Somewhat Homomorphic Encryption Scheme for Arithmetic Operations on Large Integers

Pedro Silveira Pisa*, Michel Abdalla†, and Otto Carlos Muniz Bandeira Duarte*
* Grupo de Teleinformatica e Automação, UFRJ - COPPE/PEE - Rio de Janeiro, RJ, Brazil
Email: {pisa, otto}@gta.ufrj.br
†Crypto Team - École Normale Supérieure & CNRS - Paris, France
Email: michelabdalla@ens.fr

***Abstract*—Homomorphic encryption allows for processing encrypted data without the need to decrypt them. This technique aggregates privacy and security for data processing in unreliable environments. In this paper, we propose a new encryption scheme oriented for calculating arithmetic functions of large integer numbers. Our proposal is an extension of the encryption scheme proposed in [1]. We also provide an implementation of both the original and the proposed scheme. Even though the total number of allowed homomorphic operations is limited in our scheme, our results show that the new extension is more efficient in terms of processing and that the ratio between the ciphertext and plaintext sizes is similar to that of the original scheme.**

I. Introduction

The concept of homomorphic encryption, which was first proposed by Rivest, Adleman, and Dertouzos [2] under the name of private homomorphism, emerged shortly after the invention of the RSA algorithm [3]. An encryption scheme is said to be homomorphic if it allows certain types of operations to be carried out on the encrypted data with the need to decrypt them. The RSA algorithm itself is multiplicatively homomorphic as it allows the multiplication of ciphertexts in order to obtain the ciphertext of the product. The validation of this property is due to RSA being based on exponentiation. Indeed, given a public key $K_{pub} = (N, e)$, we obtain the ciphertext Ψ by $\{\Psi \leftarrow \pi^e \bmod N\}$. Thus, k ciphertexts can be multiplied using $\prod_i^k \Psi_i = (\prod_i^k \pi_i)^e \bmod N$. The total value, however, cannot exceed N. After that, we recover the result using the private key K_{priv} for decrypting $\prod_i^k \pi_i$. Since RSA can only perform multiplication and not addition, it is only partially homomorphic. On the other hand, fully homomorphic encryption schemes allow both addition and multiplication. Since the emergence of the concept of homomorphism, it is estimated that the this technique would revolutionize privacy and security areas [4] for data processing in unreliable environments, such as cloud and grid computing. In this regard, Gentry's fully homomorphic encryption scheme [5] was the first one to prove the feasibility of this concept. His method, which is based on ideal lattices, is however quite complex and hard to understand, in addition to having extremely high computational requirements. Hence, Marten Van Dijk *et al.* [1] proposed a simple fully homomorphic encryption scheme based on modular arithmetic of integers[1]. Using the modular arithmetic, Van Dijk *et al.* simplified the method while preserving security.

In this paper, we propose a new encryption scheme oriented for calculating arithmetic functions of large integer numbers. Our proposal is an extension of the somewhat homomorphic encryption scheme proposed by Van Dijk *et al.* [1]. By somewhat homomorphic, we mean the scheme that limits the number of homomorphic operations. The DGHV scheme focuses on encryption and processing of bits, even though it is able to process small numbers, depending on the chosen primary key size. Every operation can be mapped to binary addition and multiplication, but our objective is to enhance the efficiency of the DGHV scheme for arithmetic operations over integers. Thus, in order to enable operations on integers of arbitrary size without converting them into bits, we introduce a new parameter called $BASE$, or B. This new parameter represents the maximum value the chosen key pair is able to encrypt. Therefore, the DGHV scheme is a particular case of our scheme, in which $B = 2$. Next, we provide an implementation of both the original scheme and the new extension and use it to compare the performance and security of the corresponding key generation algorithms. Finally, we reason about the semantic security of our scheme with respect to the assumptions of Van Dijk *et al.*.

The proposed method is evaluated by considering processing resources and ciphertext compactness. The processing resources are calculated from the duration of the execution. The ciphertext compactness degree of an encryption scheme is defined as the relation between plaintext and ciphertext sizes. Therefore, a compact method has the same size for plaintext and ciphertext. Both DGHV and our proposal ensure circuit privacy, which consists of hiding previous operations performed over current ciphertext. The circuit privacy is not relevant in traditional cryptography because the ciphertext has only been applied to the encryption algorithm. Nevertheless, in homomorphic encryption, different operations may be applied to the current ciphertext. Therefore, it is important to protect performed operations, since they may be secret.

The analysis also considers the λ parameter, which is

[1]Hereafter, in this paper, we name this scheme by DGHV, the initials of the authors

important for determining the security level of the system. As λ increases, the size of public and private keys, the size of the ciphertext, and the execution time increase. We perform a mathematical analysis for defining the correct limits of our proposal and, based on that, we present an upper-bond on the number of addition or multiplication operations. The proposed extension is more efficient for encrypting integers of arbitrary size than DGHV because we encrypt more than one bit each time. In DGHV, we decompose integers in bits and, then, encrypt only one bit per operation. Furthermore, applying addition or multiplication to ciphertext of multi-bit plaintext requires special addition and multiplication functions. On the other hand, in our proposal, we perform operations using standard functions, which provide better compatibility with current software and programming languages. In addition, considering memory and storage usage, there are similar relations between plaintext and ciphertext in both schemes. Our scheme requires, however, more storage to save the key pair as the B parameter grows. Therefore, we conclude that the proposed extension takes fully homomorphic encryption one step closer to being feasible, but a lot of work is still necessary in order to reduce the impact of growing λ for acceptable security levels.

