve 32-bits additions operations. The homomorphic circuit proposed in [13] for the addition has a depth of 10 levels. In these conditions, the total number of required levels is very high and leads to very inefficient implementations.

6. CONCLUSIONS

In this paper we presented a homomorphic evaluation of the SPECK *lightweight* encryption algorithm. From our best knowledge, this is the first such attempt for this algorithm. In the context of the homomorphic computations, the costliest operation of the SPECK's algorithm is the n-bits integer addition. In this regard, we proposed a reduced circuit (in terms of circuit depth) for this operation. In the case of 32-bits addition, we reduced a 10-depth circuit (proposed in [13]) to a 5-depth circuit. This result leads to the possibility of performing better homomorphic implementations for other applications which are based on the 32-bits integer additions. In the general case of n-bits integer additions and applying the design described in this paper, the n-bits additions can be homomorphically evaluated using $log_2(n)$-depth circuits. We proposed also some optimizations techniques that could be applied during the homomorphic computation processing for the adder circuit. These aim was to reduce the total number of multiplications which are expansive in terms of processing time. We made an implementation for the SPECK cipher rounds. For our implementation, we used the BGV [3] leveled–FHE scheme and HElib [17] library. The usage of leveled–FHE schemes removes the need of the *recrypt* operation specific to the bootstrapping techniques and could provide an acceptable degree of efficiency in the case of real applications. For performance reasons, we have chosen to implement the SPECK32/64 variant. This uses 16-bit integer additions, involving thus a consumption of 4 levels per round. To keep a reasonable level of efficiency and security, we made a setup in our work, for the BGV scheme configured with a modulus chain allowing a maximum of 45 levels-depth computations. With this setup in place, we have completed 11 from 22 rounds of the SPECK32/64. We presented the results in terms of timing and memory costs.

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. R. RIVEST, L. ADLEMAN, M. DERTOUZOS, *On Data Banks And Privacy Homomorphisms*, Foundations of Secure Computation, **4**, *11*, pp. 169–180, 1978.
2. C. GENTRY, *A Fully Homomorphic Encryption Scheme*, PhD Thesis, Stanford University, http://crypto .stanford.edu/craig, 2009.
3. Z. BRAKERSKI, C. GENTRY, V. VAIKUNTANATHAN, *Fully Homomorphic Encryption without Bootstrapping*, Innovations in Theoretical Computer Science Conference, pp. 309–325, 2012.
4. M. VDIJK, C. GENTRY, S. HALEVI, V. Vaıkuntanathan, *Fully Homomorphic Encryption Over The Integers*, Advances in Cryptology – Eurocrypt 2010, Lecture Notes in Computer Science, **6110**, pp. 24–43, 2010.
5. Z. BRAKERSKI, V. VAIKUNTANATHAN, *Efficient Fully Homomorphic Encryption From (Standard) LWE*, IEEE 52nd Annual Symposium on Foundations of Computer Science, pp. 97–106, 2011.
6. 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.
7. N.P. SMART, F. VERCAUTEREN, *Fully Homomorphic SIMD Operations*, IACR Cryptology ePrint Archive: Report 2011/133, http://eprint.iacr.org/2011/133.pdf, 2011.

8. 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.
9. 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.
10. 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.
11. M. NAEHRIG, K. LAUTER, V. VAIKUNTANATHAN, *Can homomorphic encryption be practical?*, CCSW '11 Proceedings of the 3rd ACM workshop on Cloud computing security workshop, pp. 113–124, ACM, 2011.
12. R. BEAULIEU, D. SHORS, J. SMITH, B. WEEKS, L. WINGERS, *The Simon and Speck Families of Lightweight Block Ciphers*, IACR Cryptology ePrint Archive: Report 2013/404, https://eprint.iacr.org/2013/404.pdf, 2013.
13. 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.
14. O. REGEV, On Lattices, Learning With Errors, *Random Linear Codes And Cryptography*, ACM Symposium On Theory of Computing, pp. 84–93, 2005.
15. 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.
16. C. PEIKERT, *Public-key Cryptosystems From The Worst-Case Shortest Vector Problem: Extended Abstract*, ACM Symposium on Theory of Computing, pp. 333–342, 2009.
17. S. HALEVI, V. SHOUP, *The HElib library*, https://github.com/shaih/HElib, 2015.
18. 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.
19. Y. DOROZ, Y. HU, B. SUNAR, *Homomorphic AES Evaluation Using NTRU*, IACR Cryptology ePrint Archive: Report 2014/039.
20. 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.
21. T. LEPOINT, M. NAEHRIG, *A Comparison of the Homomorphic Encryption Schemes FV and YASHE*, Progress in Cryptology – AFRICACRYPT 2014, Lecture Notes in Computer Science, **8469**, pp. 318–335, 2014.
22. B. CARMER, D.W. ARCHER, *Block Ciphers, Homomorphically*, Galois, Inc., 2014.
23. A. LOPEZ-ALT, E. TROMER, V. VAIKUNTANATHAN, *On-the-y multiparty computation on the cloud via multikey fully homomorphic encryption*, Proceedings of the 44th symposium on Theory of Computing, pp. 1219-1234, ACM, 2012.
24. J. HOFFSTEIN, J. PIPHER, J. SILVERMAN, *NTRU: A ring-based public key cryptosystem, Algorithmic number theory*, pp. 267–288, 1998.
25. J. BORGHOFF, A. CANTEAUT, T. GUNEYSU, E.B. KAVUN, M. KNEZEVIC, L.R. KNUDSEN, G. LEANDER, V. NIKOV, C. PAAR, C. RECHBERGER, P. ROMBOUTS, S.S. THOMSEN, T. YALCIN, *Prince – a low–latency block cipher for pervasive computing applications*, Progress in Cryptology – ASIACRYPT 2012, pp. 208–225, 2012.