Automated Exploration of Homomorphic Encryption Scheme Input Parameters

Cyrielle Feron, Loïc Lagadec, Vianney Lapotre

Automated Exploration of Homomorphic Encryption Scheme Input Parameters

Cyrielle FERON[a], Loïc LAGADEC[a], Vianney LAPOTRE[b]

[a] *UMR 6285, Lab-STICC, ENSTA Bretagne, 29806, Brest Cedex 9, France*
[b] *UMR 6285, Lab-STICC, Univ. Bretagne-Sud, F-56100, Lorient, France*

Abstract

Homomorphic Encryption (HE) aims to perform computations on encrypted data. Still in research stage, a lot of HE schemes have been created but their comparison remains costly as execution exhibits prohibitive costs. PAnTHErS is a HE schemes modeler. Modeling with a common formalism allows the evaluation of input parameters variation impact on performances of a HE scheme execution *i.e.* on its execution time and its memory cost. However, choosing a values interval that makes sense during the exploration phase still requires high expertise. This may prevent the tool from a large adoption by novice users. In this paper, we remind functionalities of PAnTHErS and present an automated exploration method. This exploration method will offer designers a simplified access to PAnTHErS functionalities.

Keywords: Homomorphic Encryption, Analysis, Modeling, Exploration

1. Introduction

Currently, Internet of things tends to increase the number of mobile data terminals as smartphones or tablets. Since their computational resources are limited, many services are deported in the cloud. This approach raises data security issues due to the fact that third parties may need to access sensitive or personal data for delivering services to the final user. Homomorphic encryption aims at answering these issues.

Homomorphic Encryption (HE) allows delegating computations on confidential data to a third party without loss of confidentiality. Figure 1 shows how a user can use a cloud computing service to modify his data with HE. User's data are secured during both transfers and computations.

To emphasis the interest of HE, let's consider a simple privacy and trust oriented example. Bob wants to use an health related application on his smartphone. He first has to enter his weight, his heart rate, etc. Then, the application (securely) transfers those data D to a server in charge of computing a personalized fitness program. In the context of classical cryptography, Bob provides his plain-data and has to trust the application. This is a major concern if he wants his data to be kept confidential.

However, HE solves this problem by keeping Bob's data D encrypted during the whole process. The considered application receives HE encrypted data $Enc(D)$ from Bob and has a function f to apply on it. This function can run on a distant server. The server processes $f(Enc(D))$ and returns the result to Bob. Bob is then able to decrypt the result by processing $Dec(f(Enc(D)))$ and then retrieves the processed data. Indeed, when a HE is used,

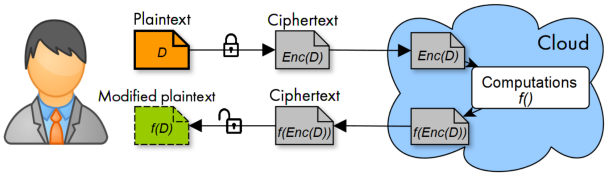


Figure 1: Homomorphic Encryption (HE) principle.

the equality

$$Dec(f(Enc(D))) = f(Dec(Enc(D))) \quad (1)$$

is true, and by definition, $Dec(Enc(D)) = D$. So, Bob recovers

$$Dec(f(Enc(D))) = f(Dec(Enc(D))) = f(D) \quad (2)$$

where $f(D)$ represents the answer of the application *i.e.* the referenced number of the proposed fitness program. On Figure 1, encryption is represented as a locked padlock and decryption by an open padlock and function f by the box *Computations*.

Unfortunately, despite all reported efforts in that field of cryptography, HE is still not usable in real cases. Indeed, expansion factor of ciphertext is important and so, length of encrypted data is huge compared to the plain data length. Moreover, significant encrypted data implies computational complexity and memory consumption issues. The cloud or the distant server must have enough resources to serve all users in terms of memory space and computational capacity. These issues prevent HE from a massive adoption despite its good properties regarding privacy, security and trust.