This paper is organized as follows. Section II discusses related work. In Section III, we present the DGHV homomorphic encryption scheme and in Section IV we define our extension for calculating arithmetic functions of large integer numbers. Section V brings an analysis of the correctness of the proposed extension and Section VI presents the circuit size limitations of our scheme. In addition, Section VII evaluates and compares the proposal against the original scheme. Furthermore, Section VIII discusses the impact of the proposed changes on the security of the scheme and in Section IX, we analyze and indicate the viability of transforming our extension into a fully homomorphic scheme considering the DGVH techniques. Finally, conclusions and future work are presented in Section X.

II. Related Work

Gentry [5] made an important contribution to the homomorphic cryptography field by proving the possibility of conceiving a fully homomorphic system. Gentry's proposal allows the execution of arbitrary function over ciphertexts, obtaining the encryption of the desired result. Nevertheless, the security of the system relies on the noise added into the cyphered text. When performing an operation, the noise associated with the ciphertexts suffers the same operation. Thus, there is a practical limit on the number of operations that can be accomplished because the noise is cumulative and eventually makes it impossible to decrypt the answer. Based on this concept, Gentry proposed the application of the decryption algorithm over the ciphertext after each operation, by using an encryption of the private key. As the decryption function removes the noise of ciphertexts, the homomorphically utilization of the algorithm provokes the reduction of the noise. Based on this operation, named bootstrapping, Gentry allows the application of an infinite number of homomorphic operations over the ciphertext because its application reduces the noise level. Even with all the benefits, Gentry's proposal has high processing and storage requirements. In order to simplify the process, Van Dijk *et al.* proposed a homomorphic encryption scheme based on modular arithmetic of integers. Van Dijk *et al.* developed a simple and secure fully homomorphic encryption system that also utilizes the bootstrapping concept.

Levieil and Naccache [6] proposed a homomorphic technique with additive properties, aiming to prevent cheating on student tests. The system is not secure regardless of the choice of parameters, mainly because the authors considered a weak attacker model[2] when analyzing the security of their scheme as argued by Van Dijk *et al.* [1]. In 2004, the first Regev [7] encryption system was proposed, based on the unique shortest vector problem and with the encryption expression $ENC(K_{priv}, m) = K_{priv} * q + 2r + m$. Nevertheless, the private key is calculated using a relation between the domain space N and a secret h, $K_{priv} = N/h$. Unlike the Gentry and DGHV encryption schemes, the proposals by Levieil and Naccache and by Regev do not offer multiplicative homomorphism.

Because of the public key size, storage is one of the main challenges in homomorphic encryption. In the DGHV scheme, the public key size has order of λ^{10} bits and the private key size is λ^2, where λ is the security parameter. Coron *et al.* [8] proposed a modification in key generation in order to reduce the public key size to λ^7 bits. The modification is based on quadratic forms of the public key elements, instead of a linear form. Coron's proposal remains semantically secure because it is based on the partial approximate-GCD (Greatest Common Divisor) problem. This problem has the same security of approximate-GCD, which is the basis of DGHV. In addition, Coron *et al.* [9] presented a few additional optimizations for the public key size, reducing it to λ^5 bits. Both proposals are based on the DGHV scheme and focus on encryption of bits. Moreover, Smart and Vercauteren [10] proposed a new method for key generation and encryption, which reduces both ciphertext and public key size of the original Gentry's proposal. Furthermore, the authors argued that their proposal is able to encrypt N bits at once, unlike Gentry's proposal and in the same direction of our extension. Gentry's proposal, however, requires decomposing the number in bits and performing the operation on each bit. Nonetheless, our proposal focuses on directly computing integers over our variation of the DGHV scheme, which is simpler than Gentry's scheme.

Gentry and Halevi [11] presented an implementation of the Gentry's original scheme, including a bootstrapping operation. The authors analyzed the performance of their implementation, concerning the basic algorithms, the bootstrapping and the public key size. In order to improve the performance of the original Gentry's scheme, Brakerski e Vaikuntanathan [12] proposed an encryption scheme based on the standard learning

[2]Levieil and Naccache [6] even assume that the attackers do not know the mathematical base and the encryption algorithm. Therefore, the only alternative is using brute force over all other possibilities.

with errors (LWE) assumption. The security of LWE problems is based on the worst-case hardness of "short vector problems" on arbitrary lattices. This change of paradigm and the introduction of an operation for reducing vector dimension produced shorter ciphertexts, which enhanced decryption performance. Furthermore, these techniques led to another work by Brakerski *et al.* [13] that built a fully homomorphic encryption for operating on polynomial circuits without requiring the bootstrapping operations. In Brakerski *et al.*'s work, the security is based on the ring learning with errors (RLWE) assumption that was recently introduced by Lyubashevsky *et al.* [14].

