Accepted Manuscript

A methodical FHE-based cloud computing model

Paulo Martins, Leonel Sousa

PII: S0167-739X(18)31681-9
DOI: https://doi.org/10.1016/j.future.2019.01.046
Reference: FUTURE 4739

To appear in: *Future Generation Computer Systems*

Received date : 16 July 2018
Revised date : 11 December 2018
Accepted date : 22 January 2019

Please cite this article as: P. Martins and L. Sousa, A methodical FHE-based cloud computing model, *Future Generation Computer Systems* (2019), https://doi.org/10.1016/j.future.2019.01.046

A Methodical FHE-based Cloud Computing Model

Paulo Martins[a], Leonel Sousa[a]

[a] *INESC-ID, Instituto Superior Técnico, Universidade de Lisboa*
Rua Alves Redol, 9
1000-029 Lisboa, Portugal
Declaration of interests: none

Abstract

Attacks such as Meldown and Spectre have shown that traditional cloud computing isolation mechanisms are not sufficient to guarantee the confidentiality of processed data. With Fully Homomorphic Encryption (FHE), data may be processed encrypted in the cloud, making any leaked information look random to an attacker. Furthermore, a client might also be interested in protecting the processing algorithm. While there has been research on ensuring the confidentiality of the processing algorithm, the resulting systems are impractical. Herein, we propose an automatic and methodical technique to approximate a wide range of functions homomorphically. As the approximations are all evaluated in the same manner, a homomorphic evaluator has no way to distinguish them. Since the derivation of the FHE circuit is decoupled from the function development process, users benefit from traditional programming and debugging tools. The proposed tools may exploit different kinds of number representations during the homomorphic evaluation of functions, namely stochastic number representations and fixed-point arithmetic, each with its own characteristics. Additionally, an implementation of the system is presented, its applicability is verified in practice for commonly used applications, including image processing and machine learning, and the two number representations are thoroughly compared.

Keywords: Homomorphic Encryption, Computer Arithmetic, Cloud Computing

1. Introduction

Cloud computing has improved the availability of computational resources and the efficiency of their usage. Increasing computational power permits the same machine to be used by several users and corporations simultaneously. This leads to a reduction in infrastructure costs, and allows for corporations to more quickly adjust the contracted computational power to the market needs. The

Email addresses: `paulo.sergio@netcabo.pt` (Paulo Martins), `las@inesc-id.pt` (Leonel Sousa)

shared nature of computing resources in the cloud infrastructure introduces new security challenges [1], namely in what concerns the confidentiality of the offloaded data. Typical cloud architectures ensure the confidentiality of data while it is being transferred through encryption [2], and through the hypervisor separation of the guests' virtual machines when it is being processed [3]. However, attacks such as Meltdown and Spectre [4, 5] have shown that the scope of the hypervisor separation might not be enough to prevent the disclosure of sensitive data. In particular, while the hypervisor guarantees isolation at a software level, execution in a processing element leaves traces in shared hardware elements such as cache memories and branch predictors that may be exploited by other processes to extract sensitive data. Even the hardware-backed separation of virtual machines has been shown to be vulnerable to this type of attacks [6].

FHE is a technique uncovered in 2009 [7] that enables the processing of encrypted data [8]. If FHE is employed in the context of cloud computing, attacks such as Meltdown and Spectre [4, 5] are rendered useless. All data leaked through side-channel attacks will be in an encrypted form and will therefore look random to an attacker. Two main lines of research might be identified in the area of FHE-based cloud computing models. In one, an entire computer architecture is emulated with homomorphic operations [9]. In particular, memory cells are stored as ciphertexts. Traditional logic to access memory banks is afterwards translated into their homomorphic counterparts, and memory addresses are themselves encrypted. After loading a ciphertext from memory, it can be processed with a homomorphic Arithmetic and Logic Unit (ALU). The ALU is controlled by encrypted data. A combination of the previous features enables the implementation of control flow instructions. The Program Counter (PC) is treated as another encrypted register, which can be written to to implement `goto` and `if` instructions, and is used to load instructions from memory. In the other line of research, the algorithm one wants to compute is disclosed to the cloud. In this case, optimised operations and parameters can be specifically developed for the targeted application [10, 11, 12].

While a completely homomorphic computing system provides the maximum possible privacy, it has been found not to be practical [9]. Moreover, since the evaluator has no access to the instructions, it is hard to know when to stop the computation. While dedicated homomorphic circuits are the most practical, they are also the least private of the two; and might prove difficult to develop, since the users need to be aware of many low-level details of FHE. In this paper, we propose a computational framework, depicted in Figure 1, that draws from the benefits of both approaches, achieving both strong privacy and practical performance. A wide range of functions are approximated in a generic way, producing an encrypted description of them. In particular, we focus on a wide range of functions whose approximation can be efficiently evaluated in a homomorphic manner, namely multivariate continuous functions, which might even be non-analytic, thus going beyond traditional Taylor-series based approaches. This type of functions is applied in fields such as machine learning and image processing. When processing the encrypted function descriptions, the homo-

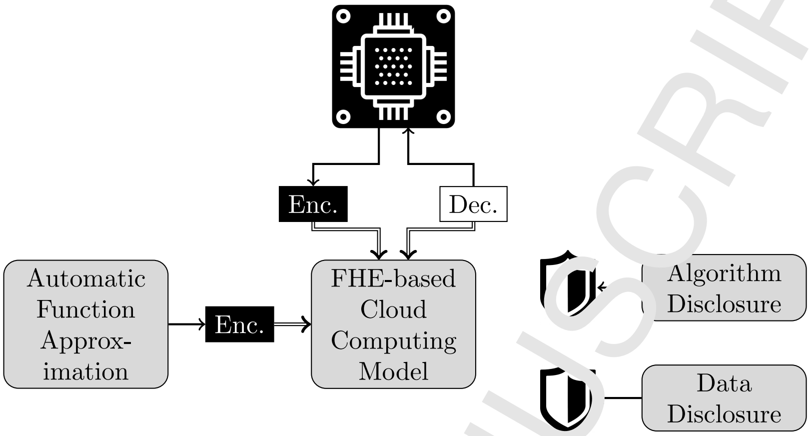

Figure 1: Operation of the proposed FHE-based cloud computing model. Double lines represent encrypted values. Shields represent blocked attack vectors

morphic evaluator will not be able to infer anything from the function except for how precisely it is being approximated. This is embodied in Figure 1 as the blocking of algorithm disclosure. By having a larger number of functions that can be supported, security is being improved, since the amount of possibilities for the function that is being processed is also large. Moreover, since the algorithm inputs are encrypted, data disclosure is also prevented.

The translation from the user code to the encrypted approximated description treats the function in a black box approach, which means that the user does not need to change his typical development flow, and that he or she can call other functions, use complex control flows, etc. Furthermore, the encrypted function approximation can target one of two number representations: a stochastic one [13] or another one supported on a variant of fixed-point arithmetic [14]. Stochastic representations consist of sequences of bits, where a number is represented by the frequency of bits equal to '1'. With batching, one is able to encrypt a stochastic sequence of bits in a single cryptogram. With the variant of fixed-point arithmetic, after each homomorphic multiplication, numbers are scaled down so that the amount of bits necessary to represent them remains approximately constant, allowing for the selection of efficient scheme parameters [14].

The rest of this paper is organised as follows. After a brief introduction to FHE and function approximation in Section 2, the proposed formulations and algorithms are presented in Section 3. An implementation of the proposed method is discussed in Section 4. Experimental evaluations are also carried out in Section 4. We not only evaluate the performance of the proposed cryptographic operations in a general purpose processor, but since the device producing and consuming data in Figure 1 may have limited computational power, we also evaluate the performance of encryption and decryption in an embedded system. A comparison with related art is the focus of Section 5. Section 6 concludes this paper.

2. Background

The techniques used in Section 3 to develop a confidential cloud computing model are herein introduced. Homomorphic encryption arithmetic and polynomial approximations of functions are discussed. These techniques will later serve as the basis to homomorphically evaluate multivariate continuous functions, which might be non-analytic.

2.1. Homomorphic Encryption

While other cryptosystems could be considered for the implementation of the techniques described herein, in this section the Brakerski-Gentry-Vaikuntanathan (BGV) [15] cryptosystem will be described to support the proposed FHE approach. This choice was mainly motivated by the availability of mature libraries for implementing the proposed system [16, 17].

