Minimizing the Number of Bootstrappings in Fully Homomorphic Encryption

Marie Paindavoine, Bastien Vialla

Minimizing the Number of Bootstrappings in Fully Homomorphic Encryption.

Marie Paindavoine[1,2] and Bastien Vialla[3] *

[1] Orange Labs, Applied Crypto Group, Caen, France
[2] Université Claude Bernard Lyon 1, LIP (CNRS/ENSL/INRIA/UCBL),
46 Allée d'Italie, 69364 Lyon Cedex 07, France.
marie.paindavoine@ens-lyon.fr
[3] Université Montpellier, LIRMM, CNRS, 161 rue Ada, F-34095 Montpellier, France
bastien.vialla@lirmm.fr

Abstract. There has been great progress regarding efficient implementations of fully homomorphic encryption schemes since the first construction by Gentry. However, evaluating complex circuits is still undermined by the necessary resort to the *bootstrapping* procedure. Minimizing the number of times such procedure is called is a simple yet very efficient way to critically improve performances of homomorphic evaluations. To tackle this problem, a first solution has been proposed in 2013 by Lepoint and Paillier, using boolean satisfiability. But their method cannot handle the versatility of fully homomorphic encryption schemes. In this paper, we go one step forward providing two main contributions. First, we prove that the problem of minimizing bootstrapping is NP-complete with a reduction from a graph problem. Second, we propose a *flexible* technique that permits to determine both such minimal number of bootstrappings and where to place them in the circuit. Our method is mainly based on linear programming. Our result can advantageously be applied to existing constructions. As an example, we show that for the Smart-Tillich AES circuit, published on the Internet in 2012, we find about 70% less bootstrappings than naive methods.

Keywords: Fully Homomorphic Encryption, Bootstrapping, Complexity Analysis, Mixed Integer Linear Programming.

1 Introduction

Homomorphic encryption extends traditional encryption in the sense that it becomes feasible to perform operations on ciphertexts, without the knowledge of the secret decryption key. As such, it enables someone to delegate heavy computations on his sensitive data to an untrusted third party, in a secure way. More precisely, with such a system, one user can encrypt his sensitive data such that the third party can evaluate a function on the encrypted data, without

* This work is partially funded by the HPAC project of the French Agence Nationale de la Recherche (ANR 11 BS02 013), (ANR 12 BS02 001).

learning any information on the underlying plain data. Getting back the encrypted result, the user can use his secret key to decrypt it and obtain the result of the evaluation of the function on his sensitive plain data. For a cloud user, the applications are numerous, and reconcile both a rich user experience and a strong privacy protection.

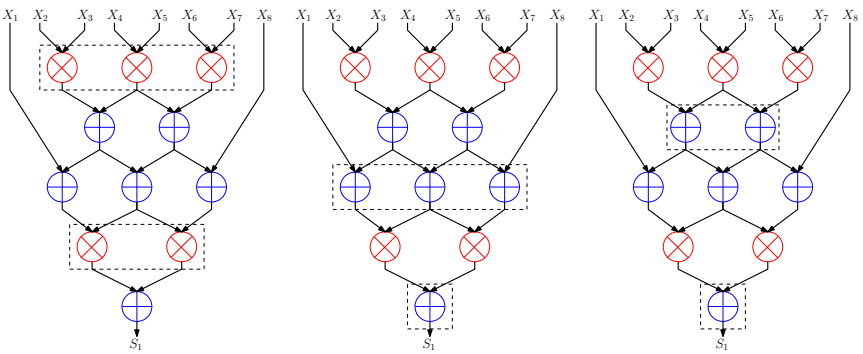
Such a promising idea has first been proposed by Rivest, Adleman and Dertouzos in 1978 [20]. The first homomorphic cryptosystems were able to handle only additions (*e.g.* [14,19]), or only multiplications (*e.g.* [8]), or an arbitrary number of additions but only one multiplication [2]. The first fully homomorphic encryption (FHE) scheme, able to handle an arbitrary number of additions and multiplications on ciphertexts, has been proposed by Gentry in 2009 [10].

In homomorphic encryption schemes, the executed function is typically represented as an *arithmetic circuit*. In practice, any circuit can be described as a set of successive operation gates, each one being either a sum or a product performed over some ring. As we will see, the multiplication is the most important operation to be studied for efficiency optimization of a FHE schemes, and the *multiplicative depth* of a circuit, that is the maximum number of multiplications in a path, is an important parameter for FHE schemes.

In Gentry's construction, based on lattices, each ciphertext is associated with some noise, which grows at each operation (addition or multiplication) done throughout the evaluation of the function (procedure called `HE.Eval` in the sequel). When this noise reaches a certain limit, decryption is not possible anymore. To overcome this limitation, closely related to the number of operations that the `HE.Eval` procedure can handle, Gentry proposed in [10] a technique of noise refreshment called "bootstrapping".

The main idea behind this bootstrapping procedure is to homomorphically run the decryption procedure of the scheme on the ciphertext, using an encrypted version of the secret key. It comes along with a circular security assumption, as we have to feed the decryption circuit with an encryption of the secret key. This permits to get a "refreshed" ciphertext, which encrypts the same plaintext, but with less noise: the decryption is then always feasible. However, the counterpart is that its computational cost is quite heavy and it should be avoided as much as possible [16]. Ducas and Micciancio proposed a bootstrapping procedure in less than a second [7], but their procedure can only be applied to ciphertexts encrypting a single bit. `HElib` [15] procedure, on the other hand, takes roughly 6 minutes. However, the plaintext space is much larger, yielding an amortized cost per bit operation of the same order. In such a context, it is of great importance to determine the exact minimum number of bootstrappings needed to evaluate a given circuit. This way, the time execution for the evaluation of a function will be optimal for a given FHE scheme.