In this context, the adoption of HE relies on the capacity of the application designer to identify the scheme that

best fits a given application. For that purpose, several criteria are fixed by the application and server requirements. These criteria can include maximal execution time, multiplicative depth *i.e.* the maximal number of homomorphic operations that can be processed on encrypted data, memory space and security.

Due to memory and complexity issues, the analysis of all HE schemes in the literature implies considerable software executions time. For instance, in FV scheme [1], key generation and encryption of one bit takes 0.044 seconds with the tool Cingulata [2]. Encrypted result (resp. keys) has a size of 6 153 bytes (resp. 20.4 KB). On the other side, AES-CBC encryption with OpenSSL 1.0.1t takes less than 0.001 seconds to return a ciphertext (resp. key) of 32 bytes (resp. 16 bytes). Furthermore, design space exploration of homomorphic schemes means varying the input parameters of the HE scheme, which are at least three. As HE is very memory and time consuming, HE analysis by software simulations is not scalable. Besides, some HE schemes have several variants making exploration even more time consuming.

In this work, the tool PAnTHErS is presented: PAnTHErS helps the designer analyzing such HE schemes. Already usable by HE experts, PAnTHErS is extended by an exploration method for non-specialists. This exploration method allows to find the most interesting scheme and input parameters optimizing both complexity and memory cost for a given application.

The remainder of this paper is organized as follows. Section 2 presents some related work. Section 3 introduces the tool PAnTHErS. Then, Section 4 details the exploration method we implemented in PAnTHErS. Section 5 applies exploration method on four HE applications including one in a medical context. Section 6 recaps the different usages of PAnTHErS. Finally, Section 7 draws some conclusions and presents future work.

2. Related Work

In 1978, Rivest et al. [3] introduced the problem of creating a homomorphic encryption scheme which permits computations on encrypted data. At the time, they talked about *privacy homomorphism*. On the one hand, RSA cryptosystem [4] created by Rivest, Adleman and Shamir appears to be multiplicatively homomorphic. On the other hand, some schemes are additively homomorphic like Paillier cryptosystem [5]. Interest of full HE is to be multiplicatively and additively homomorphic. In 2009, Gentry constructed the first Fully Homomorphic Encryption (FHE) [6], which is based on ideal lattices. Since Gentry's work, numerous HE schemes have been created. FHE is achieved with Somewhat Homomorphic Encryption (SHE) scheme (bounded computation depth) combined with bootstrapping technique.

A SHE scheme only has a limited number of multiplications that can be applied on encrypted data: this number is called the multiplicative depth. Essentially, HE schemes rely on summing a noise to plaintext to produce a ciphertext. The final result can be decrypted only if the noise is below scheme specific threshold. Thus, noise expansion has to be controlled. In 2009, Gentry [6] used, on a SHE scheme, the bootstrapping technique that decreases (or "refreshes") noise in ciphertexts, without requiring the plaintext. He obtained a FHE scheme which can produce an unlimited number of multiplications.

More efficient noise management techniques have been introduced for HE schemes as modulus switching [7]. Also, ciphertext maintenance is needed during computations in order to preserve consistency (key switching [7], scale invariance technique [8], dimension modulus reduction [9], relinearization [9]). Moreover, SHE was combined with symmetric-key encryption to make trans-ciphering [10] in order to reduce the size of encrypted data that has to be transferred to the remote server.

Not all created schemes rely on ideal lattices like the one by Gentry. They are based on different hardness assumptions. HE schemes over the integers [11, 12, 13, 14, 15] are all based on the *approximate-GCD* problem. The idea of the *approximate-GCD* problem is to find the approximate common divisor of n random integers which are close multiples of the same large integer.