III. DGHV Somewhat Homomorphic Encryption Scheme

A standard encryption scheme has three basic algorithms: *Key Generation*, *Encryption*, and *Decryption*. In a homomorphic scheme, we add the *Evaluation* algorithm, which aims to apply the desired operation over ciphertexts and obtain the ciphertext of the result. Although the correctness of a homomorphic encryption scheme is basically the same as any standard encryption scheme, proving its correctness requires decrypting a ciphertext produced by *Evaluation* algorithm. Therefore, the homomorphic encryption scheme is correct for any operation Φ whether the equation *Decryption* $(K_{priv},$ *Evaluation* $(K_{pub}, \Phi, \vec{\Psi})) = \Phi(m_1, m_2, \ldots, m_k)$ is valid [1], for any key pair generated by the *Key Generation* algorithm, any number of messages $m_1, m_2, \ldots, m_k$, and their corresponding ciphertexts $\vec{\Psi} =< \Psi_1, \Psi_2, \ldots, \Psi_k >$ encrypted by the *Encryption* algorithm. Hence, from this definition, this section describes the parameters and the four algorithms (*Key Generation*, *Encryption*, *Decryption*, and *Evaluation*) of the DGHV scheme, over which our extension is built.

A. Parameters Definition

The construction of the scheme makes use of five parameters, which depend polynomially of the security parameter λ. The first parameter η defines the size of the secret key in bits, the second parameter γ represents the public key elements, also in bits, and the third parameter τ is the number of public key elements. The last two parameters ρ and ρ' determine respectively the bit-length of the public key elements noise and the ciphertext noise. Van Dijk *et al.* [1] choose the value of η, γ, ρ, ρ', and τ parameters considering a series of attacks, such as brute force over the noise and approximate-GCD attacks. Hence, the following relations among these parameters and the security parameter (λ) are used: $\eta = \lambda^2$, $\gamma = \lambda^5$, $\rho = \lambda$, $\rho = 2\lambda$, and $\tau = \gamma + \lambda + 1$. Both ρ and ρ' parameters are chosen to be robust against brute-force on the noise. The η parameter is defined considering correctness of the scheme. γ and τ parameters are related to the approximate-GCD problem reduction. The γ parameter is chosen to prevent lattice-based attacks on the approximate-GCD problem, whereas the τ parameter is defined to apply the problem reduction[3].

[3]In our proposal, we consider τ as $\gamma+\lambda+\log_2 2$. In order to maintain the correctness of the approximate-GCD problem reduction, we must reformulate the logarithm as $\log 2\ B$.

B. Key Generation (λ)

The private key, K_{priv}, is an odd integer number, chosen randomly in range $\left[2^{\eta-1}, 2^{\eta}\right)$. The η parameter determines the private key size in bits. Thus, since $\eta = \lambda^2$ [1], the private key size increases polynomially as the security parameter λ grows.

For the public key K_{pub}, we sample elements $x_i = D_{\gamma,\rho}$, for $i = 0, \ldots, \tau$. The distribution $D_{\gamma,\rho}$ is given by $\{$*choose* $q \in [0, 2^{\gamma}/K_{priv})$ *and* $r \in (-2^{\rho}, 2^{\rho})$: *define* $x = K_{priv} \times q + 2r\}$. After that, we relabel x_i elements so that x_0 is the largest x_i. Furthermore, x_0 must be odd and the remainder of x_0/K_{priv} must be even. If x_0 does not comply with these constraints, we should restart the public key generation process. In the end, the public key is $K_{pub} =< x_0, x_1, \ldots, x_{\tau} >$, an array of integers.

C. Encryption (K_{pub}, m)

In order to encrypt a message $m \in \{0, 1\}$, we choose a random subset $S \subseteq \{1, 2, \ldots, \tau\}$, in which $\tau + 1$ represents the number of elements in the public key. We also choose an integer number in $(-2^{\rho'}, 2^{\rho'})$ interval. The ciphertext is then computed as $(m + 2r + \sum_{i \in S} x_i) \bmod x_0$.

D. Decryption (K_{priv}, Ψ)

Decryption is obtained by $m' = (\Psi \bmod K_{priv}) \bmod 2$. The remainder of the division of a ciphertext by the private key ($\Psi \bmod K_{priv}$) has the same parity of the encrypted bit because it eliminates all terms $K_{priv} \times q_i$. Therefore, the remainder of the division by 2 removes the encryption noise, since the terms of the sum $2\sum\ r_i$ are even and the noise of x_0 are also even due to its constraints. These facts afford the correct decryption because the only way to remove the encryption noise is by considering the parity.

E. Evaluation ($\Phi, < \Psi_1, \Psi_2, \ldots, \Psi_n >$)

Given a binary circuit operation Φ with n inputs and n ciphertexts Ψ_i, we build an algorithm that executes Φ over ciphertexts. For instance, to process an addition, we simply sum all the ciphertexts $< \Psi_1, \Psi_2, \ldots, \Psi_n >$.

IV. Proposed Extension

The DGHV scheme, described in Section III, is restricted to binary operations. Therefore, the DGHV scheme requires a higher number of operations for calculating an arithmetic function with larger integer numbers. In order to allow encryption of arbitrary size integers in one single operation, we propose the introduction of the B (BASE) parameter, which represents the quantity of numbers that the key pair is able to encrypt and decrypt. Thus, we generalize the parity concept used in DGHV for the divisibility concept. For that reason, it is required that the noise is a multiple of B instead of being even. In any case, the DGHV scheme is a particular case of our extension, in which $B = 2$. Analogously, when the original scheme requires odd numbers, we require numbers that are not multiple of B.