Noise Growth Model. Such a study requires a model to point out how noise grows operation after each operation. Following [17], we associate to each ciphertext c_i a discrete noise level l_i with $l_i = 1, 2, \ldots$. Level 1 corresponds to the noise of encryption procedure output. The last level at which it is necessary to either stop the computation or to bootstrap the ciphertext is denoted l_{max}. The boot-



(a) Bootstrapping after each multiplication.

(b) Bootstrapping before each multiplication.

(c) Optimal solution.

Fig. 1: In dashed rectangle, the bootstrapping positions given by the different heuristics in a FHE scheme with $l_{max} = 2$. (a) The first heuristic uses 5 bootstrappings. (b) The second heuristic uses 4 bootstrappings. (c) Whereas the optimal solution is 3 bootstrappings.

strapping procedure does not reset the noise level of a ciphertext to 1 in general but to a level $1 \leq N < l_{max}$. As we will see later, FHE schemes can be divided into two categories depending on the effect of multiplication on noise level, the exponential ones and the linear ones.

Minimizing Bootstrapping. We introduce the *l_{max}-minimizing bootstrapping problem* as finding (one of) the minimal set of ciphertexts one has to bootstrap in order to correctly evaluate a given circuit. Naively, two heuristics can be used in order to avoid unnecessary bootstrappings.

> **Heuristic 1**: One can bootstrap a ciphertext as soon as its noise level reaches l_{max}. It usually means to bootstrap a ciphertext just after a multiplicative gate.
> **Heuristic 2**: When a ciphertext with noise level l_{max} is produced, one waits as long as possible before bootstrapping it. It usually means to bootstrap a ciphertext just before it is used as input into a multiplicative gate.

But, as shown in figure 1, these two heuristics most of the time fail to produce a solution to the l_{max}-minimizing bootstrapping problem. In this paper, our aim is then to provide a generic method to find such solution.

Previous Works. To the best of our knowledge, the only method to compute a minimal number of bootstrappings has been proposed by Lepoint and Paillier in [17]. It is based on the SAT problem, known to be NP-complete, and on the definition of some noise management rules. They focus on exponential schemes and proposed a method for any l_{max}. In order to handle linear schemes as well, they need to modify the circuit so they can apply their algorithm as a

blackbox. Regarding efficiency, the running time of their solving algorithm grows exponentially with l_{max}, and they do not give timings for $l_{max} \geq 4$.

Outline and Contributions. In this context, our contribution is twofold. We first prove that the l_{max}-minimizing bootstrapping problem is polynomial for $l_{max} = 2$ and NP-complete for $l_{max} \geq 3$. We then propose a new method to determine the minimal number of bootstrappings needed for a given FHE scheme and a given circuit. As well as the previous work, our method also permits to exactly know where to place them in the circuit. We use linear programming to find the best outcome for our problem. The main advantage of our method over the previous one is that it is highly flexible and can be adapted for numerous types of homomorphic encryption schemes and circuits.

The paper is organized as follows. In the next section, we introduce the tools we need all along the paper. Section 3 provides our complexity analysis: the l_{max}-minimizing bootstrapping problem is polynomial for $l_{max} = 2$ and NP-complete for $l_{max} \geq 3$. Finally, section 4 gives our new method for solving the l_{max}-minimizing bootstrapping problem.

2 Background

In this section, we first recall some technical details about graph theory, and in particular arithmetic circuits. We then describe noise growth model during homomorphic evaluation of an arithmetic circuit. Next, we introduce some basic notions of complexity theory. Finally, we present our main tool for solving the l_{max}-minimizing bootstrapping problem which is mixed integer linear programming.

2.1 Graph Theory

As sketched in the introduction, functions handled by homomorphic encryption are arithmetic circuits. They are a particular type of graph. This allows us to make use of complexity results over graph problems for our complexity analysis.

A *graph* G is a couple (V, E) where V is the set of *vertices* and E is the set of *edges*. An edge from a vertex u to a vertex v is noted (u, v). A *directed* graph is a graph where all the edges are oriented, meaning that $\forall u, v \in V,\ (u, v) \neq (v, u)$. For a directed edge (u, v), u is called the *tail* and v the *head*. A $(u_1 - u_{n+1})$-*path* of *length* n is a collection of n edges $((u_1, u_2), (u_2, u_3) \cdots, (u_n, u_{n+1}))$ and a *cycle* is a path where the first vertex equals the last. A directed graph is said *acyclic* if it does not contain any directed cycles. A directed acyclic graph is denoted DAG. The input degree of a vertex x is $|\{(u, x) \in E\}|$, and a vertex whose input degree is equal to 0 is called a *source*. The output degree of a vertex x is $|\{(x, u) \in E\}|$, and a vertex whose output degree is equal to 0 is called a *sink*.

Arithmetic Circuit. An *arithmetic circuit* $\mathcal{C} = (\mathcal{G}, \mathcal{W})$ is a DAG defined over a ring $\mathbb{R}$ and a set of n variables $X = \{X_1, X_2, \cdots, X_n\}$ as follows. The vertices $\mathcal{G}$ of $\mathcal{C}$ are called *gates*. The edges $\mathcal{W}$ of $\mathcal{C}$ are called *wires*. A gate of input degree 0

is an *input gate* and is labelled either by a variable from X or a ring element. Every other gate has an input degree 2, and is labelled either by $\times$ or $+$. We respectively call them *product gates* and *sum gates*. Every gate of output degree 0 is called an *output gate*. In the case of binary circuits defined over $\mathbb{F}_2$, we also have gates of input degree 1. They are labelled `NOT`, and are called *`NOT` gates*.