HE schemes on lattices are mainly based on *Learning With Error (LWE)* [16, 9, 8] and *Ring-LWE* (R-LWE) [17, 7, 1] problems. Introduced by Regev [18] in 2005, LWE problem is : given q and n, find $s \in \mathbb{Z}_q^n$ from "approximate" random linear equations on s. It means that each equation of s is perturbed by a small amount of noise. The R-LWE problem, also introduced by Regev [18], is basically to shift from $\mathbb{Z}_q^n$ to the ring $\mathbb{Z}_q[x]/\langle x^n+1\rangle$ in LWE problem definition. These two problems are post-quantum -meaning they resist to an attack by a quantum computer- what makes them very interesting in cryptography.

A method for LWE-based HE schemes, called approximate eigenvector, was introduced by Gentry, Sahai and Waters in 2013 [19]. In their scheme, the secret key is an approximate eigenvector of the ciphertext and the message is the eigenvalue. Error is introduced in eigenvector to enforce the security of the scheme on LWE. [20] is also based on this method. Introduced by Hoffstein, Pipher and Silverman in 1998 [21], NTRU is one of the first public-key cryptosystems based on lattices. It was then taken in reference to make HE schemes by [22, 23].

Some implementations of HE are now open-sources. HElib [24] is a well-known implementation of Brakerski, Gentry and Vainkuntanathan's scheme. HElib is a software C++ library, implemented by Halevi and Shoup. Also, a implementation of [11] by Coron et al. is available in Sage [25]. Aguilar-Melchor et al.[26] implemented a C++ library called NFLlib [27]. This is a library for ideal lattices cryptography in the polynomial ring $\mathbb{Z}_p[x]/\langle x^n+1\rangle$. This library allows the creation of FV-NFLlib [28] which is a implementation of [1] using NFLlib.

This section demonstrates that the quest for a practical

HE scheme has implied the creation of numerous schemes having relatively similar properties. Still, computational complexity and memory cost are the main issues of HE. Despite HE schemes similarities, no generic method allows analyzing or comparing HE schemes. Indeed, analyzing HE schemes means executing them, leading to a prohibitive cost while performing exploration and optimization tasks. To deal with this issue, this paper proposes to extend the PAnTHErS tool with an automated exploration of HE scheme input parameters allowing the extraction of an optimized configuration for a given application.

3. PAnTHErS

One limitation of HE is the hardness to estimate HE schemes complexity and memory cost and so, comparing them. Analysis requires HE experts to implement the scheme (whichever the used language and the fixed target architecture). Once implemented, which implies costly design and debug phases, numerous executions are still required to explore several input parameters impact.

PAnTHErS is an all-in-one solution to cut these costs. First, PAnTHErS offers a library to easily integrate HE schemes, alleviating the need for starting implementation from scratch. This strongly promotes legacy reuse. Also, thanks to a calibration phase, PAnTHErS provides facilities for rapid evaluation which only require a limited number of executions. Those evaluations allow the user to detect costly operations, highlighting algorithm sections on which refactoring effort must focus. Also, analysis makes it easier to choose which input parameters to concentrate on – if not proper values to select - according to implementation design and user's application needs. In addition, PAnTHErS allows comparison of schemes, and provides visual feedback, producing graphs showing complexity and memory cost. Thus, not only does the tool help HE experts to analyze and compare schemes but, it also meets the needs of designers, new to cryptography, wishing to use HE in future applications.

Before analyzing HE schemes, schemes must first be modeled into a common format. PAnTHErS (Prototyping and Analysis Tool for Homomorphic Encryption Schemes) aims at answering to this problem. The tool, illustrated in Figure 2, is divided in several steps including a modeling step, a theoretical analysis step and a calibration step. This section presents those different steps which are fully detailed in [29, 30].

3.1. Modeling

HE schemes are first modeled into a succession of algorithms. Three types of algorithms exist: Atomics (each one represents one operation), Specifics (representing series of instructions *i.e.* series of Atomics and/or Specifics) and HE basics. HE basics represents the five functions of a HE scheme: key generation, encryption, homomorphic addition, homomorphic multiplication and decryption. Specifics are composed of instructions which are identical in several HE schemes and thus, they are usable in different modeled schemes.