Note that as B increases, the proportion of non-multiple and multiple numbers by B grows. Because of that, there are more possibilities for choosing the private key K_{priv}, which increases the security over brute force. In the rest of this section, we present changes in the defined parameters and in the three basic algorithms. The *Evaluation* algorithm, however, remains the same.

A. Parameters Definition

In order to preserve the semantic security reduction of the scheme for the approximate-GCD problem, the majority of the parameters value is the same in our extension. The only exception is the τ parameter, which is redefined as $\tau = \lambda^5 + \lambda + log_2\ B$ aiming to assure the statistic indistinguishably of the ciphertext considering an attack that tries to predict the least-significant bit of the private key.

B. Key Generation (λ)

The private key K_{priv} is a number indivisible by B in $\left[B^{\eta-1}, B^{\eta}\right)$. The public key elements are calculated through the expression $x = K_{priv} \times q + r$, where $q \in [0, B^{\gamma}/K_{priv})$ and $r \in (-B^{\rho}, B^{\rho})$. These changes in q and r values are performed to ensure the same assumptions that were assumed in DGHV.

C. Encryption (K_{pub}, m)

For encrypting an integer $m \in [0, B)$, we obtain the same random subset S and a random number in $(-B^{\rho'}, B^{\rho'})$. Then, we apply the expression $\Psi = (m + B \times r + \sum_{i \in S} x_i) \bmod x_0$.

D. Decryption (K_{priv}, Ψ)

The decryption expression is $m' = (\Psi \bmod K_{priv}) \bmod B$.

V. Encryption Scheme Analysis

The decryption correctness is based on the divisibility of the remainder of the division of the ciphertext by the private key and, then, by B. Thereby, if we want to correctly decrypt, we must ensure that noise is divisible by B. Since it is based on modular arithmetic, if the sum of noise is greater than half of the private key, there is decryption error because the plaintext is added by the quotient of the division of noise by B.

Given the encryption expression of $m \in \{\mathbb{Z} \mid 0 \leq m \leq B\}$, $\Psi = (m + B \times r + \sum_{i \in S} x_i) \bmod x_0$, we obtain the ciphertext via

$$\Psi = m + B \times r + \sum_{i \in S} x_i - k \times x_0, \tag{1}$$

where k is the integer quotient of the division of $m + B \times r + \sum_{i \in S} x_i$ by x_0. As we know that $x_i = K_{priv} \times q_i + r_i$, it is possible to rewrite the ciphertext as

$$\Psi = m + Br + \sum_{i \in S} B \times r_i - k \times B \times r_0 + K_{priv}(\sum_{i \in S} q_i - k \times q_0). \tag{2}$$

Then, the decryption algorithm utilizes the expression

$$m' = (\Psi \bmod K_{priv}) \bmod B. \tag{3}$$

This way, the ciphertext, after the first division is given by

$$m' = m + Br + \sum_{i \in S} Br_i - kBr_0 - k' K_{priv}, \tag{4}$$

where k' is the integer quotient of the division of Ψ in Equation 2 by K_{priv}. Thus, in order to ensure the decryption correctness, all sum elements, except m, must be divisible by B. Thereby, in modulus operation, all noise is removed and we recover the plaintext m. Then, we analyze all terms of the sum, in which the elements Br, $\sum_{i \in S} Br_i$, and $k \times Br_0$ are clearly divisible by B. On the other hand, there is no constraint on the divisibility of $-k' * K_{priv}$ by B because the private key is not divisible by B and k' can be any value. Therefore, $-k' * K_{priv}$ is required to be zero, which happens if

$$m + Br + \sum_{i \in S} Br_i - kBr_0 < \frac{K_{priv}}{2}. \tag{5}$$

Then, we define the boundary condition for the correctness of the scheme in function of security parameter λ and B. In addition to that, the major noise constraint occurs when $\lambda = 2$ because the private key is smaller. The order of magnitude of encryption noise is given by

$$B \times \lceil B^{2\lambda} \rceil + iB \times \lceil B^{\lambda} \rceil - kB \times \lfloor -B^{\lambda} \rfloor, \tag{6}$$

where k has magnitude of $i \times B$ and i is the size of subset S, whereas the biggest value of the sum is achieved when all elements x_i are slightly smaller than x_0. The size of subset S is $\lceil \lambda^5 \rceil$ because S is the random subset of public key elements. Consequently, concerning only B and λ, our proposed extension is correct if

$$B^{2\lambda+1} + \lambda^5 B^{\lambda}(B^2 + B) < \frac{K_{priv}}{2}, \tag{7}$$

which defines the upper limit for noise. Therefore, if this inequation is not valid, there is decryption error.