Let $\mathcal{P}$ be a path in $\mathcal{C}$. We call the *multiplicative length* of $\mathcal{P}$ the number of product gates in $\mathcal{P}$. Let us note that the multiplicative length of $\mathcal{P}$ is defined with respect to the number of product gates of $\mathcal{P}$, whereas its length is defined with respect to the number of edges. Therefore, for a path $\mathcal{P}$ that is only composed of k product gates (and no sum gates), its length is $k-1$ and its multiplicative length is k.

2.2 Noise Growth Model

In existing homomorphic encryption schemes, each ciphertext has some noise attached to it. This noise grows throughout the `HE.Eval` procedure. In this section, we model how the noise grows operations-wise. As pointed out in the introduction, we use a discretized noise model.

Additions in homomorphic encryption are almost free. The noise growth induced by additions is indeed logarithmic with regard to the noise growth induced by multiplications. It can therefore be neglected most of the time. In this case, let c_1, c_2 be two ciphertexts of noise level l_1 and l_2, and let $c_3 = c_1 + c_2$. We have $l_3 = \max(l_1, l_2)$. However, this restriction is not necessary to apply our method for solving the l_{max}-minimizing bootstrapping problem, and the logarithmic noise induced by additions can be taken into account in our model.

The effect of a multiplication on noise levels divides FHE schemes into two categories. Let c_1, c_2 be two ciphertexts of noise level l_1, l_2 and $c_3 = c_1 \cdot c_2$ with noise level l_3.

- The *exponential* schemes [9,24,6,4]: in these schemes, we have $l_3 = l_1 + l_2$. Therefore, the evaluation of a circuit with a multiplicative depth D will require $l_{max} > 2^D$. This becomes quickly unacceptable and in practice l_{max} is set to 2.
- The *linear* schemes [3,11]: in these schemes, we have $l_3 = \max(l_1, l_2) + 1$. However, in those schemes, the user can set l_{max} to be greater than the multiplicative depth of the circuit to be evaluated. This comes at the cost of greater public parameters. When the multiplicative depth of the circuit is not known in advance, or is too important, one still has to resort to bootstrapping.

2.3 Complexity Theory

We recall the basic definitions of the classic complexity classes that we use in section 3.

A *decision problem* is a yes-or-no question on an infinite set of inputs. A problem P is in the NP class if the verifying a feasible solution can be done in

polynomial time. A problem P is *NP-hard* if P is at least as hard as the hardest problem in NP. In particular, a NP-hard problem is not necessary in NP.

To prove that a problem P is NP-hard we use a *reduction* that preserves the NP-hardness defined as follows:

Definition 1 (Reduction). *Let A and B be two decision problems, A NP-hard. Let x be an instance of A. A reduction is a pair of algorithms (f, g) such that:*

- *f is a polynomial algorithm transforming x into an instance $f(x)$ of B,*
- *g is a polynomial algorithm transforming a solution y of B in $f(x)$ into a solution $g(x, y)$ in x of A.*

A problem P is *NP-complete* if P is in NP and P is NP-hard.

2.4 Mixed Integer Linear Programming

To solve the l_{max}-minimizing bootstrapping problem, we use linear programming [21], and especially mixed integer linear programming (MILP). Linear programming is used to minimize a linear function whose variables are subject to linear constraints. An *integer linear programming problem* is expressed in the following form. Let A be a matrix in $\mathcal{M}_{m\times n}(\mathbb{R})$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, $x, l, u \in \mathbb{Z}^n$. The program objective is:

$$\begin{aligned} \text{Minimize} \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{Subject To} \quad & a_{11}x_1 + a_{12}x_2 + \cdots + c_{1n}x_n \geq b_1 \\ & \vdots \\ & a_{n1}x_1 + a_{n2}x_2 + \cdots + c_{nn}x_n \geq b_n \\ & \forall x_i, l_i \leq x_i \leq u_i. \end{aligned}$$

We call $c^T x$ the *objective function*, x the *problem variables*, l the *lower bounds* on x, u the *upper bounds* on x and Ax the *linear constraints*. Constraints should not be defined with strict inequalities. If $x_i \in \{0, 1\}$, they are named *boolean variables*. The goal of this formulation is to find values for x that minimize the objective function without violating any constraints.

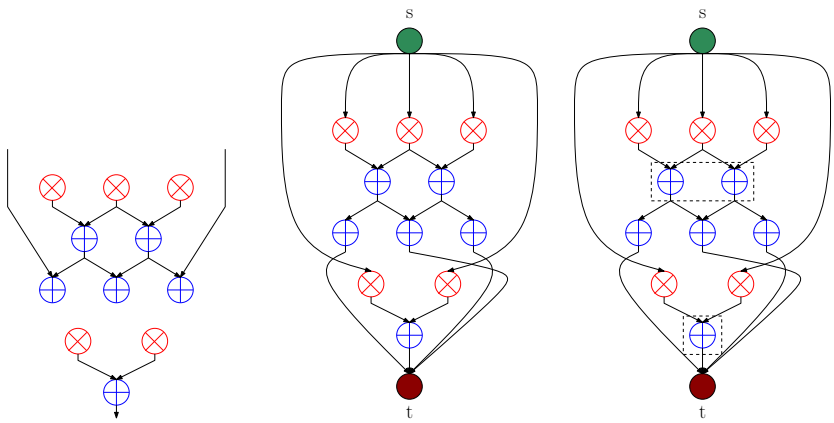
A *mixed integer linear programming problem* is an integer linear programming problem where some of the x_is (and the corresponding u_is and l_is) are allowed to be in $\mathbb{R}$.

Note that non-linear terms are not allowed in the model. Expressing constraints on the multiplication of variables or the maximum of variables is not straightforward, but is still possible with various techniques.

As for any optimization problem, a solution that satisfies all constraints is a *feasible solution*. An *optimal solution* is a feasible solution that achieves the best objective function value.