The created algorithms are stored in PAnTHErS library to be used in several scheme modeling. Then, the more the library possesses algorithms, the more it is possible to reuse algorithmic sequences, allowing to speed up modeling of new schemes.

Thanks to the modeling method (detailed in [29]), modeled schemes can be analyzed on a common basis. From models, PAnTHErS evaluates both computational complexity and memory cost of HE schemes. Analysis results allow HE scheme comparison in order to find the best scheme (*i.e.* implying the less complexity and/or the lowest memory consumption).

3.2. Theoretical analysis

Scheme theoretical analysis, deeply described in [29], is made in the worth case. Each algorithm A of PAnTHErS library *i.e.* Atomic, Specific and HE basic, is linked to two evaluation modules: computational complexity and memory consumption.

Complexity analysis module follows complexity evolution by counting the number of operations that are performed in A. In the case of memory analysis module, memory cost evolution is followed by listing (and updating the list) containing information about variables that are created or modified by A. At the end of an analysis, complexity and memory analysis modules return respectively the maximal number of multiplications performed during scheme execution and the maximal number of 32-bit integers simultaneously stored during scheme execution.

With these modules, a HE scheme can be analyzed without requiring its concrete execution. Results which are returned by theoretical analysis show evolution of computational complexity and memory cost in function of HE scheme input parameter variations. These results provide input parameter influence on the analyzed HE scheme flow. Moreover, it is also possible to observe computational complexity and memory cost after each call of algorithms that compose the scheme. It highlights costly operations (for instance, multiplication of polynomials), storage of temporary variables and outputs in a scheme (for example, keys, plaintexts and ciphertexts). Thus, this information could help designer while optimizing their scheme implementation.

However, theoretical analyses highlight input parameters influence on a scheme without reflecting a concrete scheme execution on a given implementation. Consequently, it is hard to estimate the scheme execution time from theoretical analysis of complexity. Moreover, theoretical analyses which are made in the worst case do not provide a satisfying representation of parameter concrete influence during practical usage of HE schemes. For instance, worst case analysis can show that a scheme S_1 has a less important complexity than a scheme S_2. Nonetheless, it is possible that S_1 execution will be longer than S_2 execution. This situation is not sufficient because PAnTHErS

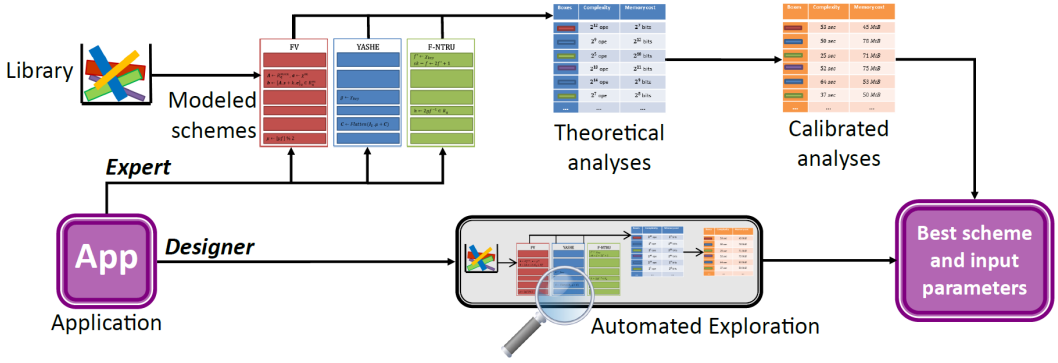


Figure 2: PAnTHErS overview.

aims at comparing several schemes between them. To do so, we provided a module to estimate execution time of schemes from theoretical analyses: this module is named p-calibration.