$\boldsymbol{\phi}_m(X) \in \mathbb{Z}[X]$ is used to denote the m-th cyclotomic polynomial of degree $n = \varphi(m)$, where φ is Euler's totient function. The ring $\mathcal{R} = \mathbb{Z}[X]/(\boldsymbol{\phi}_m(X))$ is the main structure of BGV. An element of $\mathcal{R}$ can be thought of as a polynomial with integer coefficients and a degree strictly smaller than n. The underlying space for ciphertexts is $\mathcal{R}_q = \mathcal{R}/q\mathcal{R} = \mathbb{Z}/q\mathbb{Z}[X]/(\boldsymbol{\phi}_m(X))$, which is composed of elements of $\mathcal{R}$ with coefficients reduced modulo q. Herein, the notation $[\cdot]_q$ denotes the centred residue modulo q in $[-q/2, q/2)$, while $\lfloor\cdot\rceil$ denotes rounding to the nearest integer.

In the context of the BGV scheme, the secret-key $\boldsymbol{s} \in \mathcal{R}$ is defined as a "small" polynomial drawn from a distribution χ_{key}. An encryption of $\boldsymbol{m} \in \mathcal{R}_t$, for $t \in \mathbb{N}$, corresponds to a pair of polynomials $\mathtt{ct} = (\boldsymbol{c_0}, \boldsymbol{c_1}) \in \mathcal{R}_q^2$ satisfying:

$$[\boldsymbol{c_0} + \boldsymbol{c_1}\boldsymbol{s}]_q = [[\boldsymbol{m}]_t + t\boldsymbol{v}]_q \tag{1}$$

where $\boldsymbol{v}$ is a noise term that is originally introduced during encryption (which is related to a distribution χ_{err}) and that grows as homomorphic operations are applied. Decryption operates correctly so long as this noise is below a certain bound, which limits the amount of homomorphic operations one can perform.

BGV provides for the homomorphic addition and multiplication of polynomials in $\mathcal{R}_q$. The homomorphic addition of two ciphertexts corresponds to the pairwise addition of the ciphertexts' polynomials. Regarding homomorphic multiplication, it is useful to see ciphertexts as first degree polynomials with coefficients in $\mathcal{R}$. For a polynomial $\boldsymbol{c_0} + \boldsymbol{c_1}\boldsymbol{y}$, evaluating it at $\boldsymbol{y} = \boldsymbol{s}$ would lead to (1). In this context, the homomorphic multiplication of $\mathtt{ct}^1$ and $\mathtt{ct}^2$ takes place in two steps. First, $\mathtt{ct}_{mult} \leftarrow \left(\left[\boldsymbol{c}_0^1\boldsymbol{c}_0^2\right]_q, \left[\boldsymbol{c}_0^1\boldsymbol{c}_1^2 + \boldsymbol{c}_1^1\boldsymbol{c}_0^2\right]_q, \left[\boldsymbol{c}_1^1\boldsymbol{c}_1^2\right]_q\right)$ is computed. Evaluating $\mathtt{ct}_{mult,0} + \mathtt{ct}_{mult,1}\boldsymbol{y} + \mathtt{ct}_{mult,1}\boldsymbol{y}^2$ at $\boldsymbol{y} = \boldsymbol{s}$ would also enable decryption. In order to prevent the continuous growth of the number of elements in ciphertexts, one has to convert the three-element ciphertext back to a two-element ciphertext, through a process called *relinearisation*. In a nutshell, $\mathtt{ct}_{mult,2}$ is multiplied by a pseudo-encryption of $\boldsymbol{s}^2$ and the result is added to $(\mathtt{ct}_{mult,0}, \mathtt{ct}_{mult,1})$ [15].

Finally, a noise management technique is applied to reduce the growth rate of the norm of $\boldsymbol{v}$ in (1) due to the homomorphic multiplication. This technique is called modulus-switching and consists of scaling the ciphertext to a smaller ring $R_{q'}$ with an appropriate rounding, which is performed in two steps:

$$\begin{aligned} \boldsymbol{\delta}_i &\leftarrow t \cdot [-\mathtt{ct}_{\mathtt{mult},i}/t]_{q/q'} \text{ for } i = 0, 1 \\ \mathtt{ct} &\leftarrow \Big([q'/q \cdot (\mathtt{ct}_{\mathtt{mult},0} + \boldsymbol{\delta}_0)]_q \\ &\qquad [q'/q \cdot (\mathtt{ct}_{\mathtt{mult},1} + \boldsymbol{\delta}_1)]_{q'} \Big) \end{aligned} \tag{2}$$

2.2. Homomorphic Arithmetic

In [13], a system was proposed for homomorphic processing exploiting stochastic number representations. A stochastic representation of a number $x \in [0, 1]$ is defined to be a sequence of n bits, $x_1, \ldots, x_n$ drawn from a Bernoulli distribution, such that the probability $P(x_i = 1) = x, \forall_{1 \leq i \leq n}$ [18]. Batching [19] is therein exploited to encrypt multiple bits in a single ciphertext, so that one can AND and XOR the bits of two stochastic representations homomorphically. While batching can be employed in many cryptosystems [19, 20, 21], we exemplify how it can be implemented in BGV. A binary plaintext space has the following structure:

$$\mathcal{P} = \mathbb{Z}[X]/(\boldsymbol{\phi}_m(X), 2) \tag{3}$$

$\boldsymbol{\phi}_m$ factors modulo 2 into l polynomials $F_0, \ldots, F_{l-1}$ with the same degree d. Batching consists in exploiting this factorisation to encrypt multiple bits in a single ciphertext so that additions and multiplications operate on them in parallel. To do so, bits $m_0, \ldots, m_{l-1}$ are encoded as a polynomial $m(x)$ satisfying:

$$m_i = m(x) \bmod (F_i(x), 2)\, \forall_{0 \leq i < l} \tag{4}$$

Finally, rotations of the plaintext slots can be obtained through mappings of the form $\kappa_i : X \to X^i$ [22].

We highlight the following two stochastic operations, which can be computed by composing the aforementioned operations, for three independent stochastic representations of $x, y, s \in [0, 1]$ (note that when two representations are not statistically independent, they can be rotated to become so):

$$z_i = x_i \wedge y_i \qquad \Rightarrow z = xy \tag{5}$$

$$z_i = ((1 \oplus s_i) \wedge x_i) \oplus (s_i \wedge y_i) \quad \Rightarrow \quad z = (1 - s)x + sy \tag{6}$$

where $\wedge$ and $\oplus$ stand for the AND and XOR operations, respectively.

Most FHE schemes include a form of randomisation by adding noise to encrypted messages. In [14], noise is conceived as being part of a noisy representation of numbers. For example, the BGV cryptosystem can be modified so as to support messages in $\mathcal{R}$, with the following decryption operation:

$$[\boldsymbol{c_0} + \boldsymbol{c_1}\boldsymbol{s}]_q = [\boldsymbol{m} + \boldsymbol{v}]_q \tag{7}$$

With BGV one applies modulus-switching to control the noise growth due to homomorphic multiplications without affecting the underlying plaintext. This is possible due to the rounding operation described in Section 2.1. [14] suggests instead that both the ciphertext modulus and the plaintext are scaled down by the same amount when modulus switching is applied, through an operation called rescaling that replaces (2) by (8).

$$\begin{aligned} \mathtt{ct} \quad \leftarrow \quad & \Big([\lfloor q'/q \cdot \mathtt{ct}_{\mathtt{mult},0} \rceil]_{q'}, \\ & [\lfloor q'/q \cdot \mathtt{ct}_{\mathtt{mult},1} \rceil]_{q'} \Big) \end{aligned} \tag{8}$$

The arithmetic techniques in [14] mimic fixed-point arithmetic. A number $x \in \mathbb{R}$ is represented as a polynomial:

$$\boldsymbol{x} = \lfloor \Delta x \rceil + \boldsymbol{v} \tag{9}$$

where $\Delta \approx q/q'$ is a scale factor and $\boldsymbol{v}$ corresponds to the noise described in (7). It is clear that the format in (9) is preserved after additions. After a homomorphic multiplication $\boldsymbol{z} = \boldsymbol{x}\boldsymbol{y}$, $\boldsymbol{z}$ is scaled down by a factor of Δ due to rescaling, such that (9) is preserved.