VI. Maximum Circuit Size Allowed

Our proposed scheme has a limited number of operations that could be performed, because the ciphertext noise increases with the execution of each operation. In Section V, in order to correctly decrypt the ciphertext, we conclude the total noise must be shorter than half of private key, as defined in Equation 5. Hence, when we perform a sum $(A + B)$ or a multiplication $(A \times B)$, the aggregated noise is given by $R_A + R_B$ or $R_A \times R_B$ and it must be shorter than $\frac{K_{priv}}{2}$. For this reason, there is a limit in the number of operations that can be applied to ciphertexts.

In general, the specific limit of operations depends on the noise of each term and the operation to be performed. Nevertheless, to analyze the worst case scenario, we consider the maximum noise for each operation and we analyze

Table I
MAXIMUM NUMBER OF OPERATIONS FOR THE WORST CASE SCENARIO.

Base (B)	λ	K_{priv} (bytes)	K_{pub} (MBytes)	Additions	Multiplications
2	5	4	2	27	1
	10	13	1,193	1.03×10^{21}	3
	20	50	1,220,712	5.78×10^{106}	9
2^8	5	25	10	2.59×10^{33}	2
	10	100	9,538	8.91×10^{189}	4
	20	400	9,765,711	1.8×10^{864}	9
2^{16}	5	50	19	1.35×10^{67}	2
	10	200	19,076	1.59×10^{380}	4
	20	800	19,531,470	$6.53 \times 10^{1,728}$	9
2^{32}	5	100	38	3.63×10^{134}	2
	10	400	38,152	5.04×10^{760}	4
	20	1,600	39,063,135	$8.54 \times 10^{3,457}$	9

multiplication because it is more critical than additions for increasing noise. Therefore, the maximum number of operations is the largest d that satisfies $R^d < \frac{K_{priv}}{2}$, where R is the maximum acceptable noise. Through estimating noise magnitude (Equation 7), we are able to define the maximum acceptable noise R in terms of λ and B. Thus, we obtain

$$\left(B^{2\lambda+1} + \lambda^5 B^\lambda (B^2 + B)\right)^{d+1} < \frac{K_{priv}}{2}. \tag{8}$$

Analogous inequations are built for addition operations. In Table I, we present the maximum number of operations for the worst case scenario and include the public and private key size.

We observe, hence, that acceptable number of additions is quite larger than the acceptable multiplications. Moreover, as we increases B and λ, more operations can be performed. Nonetheless, the introduction of B parameter allows more operations with little increase in public and private key size. In the original DGHV scheme, for augmenting the number of operations, we have to increase λ, which implies greater growth for public key size. The number of multiplications, however, remains the same when B increases. Therefore, B parameter makes more additions possible and does not increase the number of multiplications.

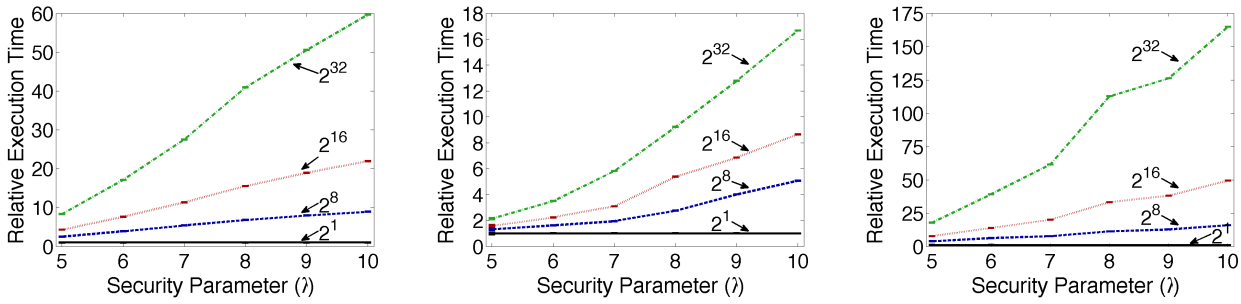
VII. EVALUATION OF OUR EXTENSION

We provide an implementation of both the original scheme [1] and the new extension. We also develop performance tests related to execution time and resource utilization. Both prototypes are implemented in Python, using standard random number module and GNU Multiple Precision (GMP) library for executing big numbers calculations. Practical evaluation is performed in an Intel Xeon X5570 @2.93 GHz computer with 96 GB of RAM. All results are obtained using a confidence level of 95%.

A. Results

We observe in Figure 1 a performance analysis based on execution time of the *Key Generation*, *Encryption*, and *Decryption* algorithms as a function of the security parameter λ. The evaluated λ values should not be used in practice due to lack of security. The parameter choice with security must ensure λ at least higher than 80 because, for $\lambda = 80$, we attain a security level similar to the one obtained in RSA for a 1024-bit key. The small values, however, were chosen to show performance tendencies under current computational conditions. The main problem with this approach is the public key storage, which reaches a magnitude of $O(\lambda^{10})$, demanding high memory or storage capacity. The curves express the different evaluated values for B and the curve in which $B = 2^1$ represents the DGHV method. The B values represent the quantity of values that can be encrypted with the given key set and are presented as base 2 exponents to allow a proper comparison with the basic numeric data types, such as byte, short and integer, with 8, 16, and 32 bit size, respectively.