3.3. Calibration

p-calibration [30] allows converting computational complexity in execution time and performing evaluation of memory consumption in mebibytes (MiB). p-calibration is based on concrete executions of p sets of parameters called referencing points. A p-calibration needs p concrete scheme executions with the chosen referencing points. Then, computational complexity (resp. memory consumption) first expressed in number of multiplications (resp. number of 32-bits integers stored) is converted into execution time in seconds (resp. in MiB), based on concrete execution results.

2-calibration definition for complexity: Let p_1 and p_2 be referencing points such as $p_i = (CompTheo_i, CompPrac_i)$ with $CompTheo$ theoretical complexity analysis and $CompPrac$ execution time obtained after scheme execution. Referencing points are chosen considering the analyzed chosen values interval I of one scheme input parameter. Referencing point p_1 (resp. p_2) then corresponds to the complexity analysis of the parameter having the lowest value (resp. highest value) in I.

2-calibration consists in finding y_1 and y_2 variables such as:

$$\begin{cases} CompTheo_1 \times y_1 + y_2 = CompPrac_1 \\ CompTheo_2 \times y_1 + y_2 = CompPrac_2 \end{cases} \quad (3)$$

Then, all theoretical analyses $CompTheo_i$ linked to each value in I (including $CompTheo_1$ and $CompTheo_2$) are calibrated by computing $y_1 \times CompTheo_i + y_2$.

Error rate can be calculated between p-calibrated results and concrete execution times (resp. MiB). By taking a bigger p, error rate can be reduced but p-calibration execution is increased. Calibration experiments [30] showed that for $p = 2$, 3 and 4:

- Average of error rates does not exceed 10 % for $p = 2$, 7 % for $p = 3$ and 8 % for $p = 4$.
- At least 95 % of calibrated results have an error rate inferior to 20 % for $p = 2$, 12 % for $p = 3$ and 14 % for $p = 4$.

Considering those error rates and p-calibration execution time, $p = 2$ is used in the exploration step (section 4). A user who prefers lower error rates (despite a greater p-calibration execution time) can take a larger p.

When analyzing input parameters, the application designer has to carefully select set of parameters that respect the multiplicative depth required by the targeted application. This task requires a good knowledge of HE schemes. Thus, the designer has to be supported by a HE expert.

In order to make PAnTHErS usable by both HE expert and designers, we extend the tool with an exploration phase presented in the next section.

4. Exploration

The analysis of HE schemes is realized by the tool PAnTHErS [29]. First, a modeling step is necessary to represent a scheme into a succession of functions. Authors of [29] explained how a HE expert can process theoretical analyses on computational complexity and memory cost, while in [30], they show calibration procedure which calibrates theoretical analyses into concrete results that are expressed in seconds for execution time and mebibytes for memory consumption.

This section introduces an input parameter exploration method of HE schemes for a given application. This exploration method allows experts and non-experts in cryptography to elect the scheme along with input parameters that better fit the given application *i.e.* resulting the lowest execution time and/or the weakest memory consumption.

4.1. Exploration description

Exploration method consists in using modeling, analysis and calibration methods, as explained in the previous sections, in order to either reach the lowest cost in execution time and memory usage or to trade between the

two metrics. Exploration is divided in five steps as follows.

Step 1: *Application selection*

This step fixes an application requirement for the exploration. Concretely, the user defines the minimum multiplicative depth d required by the considered application. Alternatively, an interval of values for the multiplicative depth can be defined. During the exploration, configurations that do not provide the minimum required depth are automatically discarded.

The following steps (2, 3 and 4) are automatically executed for every modeled scheme available in PAnTHErS library. In our case, available schemes are FV [1], YASHE [22] and F-NTRU [23][1].