2.3. Approximating Functions with Polynomials

While one could produce a function-approximating polynomial using interpolation techniques, such as Lagrange's, there is no generic choice of n points for which these polynomials converge uniformly to the function as $n \to \infty$. In contrast, Bernstein polynomials achieve that [23]. Let $n_1, \ldots, n_m \in \mathbb{N}$ and f be a function of m variables. The polynomial $B_f^{(n_1,\ldots,n_m)}(x_1, \ldots, x_m)$ is called the Bernstein polynomial of f:

$$\beta_{f,k_1,\ldots,k_m}^{(n_1,\ldots,n_m)} = f\left(\frac{k_1}{n_1}, \ldots, \frac{k_m}{n_m}\right) \tag{10}$$

$$B_f^{(n_1,\ldots,n_m)}(x_1, \ldots, x_m) = \sum_{\substack{0 \le k_l \le n_l \\ l \in \{1,\ldots,m\}}} \beta_{f,k_1,\ldots,k_m}^{(n_1,\ldots,n_m)} \prod_{j=1}^{m} \binom{n_j}{k_j} x_j^{k_j} (1 - x_j)^{n_j - k_j} \tag{11}$$

If $f : [0, 1]^m \to \mathbb{R}$ is a continuous function, then $B_f^{(n_1,\ldots,n_m)}$ converges uniformly to f as $n_1, \ldots, n_m \to \infty$ [23].

2.4. Polynomial Evaluation

[13] and [14] respectively propose stochastic and fixed-point homomorphic operations that suffice for the evaluation of univariate polynomials. De Casteljau's algorithm and Horner's method are respectively used in [13] and [14] for the evaluation of polynomials, as described in Algorithms 1 and 2. While the first exploits a Bernstein representation of polynomials, the second makes use of the power form.

Algorithm 1 De Casteljau's algorithm for the evaluation of a polynomial in Bernstein form [24]

Require: $B(x) = \sum_{i=0}^{d} \binom{d}{i} b_i x^i (1-x)^{d-i}$
Require: x_0
for $i \in \{0, \ldots, d\}$ **do**
  $b_i^{(0)} := b_i$
end for
for $j \in \{1, \ldots, d\}$ **do**
  for $i \in \{0, \ldots, d-j\}$ **do**
    $b_i^{(j)} := b_i^{(j-1)}(1-x_0) + b_{i+1}^{(j-1)} x_0$
  end for
end for
return $B(x_0) = b_0^{(d)}$

Algorithm 2 Horner's method for the evaluation of a polynomial in power form [25]

Require: $P(x) = \sum_{i=0}^{d} a_i x^i$
Require: x_0
$s := a_d$
for $i \in \{d-1, \ldots, 0\}$ **do**
  $s := a_i + x_0 s$
end for
return $P(x_0) = s$

3. Proposed FHE-based Cloud Computing System

The proposed system for homomorphically approximating user-provided continuous functions follows a black-box approach: in a first offline phase, it evaluates the functions at the points in (10), and when online computes (11) homomorphically. Algorithms 1 and 2 evaluate univariate polynomials in Bernstein and power form, respectively, using the basic operations provided by [13, 14]. In order to evaluate (11) homomorphically, one first needs to generalise Algorithms 1 and 2 to the homomorphic multivariate case, and provide methods to convert polynomials in the Bernstein form to the power form.

3.1. Homomorphic Evaluation of Multivariate Polynomials

The strategy herein proposed to generalise Algorithms 1 and 2 consists of iteratively reducing the problem of evaluating a polynomial in m variables to the problem of evaluating several polynomials in $m-1$ variables and afterwards combining the results, until constant polynomials are reached. As it will be shown, the step to combine the results of the several evaluations of polynomials with fewer variables can be performed with the base algorithms. For polynomials in the Bernstein form, we first rewrite (11) as:

$$B_f^{(n_1,\ldots,n_m)}(x_1,\ldots,x_m) = \sum_{k_1=0}^{n_1}\binom{n_1}{k_1}x_1^{k_1}(1-x_1)^{n_1-k_1}\left(\sum_{k_2=0}^{n_2}\binom{n_2}{k_2}x_2^{k_2}(1-x_2)^{n_2-k_2}\right.$$
$$\left.\ldots\left(\sum_{k_m=0}^{n_m}\beta_{f,k_1,\ldots,k_m}^{(n_1,\ldots,n_m)}\binom{n_m}{k_m}x_m^{k_m}(1-x_m)^{n_m-k_m}\right)\ldots\right) \quad (12)$$

The i^{th} parenthesised expression in (12) can be computed through the following recursive function:

$$g_{f,k_1,\ldots,k_{i-1}}^{(n_1,\ldots,n_m)}(x_i,\ldots,x_m) = \sum_{k_i=0}^{n_i}\binom{n_i}{k_i}x_i^{k_i}(1-x_i)^{n_i-k_i}g_{f,k_1,\ldots,k_i}^{(n_1,\ldots,n_m)}(x_{i+1},\ldots,x_m) \quad (13)$$

with the base case:

$$g_{f,k_1,\ldots,k_m}^{(n_1,\ldots,n_m)}() = \beta_{f,k_1,\ldots,k_m}^{(n_1,\ldots,n_m)} \quad (14)$$

As [13] can homomorphically evaluate Algorithm 1, it can also support the evaluation of:

$$B_f^{(n_1,\ldots,n_m)}(x_1,\ldots,x_m) = g_f^{(n_1,\ldots,n_m)}(x_1,\ldots,x_m) \quad (15)$$

since Algorithm 1 computes (13) by setting $b_i = g_{f,k_1,\ldots,k_i}^{(n_1,\ldots,n_m)}(x_{i+1},\ldots,x_m)$ and $x_0 = x_i$.

A multivariate polynomial in power form can be described as follows:

$$P(x_1,\ldots,x_m) = \sum_{\substack{0\le k_l\le n_l\\ l\in\{1,\ldots,m\}}}\alpha_{k_1,\ldots,k_m}^{(n_1,\ldots,n_m)}\prod_{j=1}^{m}x_j^{k_j} \quad (16)$$

The expression in (16) can be factorised as:

$$P(x_1,\ldots,x_m) = \sum_{k_1=0}^{n_1}x_1^{k_1}\left(\sum_{k_2=0}^{n_2}x_1^{k_2}\ldots\left(\sum_{k_m=0}^{n_m}\alpha_{k_1,\ldots,k_m}^{(n_1,\ldots,n_m)}x_m^{k_m}\right)\ldots\right) \quad (17)$$

As before, each parenthesised expression can be computed through a recursive function:

$$h_{k_1,\ldots,k_{i-1}}^{(n_1,\ldots,n_m)}(x_i,\ldots,x_m) = \sum_{k_i=0}^{n_i} x_i^{k_i} h_{k_1,\ldots,k_i}^{(n_1,\ldots,n_m)}(x_{i+1},\ldots,x_m) \tag{18}$$

with the following base case:

$$h_{k_1,\ldots,k_m}^{(n_1,\ldots,n_m)}() = \alpha_{k_1,\ldots,k_m}^{(n_1,\ldots,n_m)} \tag{19}$$

In addition, since [14] homomorphically evaluates Algorithm 2, it can also compute

$$P(x_1,\ldots,x_m) = h^{(n_1,\ldots,n_m)}(x_1,\ldots,x_m) \tag{20}$$

since Algorithm 2 evaluates (18) by setting $a_i = h_{k_1,\ldots,k_i}^{(n_1,\ldots,n_m)}(x_{i+1},\ldots,x_m)$ and $x_0 = x_i$.

3.2. Conversion between Bernstein and Power Forms

Univariate power polynomials can be written in terms of Bernstein polynomials through the formula provided in [26]:

$$x^j = \sum_{k=j}^{n} \frac{\binom{k}{j}}{\binom{n}{j}} \binom{n}{k} x^k (1-x)^{n-k} \tag{21}$$

(21) can be readily generalised to the multivariate case:

$$\begin{aligned} x_1^{j_1}\ldots x_m^{j_m} = & \sum_{k_1=j_1}^{n_1} \frac{\binom{k_1}{j_1}}{\binom{n_1}{j_1}} \binom{n_1}{k_1} x_1^{k_1}(1-x_1)^{n_1-k_1} \times \\ & \ldots \times \sum_{k_m=j_m}^{n_m} \frac{\binom{k_m}{j_m}}{\binom{n_m}{j_m}} \binom{n_m}{k_m} x_m^{k_m}(1-x_m)^{n_m-k_m} = \\ & \sum_{\substack{j_l \le k_l \le n_l \\ l\in\{1,\ldots,m\}}} \prod_{h=1}^{m} \frac{\binom{k_h}{j_h}}{\binom{n_h}{j_h}} \binom{n_h}{k_h} x_h^{k_h}(1-x_h)^{n_h-k_h} \end{aligned} \tag{22}$$