Theorem 1. *The decisional version of the mixed integer linear programming problem is NP-complete.*

Proof. See [21]. □



(a) Delete every edge entering a product gate.

(b) Add an edge from s to each product gate, and from every sink to t.

(c) Solve the graph connectivity problem on this instance.

Fig. 2: Algorithm for finding the optimal solution for $l_{max} = 2$ applied to the circuit from Figure 1

3 Complexity Analysis of the l_{max}-Minimizing Bootstrapping Problem

In this section, we first formally introduce the l_{max}-minimizing bootstrapping problem, before proving that it is polynomial for $l_{max} = 2$ and NP-complete for $l_{max} \geq 3$.

The l_{max}-minimizing bootstrapping problem is formally defined as a decision problem as follows.

Definition 2 (l_{max}-Minimizing Bootstrapping (l_{max}-MB)). *Let l_{max} be the desired maximum noise level and $\mathcal{C} = (\mathcal{G}, \mathcal{W})$ be an arithmetic circuit. Is there a subset $\mathcal{S} \subseteq \mathcal{G}$ of size ω such that each path $\mathcal{P} \subseteq \mathcal{C}$ of multiplicative length l_{max} has at least one gate in $\mathcal{S}$?*

3.1 A Polynomial Time Algorithm for $l_{max} = 2$

In order to prove that the l_{max}-minimizing bootstrapping problem is polynomial for $l_{max} = 2$, we design an algorithm that solves it using a *graph connectivity* algorithm as a blackbox.

In a DAG $G = (V, E)$, with a source s and a sink t we define a (s, t)-*separator*, that is, a subset $W \subseteq V$ such that each (s, t)-paths has at least one vertex in W. The *graph connectivity* problem consists in finding a minimal (s, t)-separator. This problem can be solved in $O(|V||E| \log(|V|^2/|E|))$ (see [1]).

In what follows, we describe the algorithm solving the 2-minimizing bootstrapping problem using the graph connectivity problem. Let $\mathcal{C}$ be a circuit and

$G = (V, E)$ the underlying DAG. As only one level of product is allowed between each bootstrapping, the goal is to split G into subgraphs where each path has a multiplicative length of 1.

The first step is to delete every arc $(u, v) \in E$ where v is a product gate. The resulting graph is named G'. This step is depicted in figure 2a.

The connected components of G' are also directed acyclic graphs, but the underlying circuit has at most one level of multiplication. The second step is to add an edge from the source s to each product gate and one from each component's sinks to t. With this construction, each (s, t)-path passes through one and only one product gate. Therefore, in order to correctly evaluate the circuit $\mathcal{C}$, we want to bootstrap each ciphertext once per path. In other words, we have to find the smallest subset of vertices $S \subseteq V$, for which each path has a gate in S. S is an (s, t)-separator of G.

In Figure 3, we represent our algorithm which computes the minimal set of bootstrappings. A (toy) running example is depicted in Figure 2.

Algorithm 1: Building the minimum set of bootstrappings for $l_{max} = 2$.

Data: $\mathcal{C}$ a circuit and $G = (V, E)$ the associated directed acyclic graph.
Result: The minimum set S of variables to bootstrap.
begin
  Delete every edge (u, v) where v is a product gate (figure 2a);
  Add two vertices s and t, s will be the source of G and t the sink;
  For each multiplication vertices v, add an edge (s, v) (figure 2b);
  For each edges (u, v) deleted in step 1, add an edge (u, t) (figure 2b);
  Compute the minimal (s, t)-vertex separator S (figure 2c);
  Return S;

Fig. 3: Algorithm to compute the minimal set of bootstrapping for exponential schemes.

Theorem 2. *The asymptotic complexity of Algorithm 1 is*

$$\mathcal{O}(|V||E| \log(|V|^2/|E|)).$$

Proof. The complexity of the first and third steps is $\mathcal{O}(|V|)$ and the complexity of the fourth step is $\mathcal{O}(|E|)$. The second step is executed in constant time. The complexity for computing a minimal (s, t)-separator is $\mathcal{O}(|V||E| \log(|V|^2/|E|))$, therefore, the general complexity of the algorithm is $\mathcal{O}(|V||E| \log(|V|^2/|E|))$. □

Thus, the 2-minimizing bootstrapping problem, which mostly corresponds to exponential schemes can be solved in polynomial time. Moreover, graph connectivity algorithms provide us with the optimal bootstrapping location in the circuit.

3.2 NP-Completeness of the l_{max}-Minimizing Bootstrapping Problem

In this section we prove that the l_{max}-minimizing bootstrapping problem is NP-complete for $l_{max} \geq 3$. We reduce the vertex cover problem known to be NP-complete to the l_{max}-MB problem. We need to introduce an intermediary problem: the k-path vertex cover problem.

Let us first recall the decison version of the vertex cover problem on a DAG [18].

Definition 3 (Vertex Cover in Directed Acyclic Graph (VCD)). *Let $G = (V, E)$ be a directed acyclic graph. Is there a subset $W \subseteq V$ of size ω, such that each edge in E admits a vertex of W as tail or head (or both)?*

Theorem 3. *The VCD problem is NP-complete.*

Proof. See [18]. □

Let us now extend the definition of VCD to a directed version of the k-path vertex cover problem from [5].

Definition 4 (k-Path Vertex Cover in Directed Acyclic Graph (k-PVCD)). *Let $G = (V, E)$ be a directed acyclic graph. Is there a subset $W \subseteq V$ of size ω, such that each path P in G of length k has a vertex in W,* i.e., *$P \cap W \neq \emptyset$?*

Problems	VCD		$k-$PVCD
Instances	x	$\overset{f}{\longmapsto}$	$f(x)$
Solutions $g(x, Y)$ solution of x		$\overset{g}{\longmapsto}$	Y solution of $f(x)$

Fig. 4: Scheme of a reduction from the VCD problem to the k-PVCD problem.