The *Key Generation* algorithm has the highest computational cost because it requires the generation of λ^5 public key elements. Nonetheless, this cost does not cause issues because the algorithm is executed only once for lots of encryption, evaluation, and decryption. Throughout the results from Figure 1, we observe that the utilization of larger values for B implies in augmenting the execution time. This is a result of the operations with big numbers, which has higher memory access and processor utilization costs. This occurs mainly because the processor and the memory bus iterates over fixed-size words. We must consider, however, that with our proposal, more bits are encrypted at a time, which indicates better performance. In Figure 1(b), we verify that, for the security value of 7 and B base of 2^{32}, the execution time of the algorithm is 5.81 times higher, although we encrypt 32 times more bits per operation. This generates a result 5.51 times better than the original DGHV schema. For smaller λ (such as $lambda = 5$), we achieve a performance nearly 14.95 times better. Therefore, the execution time increase is shorter than the increase in number of bits. Moreover, we show that a linear growth in execution time happens with an exponential growth in plaintext size. On the other hand, according to Figure 1(c), *Decryption* algorithm loses performance as B grows because division operations are more complex with larger divisors. For instance, for $\lambda = 7$ and $B = 2^{32}$, the execution time is 61.54 times larger than DGHV. Nevertheless, we decrypt 32 times more bits, which represents a total performance loss of 1.92 times. Thus, if we consider the encryption gain and the decryption



(a) Key Generation. (b) Encryption. (c) Decryption.

Figure 1. Execution time of basic algorithms as a function of the security parameter λ for different B values. The presented values are relative to $B = 2^1$.

loss, our extension presents 2.87 better performance. Hence, if we consider more encryption operations than decryption ones, this enhancement is even greater. Indeed, when considering homomorphic encryption schemes, *Encryption* algorithm is used more frequently because we encrypt all input values and we decrypt only the result. Therefore, our proposed extension enhances the global performance.

Figure 2 presents an analysis considering ciphertext and public and private key size in function of the security parameter λ for different B values. Private key, K_{priv}, is chosen in $\left[2^{\lambda^2-1}, 2^{\lambda^2}\right)$. Thus, K_{priv} has deterministic size that depends on λ and B. The more λ and B, the more private key size. Figure 2(a) confirms this tendency by comparing private key size for different B. In this evaluation, we consider private key from 25 bits to 3200 bits, but this is still not practical for production environments. Testing larger private key size is restricted by memory usage, since the private key growth implies a huge public key size increase.

The public key, K_{pub}, grows in number and size of elements when λ and B grows. The increase in number of elements is deterministic and follows the security parameter and B. Considering *Key Generation* algorithm, we choose $\lambda^5 + \lambda + log_2 B$ elements for composing public keys. Thus, the larger the security parameter, the more elements the public key has. On the other hand, the size of public key elements depends on q parameter, that is randomly chosen in $[0, B^{\lambda^5}/K_{priv})$. As a consequence, evaluating size of public key elements requires a statistical approach. Since we generate K_{pub} elements by $x = K_{priv} * q + r$, the average size of each element has magnitude of $(B^{\lambda^5})/2$. Thus, the average size of public key elements increase together with λ and B. Figure 2(b) compares public key element size as B grows. For instance, we observe that for encrypting 32 bits at once, the *Key Generation* algorithm produces a public key 32 times larger. Therefore, likewise in the DGHV scheme one key pair is used for all bits and we need a larger key pair, our proposed extension utilizes more memory and storage resources.

Finally, Figure 2(c) presents average size of ciphertexts generated by our extension. All values are normalized by plaintext size in order to fairly measure the required storage per bit in all evaluated bases. In our evaluation test, we randomly generate plaintext in $[0, B-1]$. After encrypting plaintext, we divide ciphertext size by plaintext size in bits. In this result, we observe that, even though the plaintext increases, storage resources required for bit encrypted remains the same. Then, our extension provides the same compactness degree of DGHV, but we work with integers. It is observed, however, that, for lower bases, like $B = 8$, small size of plaintext means lower compactness degree, as we conclude in Figure 2(c) through the curve in which $B = 2^3$, that is shown above all others. Nevertheless, this increase in relation to ciphertext and plaintext size is lower than 30%, which is not critical because the main objective is to apply our proposal to larger bases. Indeed, for larger bases, the compactness degree is closer to the DGHV degree. Therefore, our extension does not worsen the compactness degree.

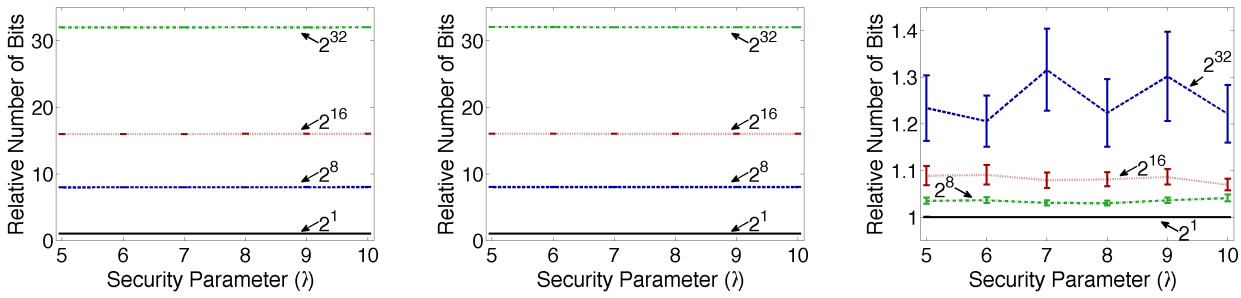
VIII. Proposed Scheme Security Basis

Considering attacks against an asymmetric encryption scheme, those which discover the private key from the public key and those which unveil some information about plaintext from the ciphertext are important attacks. The latter defines the semantically secure concept and is the strongest of the two. Therefore, we analyze the security basis of our extension under both attack models.