Step 2: *Find sets of parameters*

From its input parameters, a scheme implies a fixed multiplicative depth. Depth calculation is different for each HE scheme. Indeed, depth depends on generated noise in schemes. Authors give, in their articles, maximal bounds produced after: an encryption, a homomorphic addition and a homomorphic multiplication. These bounds allow calculating depth of the scheme from a fixed set of input parameters. Depth calculations of FV, YASHE and F-NTRU have been integrated in exploration method of the PAnTHErS tool.

Step 2 consists in finding each set of input parameters of the scheme S that implies the depth d of the application (or one depth in the interval of values chosen in step 1). For that, multiplicative depth of each set of input parameters is calculated. The sets of parameters which imply a depth from the ones chosen in step 1 are stored in a list named L.

Step 3: *Theoretical analyses*

List L, created in step 2, contains sets of valid parameters. Each set of parameters of L is theoretically analyzed as presented in 3.2.

Step 4: *Calibration*

Theoretical analyses are then calibrated with p-calibration method. p-calibration starts by executing p times the application with scheme S considering different sets of input parameters. Then, theoretical analyses are converted in concrete results (computational complexity is converted in execution time in seconds and memory consumption in mebibytes).

During step 4, only 2-calibrations are executed. Indeed, $p = 2$ has been chosen due to its acceptable produced error rates [30]. Moreover, a low value of p limits number of required application executions and thus it allows performing a faster exploration by reducing execution time of step 4.

Step 4.1: *Execution of the application with p referenced points*

In this step, the way we gather actual execution results depends on the number of varying input parameters.

General case with n variable parameters: Let $\mathcal{J}_2$ be the set which gathers sets of parameters named $j_i = (p_1, p_2, ..., p_n)$ which are chosen in step 2. Let $\forall k \in \{1, ..., n\}$, $min(p_k) = min\{p_k \in \mathcal{J}_2\}$ and $max(p_k) = max\{p_k \in \mathcal{J}_2\}$ be the minimum and the maximum of each parameter used in $j_i \in \mathcal{J}_2$. The executed sets of parameters are the sets corresponding to one combination of $min(p_k)$ and/or $max(p_k)$. Consequently, application will be executed with 2^n different sets of parameters j_{exec}. The sets of parameters j_{exec} are not necessary included in $\mathcal{J}_2$. Afterwards, $\mathcal{J}_{exec}$ represents the set of all j_{exec}.

F-NTRU case with $n = 2$: Parameters q (ciphertexts module) and w (integer for words decomposition) are input parameters in F-NTRU scheme. All $j_i \in \mathcal{J}_2$ have $(p_1, p_2) = (q, w)$ form. Minimum and maximum of each parameter are found: $min(q), max(q), min(w)$ and $max(w)$. Application is then executed with $2^n = 2^2 = 4$ different sets of parameters which are listed in equation 4. The sets of parameters j_{exec} in equation 4 are not necessarily included in $\mathcal{J}_2$.

$$\begin{aligned} A &= j_{exec_1} = (min(q), min(w)) \\ B &= j_{exec_2} = (max(q), min(w)) \\ C &= j_{exec_3} = (max(q), max(w)) \\ D &= j_{exec_4} = (min(q), max(w)) \end{aligned} \tag{4}$$

FV and YASHE case with $n = 3$: In addition to q and w, parameter t (plaintexts module) is a variable in FV and YASHE schemes. Each $j_i \in \mathcal{J}_2$ has $(p_1, p_2, p_3) = (q, w, t)$ form. Application is then executed with $2^n = 2^3 = 8$ different sets of parameters which are listed in equation 5.