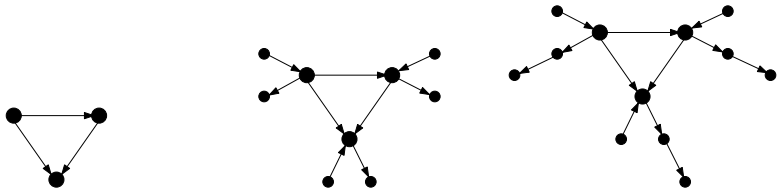
Theorem 4. *The k-PVCD problem is NP-complete for $k \geq 2$.*

Proof. A scheme of the reduction is depicted Figure 4.

Note that for $k = 2$, k-PVCD is the same as VCD which is NP-complete.

For $k > 2$ we show a reduction (f, g) from the VCD problem to the k-PVCD problem.

Let $G = (V, E)$ be an arbitrary directed acyclic graph. We transform G into a k-PVCD instance $f(G) = G'$. Let $G' = (V', E')$ be the graph obtained from G such that for all $x \in V$ we add a directed path of $\left\lfloor \frac{k}{2} \right\rfloor - 1$ new vertices where x is the head; and a path of size $\left\lceil \frac{k}{2} \right\rceil - 1$ new vertices where x is the tail. We call the vertices of G *original vertices*, and the others the *new vertices*. This transformation f has a linear complexity with respect to $|V|$. An example is depicted in Figure 5.



(a) Original graph G. (b) Graph G' for $k = 4$. (c) Graph G' for $k = 5$.

Fig. 5: Example of the G' construction.

We now have to transform back a k-PVCD feasible solution Y in G' into a VCD feasible solution $g(G, y)$ in G.

Let y be a k-path vertex cover in G'. Suppose that y contains a new vertex u that lies in one of the added path, *i.e.*, $\exists u \in Y,\ u \notin V$. Let $v \in V$ be the original vertex closest to u. Note that u only secures one path, hence we can swap u with v in Y. We can apply this procedure until all vertices of y are in V. Let us name g the algorithm just described. We claim that $g(G, Y) \subseteq V$ is a vertex cover in G.

Let us suppose otherwise. There is an edge $(u, v) \in E$ such that $u, v \notin g(G, Y)$. Depending on the orientation of the edge between u and v, consider the path P in G', composed of the path attached to x where u is the head (resp. the tail), of the edge (u, v) (resp. (v, u)), and of the path attached to v where v is the tail (resp. the head). Then P does not contain any vertex from y, and it has $\lfloor \frac{k}{2} \rfloor + \lceil \frac{k}{2} \rceil - 2 + 2 = k$ vertices, which is a contradiction. Hence, $g(G, Y)$ is a vertex cover in G.

The transformation g of a k-PVCD feasible solution in G' into a VCD feasible solution in G has a linear complexity with respect to $|V|$.

Conversely, we prove that a vertex cover X in G yields a k-path vertex cover in G'. Let us suppose otherwise. There is a path P of length k in G' such that $P \cap X = \emptyset$. By construction of G', at least one edge of P is in G, let $e = (u, v) \in P$ be this edge. So, $u, v \notin X$ which is a contradiction because X is a vertex cover. Hence, X is a k-path cover in G'.

Thus, there is reduction (f, g) from VCD to k-PVCD: k-PVCD is NP-hard.

We finally prove that an alleged solution of k-PVCD in a DAG $\tilde{G} = (\tilde{V}, \tilde{E})$ can be verified in polynomial time. Let Δ^- be the maximum output degree of $\tilde{G}$. The number of paths of size k in $\tilde{G}$ is at most $O(|\tilde{V}|\Delta^{-k})$, and the paths of length k can be computed using a truncated breadth first search on every vertex, with a complexity of $O(|\tilde{V}|(|\tilde{V}| + |\tilde{E}|))$. So a solution can be verified in polynomial time. k-PVCD lies in NP.

Hence k-PVCD is NP-hard and NP: it is NP-complete. □

Now we can prove that l_{max}-MB is NP-complete by reducing the k-PVCD problem to the l_{max}-MB problem. A scheme of the reduction is depicted Figure 7.

Theorem 5. *l_{max}-MB is NP-complete for $l_{max} \geq 3$.*

Proof. We show a reduction (f, g) from the k-path vertex cover problem to the l_{max}-minimizing bootstrapping, for $l_{max} \geq 3$ and $k = l_{max} - 1$.

Let $G = (V, E)$ be an arbitrary directed acyclic graph. We transform G into a l_{max}-MB instance $f(G) = \mathcal{C}$. $\mathcal{C}$ is not required evaluate any "interesting" function. For our reduction purpose, we only need that any path P of length k in G is transformed into a path $\mathcal{P}$ in $\mathcal{C}$ with multiplicative length l_{max}.

Let $\Delta^+(G)$ be the maximum input degree of G. In order to transform G into a circuit $\mathcal{C} = (\mathcal{G}, \mathcal{W})$, we distinguish three cases. When a vertex of G has input degree 2, it is directly transformed into a product gate. When a vertex of G has input degree 1, it is transformed into a product gate, the second input of the gate being a field constant. Finally, every vertex $x \in V$ with input degree at least 3 is transformed into a subcircuit only composed of sum gates (each of input degree 2) except for the last one that will be a product gate, see Figure 6. Note that f is a bijection between the vertices of G and the product gates of $\mathcal{C}$, and that f has a linear complexity with regard to $|V|$.

We now have to transform a l_{max}-MB feasible solution y in $\mathcal{C}$ into a k-PVCD feasible solution $g(G, y)$ in G.