We associate a vector $\vec{b}$ with the coefficients of a polynomial $B_f^{(n_1,\ldots,n_m)}(x_1,\ldots,x_m)$ as follows:

$$\vec{b}_k = \beta_{f,k_1,\ldots,k_m}^{(n_1,\ldots,n_m)}\, \forall k \in \left\{0,\ldots, \prod_{1\le i\le m}(n_i+1)-1\right\} \tag{23}$$

where $(k_1,\ldots,k_m)$ is the unique tuple such that $k = k_1 + k_2(n_1+1) + \ldots + k_m(n_1+1)(n_2+1)\ldots(n_{m-1}+1)$ with $0 \le k_l \le n_l\, \forall l \in \{1,\ldots,m\}$. A similar construct is used to associate the j^{th} entry of a vector $\vec{p}$ with the coefficient

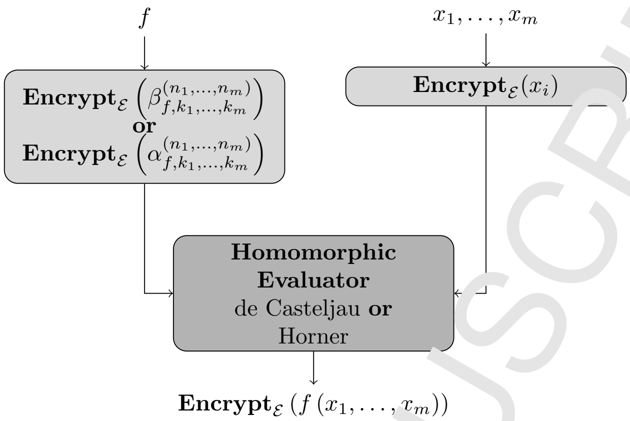

Figure 2: Proposed scheme to derive an encrypted description of multivariate continuous functions and homomorphically evaluate them

$\alpha^{(n_1,\ldots,n_m)}_{j_1,\ldots,j_m}$ of the polynomial $P(x_1, \ldots, x_m)$. Based on (22), a matrix $C \in \mathbb{R}^{\left(\prod_{1\leq i\leq m}(n_i+1)\right)\times\left(\prod_{1\leq i\leq m}(n_i+1)\right)}$ can be built to change a vector in the power form to the Bernstein form:

$$\vec{b} = C\vec{p} \tag{24}$$

where

$$C_{k,j} = \begin{cases} \prod_{h=1}^{m} \frac{\binom{k_h}{j_h}}{\binom{n_h}{j_h}}, & \text{if } k_h \geq j_h \text{ for } 1 \leq h \leq m \\ 0, & \text{otherwise.} \end{cases} \tag{25}$$

and $k = k_1+k_2(n_1+1)+\ldots+k_m(n_1+1)(n_2+1)\ldots(n_{m-1}+1)$ and $j = j_1+j_2(n_1+1)+\ldots+j_m(n_1+1)(n_2+1)\ldots(n_{m-1}+1)$ with $0 \leq k_l, j_l \leq n_l \, \forall l \in \{1, \ldots, m\}$.

Conversely, if $\vec{b}$ is provided in Bernstein form, one can obtain the power form $\vec{p}$ by solving the linear system in (24).

3.3. Confidential Computing Model

A diagram of the proposed scheme for the automatic derivation and evaluation of homomorphic circuits can be found in Figure 2. During an offline phase, the function one wants to approximate homomorphically is evaluated with a black-box approach, producing the values of $\beta^{(n_1,\ldots,n_m)}_{f,k_1,\ldots,k_m}$. The $\beta^{(n_1,\ldots,n_m)}_{f,k_1,\ldots,k_m}$ values underpin an approximate representation of f through a Bernstein polynomial. These values are encrypted and sent to a server. When the system is online, a client can encrypt a tuple $x_1, \ldots, x_m$ of data. The resulting cryptograms are afterwards sent to the server who applies the proposed multivariate de Casteljau algorithm homomorphically to produce an encryption of $f(x_1, \ldots, x_m)$. Alternatively, the $\beta^{(n_1,\ldots,n_m)}_{f,k_1,\ldots,k_m}$ values associated with the Bernstein polynomial can be converted to an equivalent power form with coefficients $\alpha^{(n_1,\ldots,n_m)}_{f,k_1,\ldots,k_m}$. These latter values are encrypted and sent to the server instead of the $\beta^{(n_1,\ldots,n_m)}_{k_1,\ldots,k_m}$. When the server is provided with encryptions of $x_1, \ldots, x_m$ it can similarly compute an encryption of $f(x_1, \ldots, x_m)$, but using the proposed multivariate Horner scheme. In the model depicted in Figure 2, the homomorphic evaluator learns nothing about the function that is being evaluated, which could be any continuous, possibly non-analytic, function, except for the degree of the polynomial that was used to approximate it. In practice, a service provider might wish to make several FHE parameters available, each supporting a certain approximating polynomial degree, and charge its users according to the quality of the approximation provided.

3.4. Performance Characterisation

The two considered number representations, namely the stochastic representation for the de Casteljau algorithm, and the fixed-point approach for the Horner scheme, are supported on different assumptions about the underlying homomorphic encryption scheme. A stochastic number representation can be applied to schemes where batching is used at the bit level. In contrast, a fixed-point representation assumes that one can rescale the encrypted messages. While batching has been successfully applied to a large amount of cryptosystems [15, 19, 20, 21], rescaling consists of a modification to modulus-switching, which is applicable to a more restricted set of cryptosystems [15, 21]. Thus, stochastic number representations are a more general technique than fixed-point arithmetic. Nevertheless, the fixed-point approach allows for the application of the Horner scheme, which is computationally more efficient. Therefore, we use both representations in this paper, since each can be adopted depending on the applications and requirements.

While for the multivariate Horner scheme one needs to perform

$$n_1 + (n_1 + 1) \times (n_2 + (n_2 + 1) \times (\ldots \times n_m)) = O\left(\prod_{1 \le i \le m} n_i\right) \tag{26}$$

homomorphic multiplications; with the multivariate de Casteljau algorithm,

$$\begin{aligned} &\frac{3}{2}n_1(n_1+1) + (n_1+1) \times \\ &\quad \left(\frac{3}{2}n_2(n_2+1) + (n_2+1) \times \left(\ldots \times \frac{3}{2}n_m(n_m+1)\right)\right) = \\ &\qquad O\left(\left(\prod_{1 \le i \le m} n_i\right) \times n_m\right) \end{aligned} \tag{27}$$

homomorphic multiplications are required.

It should be noted though that, if enough parallelism is available, the execution of Algorithm 1 is linear with d, since each iteration of the `for` loop in line 5

Scheme	n_1	n_2	Seq. Time Complexity	Par. Time Complexity
Fixed-Point	5		5	-
Fixed-Point	10		10	-
Fixed-Point	15		15	-
Fixed-Point	2	2	8	4
Fixed-Point	3	3	15	6
Fixed-Point	4	4	24	8
Stochastic	5		45	5
Stochastic	10		165	10
Stochastic	15		360	15
Stochastic	2	2	36	4
Stochastic	3	3	90	6
Stochastic	4	4	180	8

Table 1: Time complexity of the proposed scheme in terms of homomorphic multiplications for univariate and bivariate polynomial approximations of degree n_1 in x_1 and n_2 in x_2 using both a fixed-point approach with Horner's scheme and a stochastic number representation with de Casteljau's algorithm. Since no parallelism can be exploited for the univariate Horner scheme, values were omitted in that case

is independent of one another, and can be computed in parallel. Hence the multivariate variant of this algorithm has a time of complexity of $O\left(\prod_{1\leq i\leq m} n_i\right)$. If, additionally, the recursive calls in (13) and (18) are performed in parallel, the complexity of both techniques reduces to $O\left(\sum_{1\leq i\leq m} n_i\right)$.

Table 1 presents concrete complexity analyses of the proposed methods for univariate ($f(x_1)$) and bivariate ($f(x_1, x_2)$) polynomial approximations of degree n_1 in x_1 and n_2 in x_2. Although the sequential time complexity of de Casteljau's algorithm with a stochastic number representation grows at a faster rate than the fixed-point approach with Horner's scheme, the former representation is more widely applicable, since it only depends on batching, and parallelism may help bridge the performance gap between the two number representations.