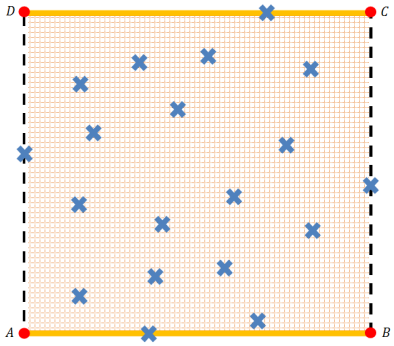
$$\begin{aligned} A &= j_{exec_1} = (min(q), min(w), min(t)) \\ B &= j_{exec_2} = (min(q), max(w), min(t)) \\ C &= j_{exec_3} = (max(q), max(w), min(t)) \\ D &= j_{exec_4} = (max(q), min(w), min(t)) \\ E &= j_{exec_5} = (min(q), min(w), max(t)) \\ F &= j_{exec_6} = (min(q), max(w), max(t)) \\ G &= j_{exec_7} = (max(q), max(w), max(t)) \\ H &= j_{exec_8} = (max(q), min(w), max(t)) \end{aligned} \tag{5}$$

Step 4.2: *Conversion of theoretical analyses*

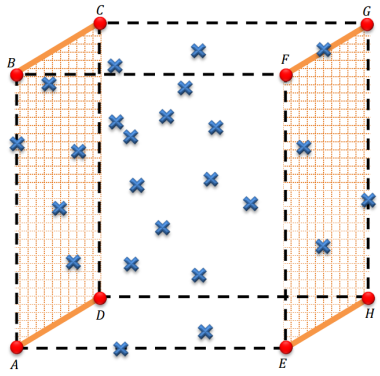
After executing sets of parameters in $\mathcal{J}_{exec}$, theoretical analyses of sets of parameters in $\mathcal{J}_2$ are then calibrated. Accordingly, the applied process for calibration is represented in Figure 3.

Figure 3a (resp. Figure 3b) corresponds to $n = 2$ case (resp. $n = 3$ case). In the two figures, blue crosses refer to sets of parameters in $\mathcal{J}_2$ and red points to sets of parameters in $\mathcal{J}_{exec}$. Coordinates of each red point are written in Figures 3a and 3b. Dotted and solid lines symbolize

[1] YASHE and F-NTRU are not secure since [31]. However, it does not impact on this paper contributions, especially on exploration feasibility.



(a) Representation for 2 parameters variation (F-NTRU).



(b) Representation for 3 parameters variation (FV and YASHE).

Figure 3: Representation of sets of parameters in $\mathcal{J}_2$, represented by blue points, and their calibration. Red points represents sets of parameters in $\mathcal{J}_{exec}$.

limits of the set $\mathcal{J}_2$. The goal, here, is to calibrate blue crosses within limits, thanks to their theoretical analysis and concrete application executions of each red point (*i.e.* each set of parameters drawn as red point).

$n = 2$ **case:** On Figure 3a *i.e.* in the case where $(p_1, p_2) = (q, w)$, application is executed for each of the four sets of parameters represented by red points. Two first calibrations are processed between :

• point $A = (min(q), min(w))$ and point $B = (max(q), min(w))$,

• point $D = (min(q), max(w))$ and point $C = (max(q), max(w))$.

These two first calibrations are presented in Figure 3a by an solid orange line (two opposite sides of the square). During the two calibrations, w is fixed and q is varied. In order to realize these calibrations, theoretical analyses of all sets of parameters included between A and B and between D and C must be first calculated.

Finally, from the four executions and the calibrated analysis, every theoretical analyses of blue crosses, *i.e.* sets of parameters in $\mathcal{J}_2$, can be calibrated. Each theoretical analysis of a set of parameters (q_i, w_i) is calibrated by taking $(q_i, min(w))$ and $(q_i, max(w))$ as referencing points. These last points correspond either to concrete results that are found after an execution or correspond to calibrated analyses. On Figure 3a, when analyses of red points and crosses on solid lines are calibrated, then theoretical analyses of all blue crosses on squared area can be calibrated. In fact, two 2-calibrations are processed successively: blue crosses calibration is executed after the first calibrations between A and B and between D and C.