Let Y be a l_{max}-MB feasible solution in $\mathcal{C}$. The transformation g consists in moving every bootstrapping that is placed on a sum gate to the next product gate downwards. Every bootstrapping is now on a product gate. We claim that $g(x, Y)$ is a $(l_{max} - 1)$-path cover of G.

Let us suppose otherwise. There is a path $P \subseteq G$ of length $l_{max} - 1$ which is not covered by $g(x, Y)$. Let $\mathcal{P} \subseteq \mathcal{C}$ be the path obtained after the transformation of P. A path of length $l_{max} - 1$ is composed of l_{max} vertices. Each of these vertices is transformed into a subcircuit that contains exactly one multiplication. So the multiplicative length of $\mathcal{P}$ is equal to l_{max}. Therefore, there is a path in $\mathcal{C}$ of multiplicative length l_{max} that is not covered by y, which is a contradiction. Hence, $g(x, Y)$ is a $(l_{max} - 1)$-path cover of G. The transformation g between a l_{max}-MB feasible solution in $\mathcal{C}$ and a k-PVCD feasible solution in G has linear complexity with regard to $|V|$.

Conversely, using a similar reasoning, we can show that a $(l_{max} - 1)$-path vertex cover of in G yields a l_{max}-MB in $\mathcal{C}$.

Hence the l_{max}-MB problem is NP-hard. We have now to prove that the l_{max}-MB problem is NP. That is any alleged solution of l_{max}-MB in an arithmetic circuit $\tilde{\mathcal{C}} = (\tilde{\mathcal{G}}, \tilde{\mathcal{W}})$ can be verified in polynomial time. Let Δ^- be the maximum output degree of $\tilde{\mathcal{C}}$. The paths of multiplicative length l_{max} can be computed using a truncated breadth first search on every vertex, with a complexity of $O(|\tilde{\mathcal{G}}|(|\tilde{\mathcal{G}}| + |\tilde{\mathcal{W}}|))$. So a solution can be verified in polynomial time, so l_{max}-MB lies in NP.

Hence, l_{max}-MB is in NP and is NP-hard: it is NP-complete. □

Thus, the l_{max}-minimizing bootstrapping problem, for $l_{max} \geq 3$, is NP-complete. In the following section we provide a constructive method to solve it.

(a) Vertex of G. (b) Vertex transformation in $\mathcal{C}$.

Fig. 6: Transformation a vertex of input degree greater than 2.

Problems	$k - \texttt{PVCD}$		$l_{max} - \texttt{MB}$
Instances	x	$\overset{f}{\longmapsto}$	$f(x)$
Solutions	$g(x, Y)$ solution of x	$\overset{g}{\longmapsto}$	Y solution of $f(x)$

Fig. 7: Scheme of a reduction from the k-PVCD problem to the l_{max}-MB problem.

4 Minimizing Bootstrappings with Mixed Integer Linear Programming

In this section we present a general and adaptable method based on mixed integer linear programming for solving the l_{max}-minimizing bootstrapping problem. We first introduce the model's variables and then we describe a general MILP model that can take into account many types of FHE operations. Moreover, one can choose the noise level at which the ciphertexts are refreshed after a bootstrapping.

4.1 Defining Variables and Objective Function of the Program

At each gate of the circuit, we attach a boolean variable which will take the value `true` if it is necessary to bootstrap after the node. The goal of our optimization program will be to minimize the sum of those bootstrapping variables.

For each gate $G^{(i)}$ of the circuit we denote by $G_1^{(i)}$ and $G_2^{(i)}$ the noise levels of the gate inputs. For each output wire of a gate, we add a fictive node corresponding to our bootstrapping boolean variable that we denote $B^{(i)}$. If $B^{(i)}$ equals to one, it means that a bootstrapping is necessary after the ith gate of the circuit. In order to keep the notations simple, $B^{(i)}$ will be used either for the boolean variable or for the fictive bootstrapping computation node. We consider that the $B^{(i)}$ node takes as input the noise level of the gate output it is attached to, which we denote $G_{in}^{(i)}$ and outputs a noise level variable $G_{out}^{(i)}$. These variables are depicted in Figure 8.

Each of those variables admits 1 as lower bound and l_{max} as upper bound. Furthermore we require that the noise level of each circuit output is strictly less than l_{max} in order to have a correct decryption or to allow further computations. Minimizing the number of bootstrappings is equivalent to minimizing the number

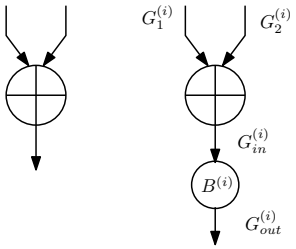


Fig. 8: Variables representing the noise level of a gate in the mixed integer linear programming problem.

of boolean variables set to true. Hence, the objective function to be minimized is:

$$\sum_i B^{(i)}.$$

4.2 Linear Constraints

We translate the relations between the noise levels of each gate into linear constraints. We describe them thoroughly for the main FHE operations: addition and multiplication. The model can easily be modified to include other kinds of FHE operations as long as the noise growth can be translated into linear constraints.

Bootstrapping. We first express the constraints that rule the noise growth after the bootstrapping gate added to each gate of the circuit. We recall that the scheme can handle l_{max} operations before the first bootstrapping and that each bootstrapping resets the noise level to N. If we do not bootstrap at a gate $B^{(i)}$, the noise level of the output of the gate is not affected, and we want $G_{in}^{(i)}$ to be equal to $G_{out}^{(i)}$. We can formulate these into a simple constraint:

$$G_{out}^{(i)} = G_{in}^{(i)} \cdot (1 - B^{(i)}) + N \cdot B^{(i)}. \tag{1}$$

This quadratic constraint can be written as a linear constraint using an auxiliary constant X such as $X \geq l_{max}$. The constraints system becomes:

$$\begin{cases} G_{out}^{(i)} \geq N \cdot B^{(i)} & (2) \\ G_{out}^{(i)} \leq N + (1 - B^{(i)}) \cdot X & (3) \\ G_{out}^{(i)} \geq G_{in}^{(i)} - X \cdot B^{(i)} & (4) \\ G_{out}^{(i)} \leq G_{in}^{(i)} + X \cdot B^{(i)}. & (5) \end{cases}$$

We can see that if the solver decides to bootstrap at gate i, both equations (2) and (3) will force the equality $G_{out}^{(i)} = N$ while equations (4) and (5) remain true. On the other hand, if the solver decides not to bootstrap, equations (4) and (5) will force the equality $G_{out}^{(i)} = G_{in}^{(i)}$ while the other two will remain true.