A. Private Key Recuperation

Private key grants access for all encrypted data. In RSA scheme, public and private keys are related through the modular multiplicative inverse, which makes it difficult to obtain private key from the public key. RSA scheme generates private key by multiplying two large prime numbers, which are quite trivial. Nonetheless, in order to recover this private key from the public key, we need to factorize the public key. This factorization requires high computational resources and it makes obtaining private key unfeasible.

The DGHV scheme generates public keys from choosing two random numbers q and r, according to the expression $x = K_{priv} \times q + r$. Thereby, obtaining private key requires the calculation of the greatest common divisor from approximate terms, which is very difficult to be solved. This problem is known by approximate-GCD and Howgrave-Graham [15]



(a) Private key size (b) Public key element size. (c) Ciphertext per bit

Figure 2. Memory resource utilization as a function of the security parameter λ for different B values. The presented values are relative to $B = 2^1$.

analyze it for two numbers. They prove the security of mechanisms based on the approximate-GCD problem. Because of this analysis, we define our parameters η, γ, ρ, ρ', and τ. Including noise is the key for security in this case. If there is no noise, the problem is as simple as calculating greatest common division (GCD). Because of the noise, private key discovering requires an exhaustive search for all noise space. Therefore, the more the introduced noise, the more the security of the scheme.

In the original DGHV [1], Van Dijk *et al.* adopt maximum noise of $\rho = 2^{\lambda}$, what implies an exhaustive search of $2^{4\lambda}$ for each evaluated pair of public key elements. The basic procedure needs to discover all possible greatest common divisors for each pair of elements taking into account all possible values for defining a set. This procedure is repeated others pairs of public key elements and the result is the intersection of the set of possible values. Therefore, we need to repeat the procedure until the intersection is a singleton.

In our extension, the maximum noise ρ is B^{λ}. Considering $B = 2$ the minimum base, the adopted noise is always larger than the DGHV noise. Indeed, in our extension, the noise size is $log_2\ B$ times larger, which increases the complexity of the problem. Formal analysis of how much security the noise growth enhances should be done in future work because we estimate that growing the noise is not required. Nonetheless, for security reasons, we adopt this larger noise in this proposal. Therefore, even though the increased security with larger noise is not formally analyzed, we assure that our extension does not reduce approximate-GCD problem complexity because the problem fundamentals are the same.

B. Semantic Security

Semantically secure cryptographic scheme means that two plaintext are completely indistinguishable after encrypting them with the scheme. In this analysis, we consider a game with a challenger and an attacker. Initially, the attacker receives the public key. Then, the attacker sends two different messages with the same size to the challenger, who chooses one to encrypt. After receiving the ciphertext, the attacker indicates which message generates the ciphertext and wins the game if he gets it right. Thus, the scheme is semantically secure if the probability of attacker wins the game is $1/2 + \epsilon$, for $\epsilon \approx 0$.

In the DGHV scheme, semantic secure is reduced for approximate-GCD problem. More specifically, Van Dijk *et al.* prove that an attacker A with advantage ϵ can be converted in an algorithm B to solve the problem with success probability of $\epsilon/2$, at least. A similar analysis can be performed for our proposal. Nevertheless, some modifications are required. First, size of η, γ, and ρ parameters must be increased by $log_2\ B$ factor because of using a greater base. Furthermore, the analysis must take into account the larger message space, which requires minor modifications in the procedure for learning the least significative bit of quotients of divisions of random integers by private key. In particular, we must increase τ by adding $log_2\ B$ in order to assure the statistical indistinguishability of ciphertext distribution in that procedure.

In our extension, once the approximate-GCD problem utilizes larger noise values, the semantic security tends to be higher. Nevertheless, to evaluate if there is a concrete security increase in our scheme, a formal analysis must to be done. This analysis includes formally reducing the scheme to the approximate-GCD problem. However, in spite of the absence of a formal analysis, we observe that the security level seems to be the same as DGHV because the basic principles of both schemes are the same.

IX. Fully Homomorphic Encryption

Although DHGV and our extension are homomorphic schemes for additions and multiplications, Section VI shows that they limit the number of operations to be performed. One solution for this issue it to adopt the Bootstrapping operation, proposed by Gentry [5]. Bootstrapping aims to reduce the ciphertext noise when it is required. The basic concept is to homomorphically decrypt and, then, homomorphically re-encryption the ciphertext. Homomorphic decryption is performed using the private key encryption and the re-encrypt utilizes another key pair. Once the decryption operation removes the ciphertext noise, the resulting noise depends on decryption circuit deep.