4. Implementation Details and Experimental Results

The proposed methods were described using C++ and compiled with GNU's C compiler[1]. The implementation was based on BGV, which supports both batching and modulus-switching. The most relevant parameters for this cryptosystem are: *i*) the underlying cyclotomic polynomial ϕ_m, which determines the amount of available batching slots; and *ii*) the size in bits of the ciphertext modulus $\log_2 q$, which defines the number of homomorphic multiplications one can compute. A combination of the two defines the level of security the scheme

[1]The source-code will be made publicly available when this manuscript is published.

provides [27]. Herein, the parameters were chosen to ensure at least 80 bits of security. It should be noted that one could also support the implementation on other cryptosystems, such as the ones in [20, 19].

The homomorphic operations based on stochastic representations exploit HElib [16]. Besides enabling an automatic selection of parameters, HElib provides interfaces for the basic FHE operations, such as encryption, decryption, homomorphic additions, multiplications and rotations. These operation were combined to provide homomorphic stochastic arithmetic. The homomorphic operations based on the fixed-point approach were implemented with NFLlib [17]. NFLlib provides solely arithmetic methods for power-of-two cyclotomic rings. With NFLlib, one selects q as a product of 62-bit primes. Based on the polynomial arithmetic offered by NFLlib, the FHE operations such as encryption, decryption, homomorphic additions and multiplications were implemented. In addition, the rescaling operation and the fixed-point arithmetic were developed. Finally, multithreading was exploited with the C++ standard library using the strategy described at the end of Section 3.4.

The experimental results herein presented focus on two applications where the usage of non-analytic functions is required, as an example of the wide range of applications that can be targeted with the proposed method. All experiments were executed on an octo-core Intel i7-5960X processor, with the Haswell microarchitecture, with 32GB of RAM, running at 3.0 GHz, operated by Fedora 21, and featuring hyperthreading (i.e. 16 threads are supported simultaneously in total at the hardware level). We also provide an experimental evaluation of the performance of encryption and decryption in embedded devices, since, often times, data processed in the cloud is produced or consumed by devices with limited computational resources. In particular, the encryption and decryption operations presented in Section 4.2 were also executed on a quad-core Cortex-A53 processor, with the Armv8-A architecture, with 8GB of RAM, running at 950 MHz, and operated by OpenEmbedded [28]. Since NFLlib natively only supports x86 platforms, it had to be modified to run on the Arm processor.

Algorithm 3 Sparsemax function for mapping scores to probabilities [29]

Require: $\boldsymbol{z} \in \mathbb{R}^K$

1: Sort $(z_1, \ldots, z_K)$ as $(z^{(1)}, \ldots, z^{(K)})$ s.t. $z^{(1)} \geq \ldots \geq z^{(K)}$

2: $k(\boldsymbol{z}) := \max \left\{ k \in \{1, \ldots, K\} | 1 + kz^{(k)} > \sum_{j \leq k} z^{(j)} \right\}$

3: $\tau(\boldsymbol{z}) := \frac{(\sum_{j \leq k(\boldsymbol{z})} z^{(j)}) - 1}{k(\boldsymbol{z})}$

4: **return** $\boldsymbol{p}$ s.t. $p_i := \max(0, z_i - \tau(\boldsymbol{z}))$

4.1. Effect of Parameters and Parallelism on Non-Analytic Machine-Learning Functions

We have evaluated how the choice of parameters and the proposed parallelisation affect the performance of the homomorphic evaluation of non-analytic functions used in machine learning. We have also assessed the accuracy of the

Function	Scheme	# slots	n_1	n_2	m	$\log_2 q$	MAE	Sequential Execution Time [s]	Parallel Execution Time [s]	Speedup
$\text{sparsemax}_1(x_1, 0)$	Fixed-point		5		2^{15}	744	0.08[illegible]3	0.489	-	-
$\text{sparsemax}_1(x_1, 0)$	Fixed-point		10		2^{15}	744	0.0495	0.689	-	-
$\text{sparsemax}_1(x_1, 0)$	Fixed-point		15		2^{16}	1550	0.033[illegible]	9.00	-	-
$\text{sparsemax}_1(x_1, x_2, 0)$	Fixed-point		2	2	2^{15}	744	0.181	0.902	0.543	1.7
$\text{sparsemax}_1(x_1, x_2, 0)$	Fixed-point		3	3	2^{15}	744	0.13[illegible]	1.57	0.687	2.3
$\text{sparsemax}_1(x_1, x_2, 0)$	Fixed-point		4	4	2^{16}	1550	0.[illegible]20	20.7	6.87	3.0
$\text{sparsemax}_1(x_1, 0)$	Stochastic	630	5		8191	327	0.[illegible]04	0.409	0.272	1.5
$\text{sparsemax}_1(x_1, 0)$	Stochastic	1024	10		21845	14[illegible]0	0.063	16.2	6.40	2.5
$\text{sparsemax}_1(x_1, 0)$	Stochastic	2160	15		55831	2592	0.036	83.0	19.5	4.3
$\text{sparsemax}_1(x_1, x_2, 0)$	Stochastic	630	2	2	8191	327	0.151	0.301	0.254	1.1
$\text{sparsemax}_1(x_1, x_2, 0)$	Stochastic	1024	3	3	21845	985	0.129	9.46	3.58	2.6
$\text{sparsemax}_1(x_1, x_2, 0)$	Stochastic	2160	4	4	55831	2592	0.112	39.6	9.78	4.0

Table 2: The functions $\text{sparsemax}_1(x_1, 0)$ and $\text{sparsemax}_1(x_1, x_2, 0)$ were approximated and homomorphically evaluated as described in Section 3.3, using both a fixed-point approach with Horner's scheme and a stochastic number representation with de Casteljau's algorithm

	Seq. Fixed-Point	Par. Fixed-Point	Seq. Stochastic	Par. Stochastic
Sample Pearson correlation coefficient	0.[illegible]	0.88	0.98	0.95

Table 3: The accuracy of the predictions in Table 1 was evaluated by computing their correlation with the values in Table 2. Values close to 1 indicate a linear dependence between the two variables

performance predictions in Table 1. The sparsemax function [29], described in Algorithm 3, is used to map vectors in $\mathbb{R}^K$ to the $(K-1)$-dimensional simplex. Such maps are used in the final layer of neural networks to convert a vector of scores to a probability distribution. sparsemax was shown to outperform traditionally used functions in multi-label classification and in attention networks for natural language inference [29]. In Table 2, one can find experimental results for the homomorphic evaluation of the functions $\text{sparsemax}_1(x_1, 0)$ (the 1^{st} element of $\boldsymbol{p}$ in Algorithm 3) using approximating polynomials with $n_1 \in \{5, 10, 15\}$ and of the function $\text{sparsemax}_1(x_1, x_2, 0)$ with $(n_1, n_2) \in \{(2, 2), (3, 3), (4, 4)\}$. The functions were approximated in the intervals $[-3, 3]$ and $[-3, 3]^2$, respectively, by linearly mapping those intervals into $[0, 1]$ and $[0, 1]^2$ and using the strategy in Section 2.3.

The experimental results for the univariate case suggest that the degree of the approximating polynomial is inversely proportional to the Mean Absolute Error (MAE). As more dimensions are considered, the Bernstein polynomials take longer to converge to the function, as the degree of the approximating polynomial increases. Since $\text{sparsemax}_1(x_1, 0)$ is univariate, no parallelism was exploited for the fixed-point approach when approximating this function homomorphically. When approximating multivariate functions, the speedup obtained by parallelising the fixed-point approach follows closely what would be expected from the complexity analysis in Section 3.3 $\left(\frac{n_1+(n_1+1)\times n_2}{n_1+n_2}\right)$. In contrast, since the i7-5960X processor supports at most 16 hardware threads, and the parallelisation of Algorithm 1 would optimally require 20 and 30 threads for the evaluation of univariate polynomials of degree 10 and 15, respectively, the speedup is limited for the stochastic representation. Moreover, one is herein able to achieve similar levels of precision for both representations, but the stochastic representation-based scheme takes on average 3.2 times longer to evaluate the same polynomial than the fixed-point approach, when parallelism is considered.

The performance analysis of Section 3.4 was based on the amount of homomorphic multiplications required, without taking into account the characteristics of the underlying FHE system. A more precise analysis should take into account, for instance, the dependency between the degree of the approximating polynomials and the parameters of the FHE scheme, along with their influence on the performance of homomorphic multiplications. Notwithstanding, one can see from Table 3 that the predicted performance figures in Table 1 correlate well with the experimental values of Table 2. This suggests that after a specific number representation and type of implementation are fixed, the analysis of Section 3.4 accurately predicts how performance depends on the n_1 and n_2 parameters.