Addition. Let c_1, c_2 be two ciphertexts with noise levels l_1, l_2 respectively. We denote $c_3 = c_1 + c_2$ with noise level l_3. We want to ensure that $l_3 = \max(l_1, l_2)$. The maximum is not a linear function, so it cannot be directly used in a constraint. We prove later that the following implication is enough for our purposes:

$$A_{in}^{(i)} = \max(A_1^{(i)}, A_2^{(i)}) \Longrightarrow \begin{cases} A_{in}^{(i)} \geq A_1^{(i)} \\ A_{in}^{(i)} \geq A_2^{(i)}. \end{cases}$$

These equations are linear so we can use them as constraints with the following bounds on the variables: $1 \leq A_{in}^{(i)} \leq l_{max}$ and $1 \leq A_j^{(i)} \leq l_{max}$.

Remark 1. If the proportion of sum gates in the circuit is overwhelming, our model can consider the logarithmic noise growth induce by additions. Let $\varepsilon \in [0, 1]$ be the noise added by a sum gate normalized with respect to the noise added by a product gate. The noise level of a sum gate output is $l_3 = \max(l_1, l_2) + \varepsilon$. Working with mixed integer linear programming instead of integer linear programming allows to consider this noise using the following linear constraints:

$$\begin{cases} A_{in}^{(i)} & \geq A_1^{(i)} + \varepsilon \\ A_{in}^{(i)} & \geq A_2^{(i)} + \varepsilon, \end{cases}$$

with the same lower and upper bounds as for the addition case.

Multiplication. Let c_1, c_2 be two ciphertexts with noise levels l_1, l_2 respectively. We denote $c_3 = c_1 \cdot c_2$ with noise level l_3. We want to ensure have $l_3 = \max(l_1, l_2) + 1$. We have the following linear constraints:

$$l_3 = \max(l_1, l_2) + 1 \Longrightarrow \begin{cases} M_{in}^{(i)} & \geq M_1^{(i)} + 1 \\ M_{in}^{(i)} & \geq M_2^{(i)} + 1, \end{cases}$$

with $1 \leq M_{in}^{(i)} \leq l_{max}$ and $1 \leq M_j^{(i)} \leq l_{max} - 1$ as upper and lower bounds for the linear program.

Other Operations. Other gates types can fit in our model as long as the noise growth rules can be expressed as linear constraints. For example, a multiplication by a constant roughly adds half a level [12] and therefore can be considered. In the GSW scheme [13], the authors used NAND gates. Our model can be applied to such a scheme as a NAND gate behaves with regard to noise growth exactly as a multiplicative gate.

Theorem 6. *The above MILP is equivalent to the l_{max}-minimizing bootstrapping problem.*

Proof. The constraints definition straightforwardly implies that every solution to the l_{max}-MB problem is a solution of the MILP.

Let us now show the converse. Let S be a MILP solution that is not a l_{max}-MB solution. There exists a path P in the circuit with multiplicative length l_{max} such

that $P \cap S = \emptyset$. The noise level of a ciphertext along this path respects all the MILP constraints. In particular, it increases by at least 1 at each product gate. Its noise level at the end of the path is thus at least l_{max}. This is in contradiction with the noise variables constraints: each one of them is bounded by l_{max} and the circuit outputs has a noise level strictly less than l_{max}. Then S cannot be a MILP solution. □

4.3 Practical Experimentations

In this section we discuss the practical results of our model on several circuits from [22], and on the AES circuit used in [12]. Circuits' characteristics are described in Table 1. We assume that the circuit's inputs noise level is equal to 1 and we require that the noise level of each circuit output is strictly less than l_{max}.

MILP Solvers. MILP solvers do not only solve the original program but also its dual. The transformation of a primal form of a MILP into its dual in our case is the following:

$$\min \left\{ c^T x \mid Ax \geq b,\ l \leq x \leq u \right\} \mapsto \max \left\{ b^T y \mid Ay \leq c,\ l \leq y \leq u \right\}.$$

A feasible solution of the dual problem gives a lower bound on the optimal solution [21] . The difference between a feasible solution of the linear program and a feasible solution of its dual is called the *gap*, until it reaches zero. It then means that the solution found is optimal. The gap gives a hint on how far the given solution is from the optimum in the worst case. As we will see in experimentations, the gap value is useful because it allows to get an approximate solution quickly.

Benchmarks. For the experimentation we ran both the Gurobi Optimizer 6 [4] and IBM CPLEX 12.6 [5] on an Apple MacBook Pro with 2.3 GHz Intel Core i7 and 16GB of RAM. Each solver implements many different optimization routines, which makes difficult to predict the computation time. We tried both solvers on small circuits and choose the faster one to tackle the problem on bigger circuits. In our case, Gurobi performs better on all circuits. The results are displayed in Table 2. We tested two settings:

1. $(l_{max} = 2, N = 1)$. For this setting, we found the same solutions as in [17].
2. $(l_{max} = 20, N = 9)$ as more realistic parameters, similar to those used in [16], except for the AES from [12] where we chose the same parameters as the authors.