Bootstrapping operation is powerful because it enables any-deep circuits processing or, for our extension scenario, any-order polynomial functions. On the other hand, decryption al-

gorithm must be performed homomorphically, which requires that decryption circuit deep is shorter than the maximum acceptable deep in the scheme. Thus, bootstrapping operation is a special utilization of *Evaluate* algorithm, in which the evaluated function is the *Decryption* algorithm and the parameters are the ciphertext (Ψ) and the encryption of private key ($E(K_{priv})$). Therefore, the new ciphertext (Ψ') loses all original noise from Ψ, but there is new noise introduced by homomorphic operations over *Decryption* algorithm.

Nevertheless, for the DGHV scheme [1], decryption circuit requires more operations than the scheme accepts. For this reason, Van Dijk *et al.* propose changes to the scheme in order to reduce decryption circuit deep. Indeed, our extension increases decryption circuit complexity because it operates over larger numbers. Thus, the advantages of applying our modifications to Van Dijk *et al.*'s fully homomorphic scheme are not clear. In this paper, we analyze the maximum limit of operations the scheme accepts. In future works, however, we aim to formally analyze bootstrapping operation over our extension. This analysis focuses on verifying the feasibility of making our extension fully homomorphic.

X. Conclusion

Homomorphic encryption provides privacy and security for data processing in unreliable environments, such as cloud and grid computing. Gentry [5] shows the fully homomorphic feasibility and Van Dijk *et al.* [1] define a simple scheme. Nevertheless, both schemes are limited for processing 1 bit at time. This paper extends the DGHV scheme in order to enable operations with integers of arbitrary size without transforming the numbers into bits. We prove the correctness of our scheme by requiring total noise shorter than the half of private key. Thus, the shorter the private keys, the shorter the acceptable noise and, then, the less operations we can perform. We also conclude that multiplication is the most critical operation and the main reason for limiting circuit deep. For instance, considering $\lambda = 20$ and $B = 2^{32}$, we perform 9 multiplications or less. At the same time, we can perform more than $10^{3,457}$ additions.

We evaluate our extension based on performance and memory requirements. Considering performance, the enhancement of encryption process is greater than the deterioration of decryption process. Therefore, our extension presents a global performance improvement. For instance, comparing with the DGHV scheme, global performance is 2.87 times better for private keys with 1,568 bits ($B = 2^{32}$ and $\lambda = 7$). On the other hand, in our extension, storing public and private keys requires more resources because we increase the random intervals by log_B. Thus, we increase scheme security against brute force attacks. Regarding the approximate-GCD problem, our extension maintains the same basic principles for reduction to the problem. Therefore, we assure the infeasibility of obtaining the private key from the public key.

In a practical evaluation, we conclude that scheme compactness is kept because the ciphertext size per bit is approximately the same in both DGHV and our extension schemes. This size, however, is larger in some scenarios with lower base, such as $B = 8$. Nevertheless, in those scenarios, our extension is not indicated because the small numbers are easily processed one bit at once.

Finally, our results show that memory requirements for public keys and ciphertexts are as larger as the ones required by DGHV. Thus, in future work, we intend to enhance scheme compactness, which will reduce public key and ciphertext size. For deploying homomorphic encryption schemes in unreliable scenarios, such as cloud and grid computing, we must adopt this approach. Furthermore, our future study includes a formal analysis of the scheme security considering impacts due to base changes. We also include an analysis of bootstrapping operation and limits in circuit deep. Therefore, we aim to formally define security in these scenarios with larger bases.

Acknowledgment

CNPq, FINEP, FUNTTEL, CAPES, FAPERJ, and UOL supported this work. This work was also supported by the European Commission through the FP7-ICT-2011-EU-Brazil Program under Contract 288349 SecFuNet. We thank the anonymous reviewers for their comments.

References

[1] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," *EUROCRYPT 2010*, pp. 24–43, 2010.

[2] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *CACM*, vol. 21, no. 2, pp. 120–126, 1978.

[3] R. Rivest, L. Adleman, and M. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of secure computation*, pp. 169–179, 1978.

[4] V. Vaikuntanathan, "Computing blindfolded: New developments in fully homomorphic encryption," in *FOCS 2011*, 2011, pp. 5–16.

[5] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC 2009*. ACM, 2009, pp. 169–178.

[6] E. Levieil and D. Naccache, "Cryptographic test correction," *PKC 2008*, pp. 85–100, 2008.

[7] O. Regev, "New lattice-based cryptographic constructions," *JACM*, vol. 51, no. 6, pp. 899–942, 2004.

[8] J. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," *CRYPTO 2011*, pp. 487–504, 2011.

[9] J. Coron, D. Naccache, and M. Tibouchi, "Optimization of fully homomorphic encryption," 2011. [Online]. Available: http://eprint.iacr.org

[10] N. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," *PKC 2010*, pp. 420–443, 2010.

[11] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," *EUROCRYPT 2011*, pp. 129–148, 2011.

[12] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *FOCS 2011*, 2011, pp. 97–106.

[13] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," *ITCS 2012*, 2012.

[14] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *Advances in Cryptology–EUROCRYPT 2010*, pp. 1–23, 2010.

[15] N. Howgrave-Graham, "Approximate integer common divisors," *Cryptography and Lattices*, pp. 51–66, 2001.