4.2. Image Processing with Non-Analytic Piece-Wise Multilinear Functions

Two non-analytic piece-wise multilinear functions were developed for image processing, namely grey stretching and image blending, and their encrypted approximations were automatically derived using the proposed methods. Images with 256×256 pixels from [30] were used to get the results presented in this

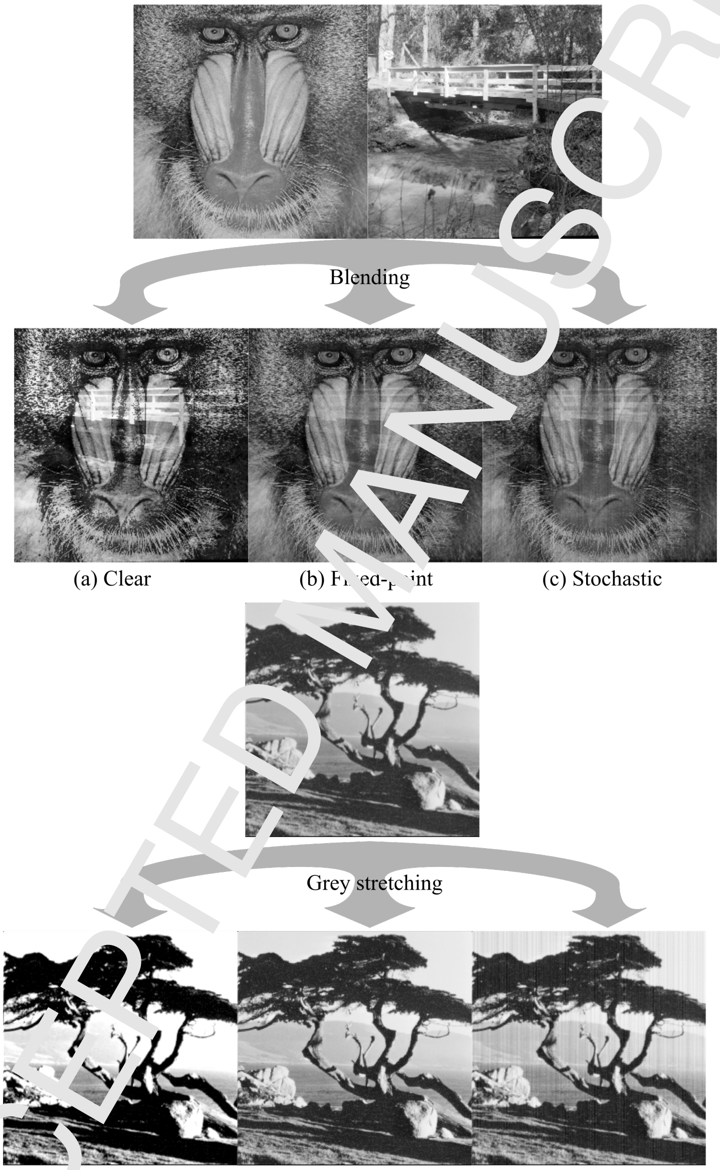

Figure 3: Image processing

System	Encryption [s] Intel / Arm	Filter [s] Intel	Decryption [s] Intel / Arm
Grey Stretching Fixed-point	52.5 / 685	341	6.9 / 134
Blending Fixed-point	52.7 / 684	885	[illegible].3 / 88
Grey Stretching Stochastic	34.5 / 914	1340	61.7 / 1172
Blending Stochastic	47.7 / 1273	2103	8[illegible].4 / 1468
Grey Stretching Floating-point [31]	324 / 7935	95.0	92.7 / 2630

Table 4: Average execution time for homomorphic image processing operations on an i7-5960X (Intel) and on a Cortex-A53 (Arm). When exploiting the fixed-point approach, NFLlib was exploited with a cyclotomic polynomial with $n = 2^{14}$, and $\log_2 q \approx 372$. For the stochastic number representation, HElib was used with $n = 4369$ and $m = 5461$ (accounting for 256 and 378 batching slots), and $\log_2 q \approx 132$ and $\log_2 q \approx 324$ for the grey stretching and blending algorithms, respectively. The last implementation corresponds to an adaption of [31] to the proposed system. [31] uses the Paillier cryptosystem with a 2048-bit modulus

paper. The pixels were mapped to the $[0, 1]$ interval, and functions (28) and (29) were applied, corresponding to grey stretching and image blending, respectively.

$$g(x_1) = \begin{cases} 0, & \text{if } x_1 \leq 0.25 \\ 2x_1 - 0.5, & \text{if } 0.25 < x_1 \leq 0.75 \\ 1, & \text{otherwise.} \end{cases} \tag{28}$$

$$b(x_1, x_2) = \begin{cases} x_1 x_2, & \text{if } x_1 \leq 0.5 \\ 1 - 2(1 - x_1)(1 - x_2), & \text{otherwise.} \end{cases} \tag{29}$$

g was approximated by a Bernstein polynomial with $n_1 = 3$ and b with $n_1 = 2$ and $n_2 = 2$. An example of image processing can be found in Figure 3, where processing was applied in the clear (i.e. without encrypting data), using the homomorphic Horner scheme with the fixed-point approach, and the homomorphic de Casteljau algorithm with the stochastic number representation. While the employing of Bernstein approximations softens the sharp changes of tone in (28) and (29), one can see that the resulting images approximate the expected result in a satisfactory manner.

The average execution times of encrypting an image, processing it and decrypting the result can be found in Table 4. We have included the timings of encrypting and decrypting images on a quad-core Cortex-A53 to have experimental results representative of scenarios where data may be encrypted or decrypted on an embedded device and processed in the cloud. Since the considered homomorphic image processing algorithms show a great level of parallelism, instead of applying parallelism at the arithmetic level, 16 threads were created on the i7-5960X processor and 4 threads on the Cortex-A53. Each thread sequentially processed different parts of the image.

Due to the limited resources of the embedded device, encryption and decryption are, on average, 19.8 and 17.9 times slower to execute on the Arm than on the Intel processor, respectively. Nevertheless, acceptable performance, in the order of hundreds of seconds, is still achieved on the Arm processor. Furthermore, while not considered in this work, techniques have been proposed in related art to reduce the encryption and decryption timings of FHE systems on embedded systems [32, 33] but at the expense of costlier homomorphic processing.

While NFLlib only deals with power-of-two cyclotomics for efficiency reasons, HElib allows for a more fined-grain tuning of the parameters. This is reflected in Table 4. For the two considered processors, encryption takes the same time for both grey stretching and blending when using a fixed-point approach, because both use the same parameter set. With a stochastic number representation, HElib allows for the use of smaller parameters for grey stretching than for blending, leading to a more efficient encryption for the former filter. Nevertheless, as homomorphic operations contribute to the reduction of the bitsize of the ciphertext modulus, decrypting the result on both processors after blending images takes a shorter period than after grey stretching for the fixed-point approach since the former application is more complex. Furthermore, the fixed-point approach takes from 2.4 up to 3.9 times less time to process images, due to the exploitation of power-of-two cyclotomics and the Horner scheme. Finally, one can see an example of the difference in accuracy in Figure 3. For the considered parameters, a stochastic number representation leads to grainier images than a fixed-point one.

5. Related Art

A comparison between the model proposed in Section 3 for cloud computing and those of related art, regarding performance, development effort, scope and privacy, is depicted in Figure 4. Traditional cloud computing models [1] process data in the clear on shared machines. As attacks such as Meltdown and Spectre [4, 5] have shown, currently employed measures to isolate processes might not be enough to ensure data confidentiality.

Homomorphic encryption solves this problem by allowing data to be processed while encrypted: even if an attacker is able to break process isolation, only random looking data will be leaked. Four approaches to cloud computing based on homomorphic encryption have been identified in Figure 4: *i*) Partial

Computing Model	Performance	Development Effort	Scope	Privacy
Traditional [1]	Directly exploits CPU architecture	Traditional programming techniques	Supports any application	Vulnerable to attacks like Meltdown and Spectre
PHE libraries [31]	Overhead associated with PHE	Intricate development. Requires strong familiarity with PHE	Limited support of applications	Hides data
FHE w/ application specific circuits (e.g. [10, 11, 12])	Overhead associated with FHE	Intricate development. Requires strong familiarity with FHE	Supports most applications	Hides data
Proposed model	Limited set of FHE operations	Traditional programming techniques	Non-analytic continuous functions	Hides data and algorithm
FHE w/ encrypted computer architecture [9]	Impractical	Halting problem may cause development issues	Supports most applications	Hides data and algorithm

Best — Worst

Figure 4: Comparative analysis including the proposed model and the remaining state-of-the-art models

Homomorphic Encryption (PHE) libraries; *ii*) FHE with application specific circuits; *iii*) the proposed model; and *iv*) FHE with a complete encrypted computer architecture. PHE refers to cryptosystems where one can either add or multiply encrypted data, but not do both. In particular, with Paillier, one is only able to homomorphically add encrypted messages, and multiply them by constants. Prior work [31] has provided a toolkit for homomorphic image processing supported on Paillier.