For the simplest circuits, such as `Adder` and `Comparator`, the heuristics find the optimal solution or a close one. For those circuits, computing the optimal solution is done in less than a second.

For bigger circuits, running time is difficult to predict. For l_{max} being small, as well as "close" to the circuit multiplicative depth the optimal solution is found

[4] http://www.gurobi.com/
[5] http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud

in a couple of minutes. Between these settings, the solver can take hours to find the optimal solution. Nonetheless, the solver always finds a good approximation, better than both heuristics, in tens of minutes. But it can take a couple of hours to prove optimality. This is where the gap value is important: one can choose to stop the computation time when the gap reaches some desired threshold. For the `DES` circuit, we stopped the solver after 3.5 hours of computation, when the gap reached 5% of error. In comparison with the more efficient heuristic, this spares 3566 bootstrappings.

Unlike circuits from [22], the AES circuit from [12] exploits all the possibilities offered by a FHE scheme. In particular they use SIMD ([23], where ciphertexts are vectors of encrypted plaintexts, and operations are performed component-wise. These vectors are regularly permuted. This does not impact the noise level of ciphertexts. The plaintext space is also bigger than for the binary circuits from [22] which explains that much fewer bootstrappings are needed to correctly evaluate it. This circuit is described is Table 1.

Circuits	Mult. gates	Add. gates	`NOT` gates	Mult. depth
Adder 32 bits	127	61	187	64
Adder 64 bits	265	115	379	128
Comparator 32 bits	150	0	150	23
Multiplier 32×32	5926	1069	5379	128
AES (expanded key)	5440	20325	1927	41
DES (expanded key)	18175	1351	10875	262
MD5	29084	14150	34627	2973
SHA256	90825	42029	103258	3977
Circuit	**Mult. gates**	**Add. gates**	**Mult. cst gates**	**Mult. depth**
AES [12]	30	220	230	40

Table 1: Circuits' characteristics.

5 Conclusion

While homomorphic encryption implementations are now available for anyone who wants to evaluate circuits on encrypted data, performances in the computation are largely undermined either by time taken by the bootstrapping step or by memory requirement when increasing l_{max}. In this paper we proposed an efficient and flexible technique to determine the minimal number of bootstrapping when evaluating circuits in homomorphic encryption. In [5], the authors give an upper bound on the size of the solution of the k-path vertex cover with respect to the vertices degree of the graph. It would be interesting to see if it is possible to

Circuits	l_{max}	N	Solution heuristic 1	Solution heuristic 2	MILP solution
Adder 32bits	2	1	127	127	**127**
Adder 32bits	20	9	5	5	**4**
Adder 64bits	2	1	265	267	**265**
Adder 64bits	20	9	10	12	**10**
Comparator	20	9	1	1	**1**
Multiplier	2	1	6350	5926	**5924**
Multiplier	20	9	105	116	**69**
AES	2	1	4504	5440	**3040**
AES	20	9	736	1600	**220±20**
AES [12]	23	11			**2**
DES	2	1	18399	18175	**18041**
DES	20	9	4435	4006	**440±20**
MD5	2	1	29084	34496	**28896**
SHA256	2	1	90825	97009	**88178**

Table 2: Minimal number of bootstrappings.

adapt those formulas for the case of the l_{max}-minimizing bootstrapping problem, as that could give constraints on the design of arithmetic circuits. Also, it should be interesting to go further in the complexity analysis of the problem by finding a monadic second order logic formulation, which would allow to apply many meta-theorems giving better insights on the problem. A future work is to provide an automatic tool that, given a circuit and a FHE scheme, could generate a new circuit with optimal bootstrapping placement.

Acknowledgments

We thank Rémi Coletta, Tancrède Lepoint and Guillerme Duvillie for their insights and expertise, and Sébastien Canard, Pascal Giorgi, Laurent Imbert and Fabien Laguillaumie for discussion and comments that greatly improved the manuscript.

References

1. Claude Berge. *Graphs*. North-Holland Mathematical Library. North Holland, 1985.
2. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 325–341, 2005.
3. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.

4. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106, 2011.
5. Bostjan Bresar, Frantisek Kardos, Ján Katrenic, and Gabriel Semanisin. Minimum k-path vertex cover. *Discrete Applied Mathematics*, 159(12):1189–1195, 2011.
6. Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 487–504, 2011.
7. Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 617–640, 2015.
8. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 10–18, 1984.
9. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
10. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.
11. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 465–482, 2012.
12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 850–867, 2012.
13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92, 2013.
14. Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 365–377, 1982.
15. Shai Halevi and Victor Shoup. Design and implementation of a homomorphic-encryption library.
16. Shai Halevi and Victor Shoup. Bootstrapping for helib. Cryptology ePrint Archive, Report 2014/873, 2014. `http://eprint.iacr.org/2014/873`.
17. Tancrède Lepoint and Pascal Paillier. On the minimal number of bootstrappings in homomorphic circuits. In *Financial Cryptography and Data Security - FC 2013 Workshops, USEC and WAHC 2013, Okinawa, Japan, April 1, 2013, Revised Selected Papers*, pages 189–200, 2013.
18. Uwe Naumann. DAG reversal is np-complete. *J. Discrete Algorithms*, 7(4):402–410, 2009.

19. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238, 1999.
20. Ronald Rivest, Leonard Adleman, and Martin Dertouzos. On data banks and privacy homomorphism. *Foundations on Secure Computation*, pages 168–177, 1978.
21. G. Sierksma. *Linear and Integer Programming: Theory and Practice, Second Edition*. Advances in Applied Mathematics. Taylor & Francis, 2001.
22. Nigel P. Smart and Stefan Tillich. Circuits of basic functions suitable for mpc and fhe. http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/.
23. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *IACR Cryptology ePrint Archive*, 2011:133, 2011.
24. Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 24–43, 2010.