Due to the limitations of [31], one cannot use it to implement the proposed scheme as described in Section 3.3. A simplified version of the proposed framework was considered, in order to enable a comparison between the implementation in Section 4.2 and one based on [31]: only the Horner scheme for univariate polynomials is supported; during the offline phase the $\alpha^{(n_1)}_{f,k_1}$ in Figure 2 are sent to the server in the clear; and a client needs to encrypt $x_1^1, \ldots, x_1^{n_1}$ when requesting the evaluation of $f(x_1)$. The server homomorphically approximates f by computing:

$$\begin{aligned} \mathbf{Encrypt}_{\mathcal{E}}\left(f(x_1)\right) \approx \alpha^{(n_1)}_{f,0} +_P \\ \alpha^{(n_1)}_{f,1} \times_P \mathbf{Encrypt}_{\mathcal{E}}\left(x_1^1\right) +_P \ldots +_P \alpha^{(n_1)}_{f,n_1} \times_P \mathbf{Encrypt}_{\mathcal{E}}\left(x_1^{n_1}\right) \end{aligned} \quad (30)$$

The grey stretching kernel was homomorphically applied for $n_1 = 3$ to the images previously considered in Section 4.2, using the scheme adapted from [31], and the experimental results can be found in Table 4. Since [31] does not support homomorphic multiplication, a large amount of computation (namely the computation of the powers of the input x) has to be transferred to the client. This is reflected in Table 4 where the encryption of images took more than 3 times longer to execute in the i7-5960X than their homomorphic processing. This problem is aggravated for systems with limited computational resources, such as the considered Cortex-A53, with the encryption taking over than 80 times more to execute than the homomorphic processing in the i7-5960X. One can thus conclude that using such a scheme would only be beneficial when the same image needs to be processed multiple times. The modified system has also the downside of, unlike the system proposed herein, not providing the ability to hide the function that is being evaluated, since the $\alpha^{(n_1)}_{f,k_1}$ are sent to the server in the clear. Finally, extending the adapted system to multivariate functions, such as blending, seems impractical.

As suggested in Figure 4, FHE extends the applicability of homomorphic encryption to a wider scope of applications when compared to systems based on PHE schemes, such as Paillier. Works such as [34, 35] have provided high-level language constructs that aid with the design of FHE circuits. In [34], a clean interface to HElib [16] is proposed, supported on Python, along with a rudimentary Integrated Development Environment (IDE). In [35], a Domain Specific Language (DSL) is proposed for FHE and Secure Multiparty Computation (SMC) with proved correctness and confidentiality. While both constructs [34, 35] provide high-level operators to process messages homomorphically, they do so without hiding the details about the plaintext space, and without any methodical way on how to design FHE circuitry. As a consequence, conversions of processing algorithms to the FHE domain (e.g. [10, 11, 12]) have traditionally required deep knowledge of the FHE cryptosystem and great development effort. For instance, in [12], numbers are encrypted bitwise and their homomorphic addition amounts to emulating a ripple-carry adder with the homomorphic operations provided by the FHE cryptosystem. The system herein proposed differs from these approaches by exploiting a layer that translates numbers in $\mathbb{R}$ into the cryptosystems' plaintext spaces seamlessly; and that automates the conversion of algorithms to the FHE domain.

Moreover, neither traditional cloud computing methods nor a straightforward conversion of them to the FHE domain hide the processing algorithm. In contrast, the proposed computing model converts algorithms to the FHE homogeneously, hiding the original algorithm. Hence, along with [9], it achieves best privacy of the models considered in Figure 4. However, due to the generality and privacy of the proposed approach, it cannot efficiently target certain applications such as private database queries or sorting [10, 11]. While one can develop dedicated circuits to evaluate these applications [10, 11] and use code obfuscation [36] to improve the circuit privacy, the applicability of code obfuscation to FHE might be limited. FHE often offers a small set of homomorphic operations, each with a very different computational complexity, which reduces the difficulty of an adversary to distinguish them. Efficiently providing algorithm privacy in these cases remains an open research question.

Finally, while [9] has proposed to emulate an entire computer architecture in the cloud with homomorphic operations, these operations are very intensive, and thus the scheme is impractical. Also, since the PC is itself encrypted, there is no practical way to know when the execution has finished; which might make development for such platforms more difficult. We believe that the system proposed herein provides a nice compromise between ease of use and performance, while, like [9], providing the ability to keep the approximated function hidden (namely by encrypting the $\beta^{(n_1,\ldots,n_m)}_{f,k_1,\ldots,k_m}$ or $\alpha^{(n_1,\ldots,n_m)}_{f,k_1,\ldots,k_m}$). Moreover, the amount of homomorphic operations to execute a circuit is known beforehand, and depends directly on the level of accuracy one wants from the approximation of the targeted function.

6. Conclusion

While attacks such as Meltdown and Spectre may have damaged the confidence on cloud computing security by providing the means to break process isolation mechanisms, cryptographic methods are available that maintain confidentiality even when processes are attacked. More concretely, FHE provides the means to process encrypted data, rendering data leaks useless. While there has also been research on protecting both the confidentiality of the processing algorithm and data, this has lead to impractical systems. Herein, we focus on a wide range of functions whose approximations can be efficiently evaluated with homomorphic operations. Since the approximations are all evaluated in the same manner, a homomorphic evaluator has no way to distinguish them. Moreover, the methods herein proposed for the derivation of homomorphic circuitry completely decouple the algorithm development from their homomorphic evaluation. As a consequence, one can use traditional tools for function design, as long as the resulting function is continuous. The proposed system may exploit two different number representations. One concludes that while stochastic representations are more widely applicable, a fixed-point approach provides for better performance.

7. Acknowledgments

This work was supported by Portuguese funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 and by the Ph.D. grant with reference SFRH/BD/103791/2014.

8. References

[1] D. Zissis, D. Lekkas, Addressing cloud computing security issues, Future Generation Computer Systems 28 (3) (2012) 583–592. doi:10.1016/j.future.2010.12.006.
URL `http://dx.doi.org/10.1016/j.future.2010.12.006`

[2] S. Alonso-Monsalve, F. García-Carballeira, A. Caldern, A heterogeneous mobile cloud computing model for hybrid clouds, Future Generation Computer Systems 87 (2018) 651 – 666. doi:https://doi.org/10.1016/j.future.2018.04.005.
URL `http://www.sciencedirect.com/science/article/pii/S0167739X17313897`

[3] B. Varghese, R. Buyya, Next generation cloud computing: New trends and research directions, Future Generation Computer Systems 79 (2018) 849 – 861. doi:https://doi.org/10.1016/j.future.2017.09.020.
URL `http://www.sciencedirect.com/science/article/pii/S0167739X17302224`

[4] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, M. Hamburg, Meltdown, arXiv:1801.01207.
URL `http://arxiv.org/abs/1801.01207`

[5] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, Y. Yarom, Spectre Attacks: Exploiting Speculative Execution, arXiv:1801.01203.
URL `http://arxiv.org/abs/1801.01203`

[6] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, R. Strackx, Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution, in: Proceedings of the 27th USENIX Security Symposium, USENIX Association, 2018.

[7] C. Gentry, Fully Homomorphic Encryption Using Ideal Lattices, in: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing, STOC '09, ACM, New York, NY, USA, 2009, pp. 169–178. doi:10.1145/1536414.1536440.
URL http://doi.acm.org/10.1145/1536414.1536440

[8] P. Martins, L. Sousa, A. Mariano, A Survey on Fully Homomorphic Encryption: an Engineering Perspective, ACM Computing Surveys 50 (6) (2017) 1–33. doi:10.1145/3124441.
URL http://dl.acm.org/citation.cfm?doid=3161158.3124441

[9] M. Brenner, J. Wiebelitz, G. von Voigt, M. Smith, Secret program execution in the cloud applying homomorphic encryption, in: 5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011), IEEE, 2011, pp. 114–119. doi:10.1109/DEST.2011.5936608.
URL http://ieeexplore.ieee.org/document/5936608/

[10] A. Chatterjee, I. SenGupta, Sorting of Fully Homomorphic Encrypted Cloud Data: Can Partitioning be effective?, IEEE Transactions on Services Computing (2017) 1doi:10.1109/TSC.2017.2711018.

[11] D. Boneh, C. Gentry, S. Halevi, F. Wang, D. J. Wu, Private Database Queries Using Somewhat Homomorphic Encryption, in: Proceedings of the 11th International Conference on Applied Cryptography and Network Security, ACNS'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 102–118. doi:10.1007/978-3-642-38980-1_7.
URL http://dx.doi.org/10.1007/978-3-642-38980-1_7

[12] V. Mai, I. Khalil, Design and implementation of a secure cloud-based billing model for smart meters as an internet of things using homomorphic cryptography, Future Generation Computer Systems 72 (2017) 327–338. doi:10.1016/j.future.2016.06.003.
URL http://dx.doi.org/10.1016/j.future.2016.06.003

[13] P. Martins, L. Sousa, A Stochastic Number Representation for Fully Homomorphic Cryptography, in: 2017 IEEE International Workshop on Signal Processing Systems (SiPS), IEEE, 2017, pp. 1–6. doi:10.1109/SiPS.2017.8109973.
URL http://ieeexplore.ieee.org/document/8109973/

[14] J. H. Cheon, A. Kim, M. Kim, Y. Song, Homomorphic Encryption for Arithmetic of Approximate Numbers, Cryptology ePrint Archive, Report 2016/421 (2016).

[15] Z. Brakerski, C. Gentry, V. Vaikuntanathan, (Leveled) Fully Homomorphic Encryption Without Bootstrapping, ACM Trans. Comput. Theory 6 (3) (2014) 13:1—-13:36. doi:10.1145/2633600.
URL `http://doi.acm.org/10.1145/2633600`

[16] S. Halevi, V. Shoup, Algorithms in HElib, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 554–571. doi:10.1007/978-3-662-44371-2_31.
URL `https://doi.org/10.1007/978-3-662-44371-2_31`

[17] C. Aguilar-Melchor, J. Barrier, S. Guelton, A. Guinet, M.-O. Killijian, T. Lepoint, NFLlib: NTT-based Fast Lattice Library, in: RSA Conference Cryptographers' Track, San Francisco, United States, 2016.
URL `https://hal.archives-ouvertes.fr/hal-01242273`

[18] B. R. Gaines, Stochastic Computing Systems, Springer US, Boston, MA, 1969, pp. 37–172. doi:10.1007/978-1-4899-5841-9_2.
URL `http://dx.doi.org/10.1007/978-1-4899-5841-9_2`

[19] N. P. Smart, F. Vercauteren, Fully Homomorphic SIMD Operations, Cryptology ePrint Archive, Report 2011/133 (2011).

[20] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, A. Yun, Batch Fully Homomorphic Encryption over the Integers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 315–335. doi:10.1007/978-3-642-38348-9_20.
URL `http://dx.doi.org/10.1007/978-3-642-38348-9_20`

[21] X. Zhang, C. Xu, C. Jin, R. Xie, J. Zhao, Efficient fully homomorphic encryption from RLWE with an extension to a threshold encryption scheme, Future Generation Computer Systems 36 (2014) 180 – 186, special Section: Intelligent Big Data Processing Special Section: Behavior Data Security Issues in Network Information Propagation Special Section: Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications. doi:https://doi.org/10.1016/j.future.2013.10.024.
URL `http://www.sciencedirect.com/science/article/pii/S0167739X13002422`

[22] C. Gentry, S. Halevi, N. P. Smart, Fully Homomorphic Encryption with Polylog Overhead, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 465–482. doi:10.1007/978-3-642-29011-4_28.
URL `http://dx.doi.org/10.1007/978-3-642-29011-4_28`

[23] C. Heitzinger, Simulation and Inverse Modeling of Semiconductor Manufacturing Processes, 2002.
URL `http://www.iue.tuwien.ac.at/phd/heitzinger/`

[24] Z. Šír, B. Juttler, On de Casteljau-type algorithms for rational Bézier curves, Journal of Computational and Applied Mathematics 288 (2015)

244–250. doi:http://doi.org/10.1016/j.cam.2015.01.037.
URL http://www.sciencedirect.com/science/article/pii/S0377042715000497

[25] D. E. Knuth, The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[26] R. T. Farouki, The Bernstein Polynomial Basis: A Centennial Retrospective, Comput. Aided Geom. Des. 29 (6) (2012) 379–419. doi:10.1016/j.cagd.2012.03.001.
URL http://dx.doi.org/10.1016/j.cagd.2012.03.001

[27] M. R. Albrecht, R. Player, S. Scott, On the concrete hardness of Learning with Errors, Cryptology ePrint Archive, Report 2015/046 (2015).

[28] ARM, 64 Bit Juno r2 ARM Development Platform, https://www.arm.com/files/pdf/Juno_r2_datasheet.pdf (11 2015).

[29] A. F. T. Martins, R. F. Astudillo, From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification, Tech. rep. (2016). arXiv:1602.02068.
URL http://arxiv.org/abs/1602.02068

[30] University of Granada, Test Images, http://decsai.ugr.es/cvg/dbimagenes/ (2014).

[31] M. T. I. Ziad, A. Alanwar, M. Alzantot, M. Srivastava, CryptoImg: Privacy preserving processing over encrypted images, in: 2016 IEEE Conference on Communications and Network Security (CNS), IEEE, 2016, pp. 570–575. arXiv:1609.00881, doi:10.1109/CNS.2016.7860550.
URL http://ieeexplore.ieee.org/document/7860550/

[32] Y. Doröz, A. Shahverdi, T. Eisenbarth, , B. Sunar, Toward practical homomorphic evaluation of block ciphers using prince, Cryptology ePrint Archive, Report 2014/233, https://eprint.iacr.org/2014/233 (2014).

[33] Y. Hu, Improving the Efficiency of Homomorphic Encryption Schemes, Ph.D. thesis, Worcester Polytechnic Institute, https://web.wpi.edu/Pubs/ETD/Available/etd-042513-154859/unrestricted/YHu.pdf (5 2013).

[34] C. T. Frame, HEIDE: An IDE for the Homomorphic Encryption Library HElib, Ph.D. thesis, California Polytechnic State University, San Luis Obispo, California (jun 2015). doi:10.15368/theses.2015.64.
URL http://digitalcommons.calpoly.edu/theses/1403

[35] A. Bain, J. Mitchell, R. Sharma, D. Stefan, J. Zimmerman, A Domain-Specific Language for Computing on Encrypted Data (Invited Talk)doi:10.4230/LIPICS.FSTTCS.2011.6.

URL https://www.researchgate.net/publication/22[illegible]336[illegible]06_A_Domain-Specific_Language_for_Computing_on_Encrypted_Data

[36] M. Hataba, A. El-Mahdy, Cloud Protection by Obfuscation: Techniques and Metrics, in: 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2012, pp. 369–372. doi:10.1109/3PGCIC.2012.18.

Paulo Martins received the MSc degree in Electrical and Computer Engineering from the Instituto Superior Técnico (IST), Universidade de Lisboa (UL), Lisbon, Portugal, in 2014. He is a Junior Researcher with the R&D Instituto de Engenharia de Sistemas e Computadores (INESC-ID). He was an intern during 4 months in 2015 at the Samsung Research United Kingdom. He did a collaboration with the Laboratoire d'informatique de Paris 6 (LIP6) during 4 months in 2016. His research interests include cryptography, computer architectures, parallel computing, and computer arithmetic. He is also a PhD student of IST and a student member of both IEEE and HiPEAC.

Leonel Sousa received a Ph.D. degree in Electrical and Computer Engineering from the Instituto Superior Técnico (IST), Universidade de Lisboa (UL), Lisbon, Portugal, in 1996, where he is currently Full Professor. He is also a Senior Researcher with the R&D Instituto de Engenharia de Sistemas e Computadores (INESC-ID). His research interests include VLSI architectures, parallel computing, computer arithmetic, and cryptography. He has contributed to more than 200 papers in journals and international conferences, for which he got several awards. He is Fellow of the IET, Distinguished Scientist of ACM and Senior Member of IEEE.

Highlights

- Proposed computing model hides both data and algorithms from evaluator
- Users’ functions are automatically converted to the proposed model
- Two different number representations are supported