$n = 3$ **case:** On Figure 3b *i.e.* in the case where $(p_1, p_2, p_3) = (q, w, t)$, application is executed with scheme by taking each of the eight red points. By fixing $p_3 = min(t)$ (resp. $p_3 = max(t)$) and by varying only q and w, theoretical analyses of points in $ABCD$ face (resp. opposite face $EFGH$) represented in Figure 3b can be calibrated as explained in $n = 2$ case. Before the calibration, theoretical analyses of all points in the two faces must be calculated. At this point, every theoretical analyses of points of (q_i, w_i, t_i) form with $t_i = min(t)$ or $t_i = max(t)$ are calibrated. Finally, each theoretical analysis of sets of parameters of (q_i, w_i, t_i) form with $t_i \neq min(t)$ and $t_i \neq max(t)$ can be calibrated by taking $(q_i, w_i, min(t))$ and $(q_i, w_i, max(t))$ as referencing points. In this case, three 2-calibrations are processed successively: blue crosses calibration is executed after $ABCD$ and $EFGH$ face calibration (corresponding to a $n = 2$ case calibration).

Step 5: *Sorting and electing of best scheme and input parameters*

Previous steps allow to get calibrated analysis of application for all schemes and for each set of parameters implying one of the multiplicative depths chosen in step 1 (either depth of the application, either one value from the chosen interval). The aim of step 5 is to elect the scheme and its sets of parameters implying best results in terms of execution time and/or memory consumption for the chosen application.

Beforehand, user can choose between optimizing lowest execution time or memory cost. In that case, parameters are sorted in function of their associated calibrated complexities (resp. memory cost). Then, the set of parameters which has the lowest execution time (resp. memory cost) is returned to the user.

If the user has no preference between complexity and memory cost, then Pareto efficiency [32] is applied as described in Definition 1.

Definition 1 (Pareto frontier in exploration method context)**.** *Let $\mathcal{J}_2$ be the set gathering all explored sets of parameters and $\mathcal{J}_P$ the set of points on Pareto frontier. Each $j \in \mathcal{J}_2$ (resp. $j_P \in \mathcal{J}_P$) produces an execution time t_j (resp. t_P) and a memory cost m_j (resp. m_P). Then $\forall j \in \mathcal{J}_2$ such as $j \notin \mathcal{J}_P$, it exists $j_P \in \mathcal{J}_P$ such as $t_P \leq t_j$ and $m_P \leq m_j$.*

Figure 4 gives an example of the Pareto frontier. Each set of parameters, represented by a square or a triangle, is placed depending on application execution time and application memory cost it implies. From Definition 1, sets of parameters on Pareto frontier are those illustrated by a

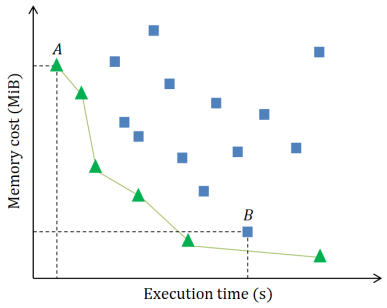


Figure 4: Graphical representation of a Pareto efficiency example. Each triangle and square represents a set of parameters in function of its execution time and memory consumption. Green triangles are sets of parameters on Pareto frontier.

green triangle on Figure 4. Graphically, it is possible to verify that a set of parameters is on Pareto frontier or not. For instance, the set of parameters A is on Pareto frontier because no other set of parameters are on the area limited by $(0,0)$, $(t_A,0)$, (t_A,m_A) and $(0,m_A)$. This means that no other set of parameters implies an execution time and a memory consumption lower than A. On the other hand, on Figure 4, a blue square is on the area limited by $(0,0)$, $(t_B,0)$, (t_B,m_B) and $(0,m_B)$; thus, set of parameters B is not on the Pareto frontier.

The sorting of the step 5 consists in finding sets of parameters that are on the Pareto frontier. These sets of parameters are then suggested to the user who will choose one of them by favoring the best execution time or the lowest memory cost.

4.2. Conclusion

The five steps of our exploration method do not require user intervention. User only gives application to be analyzed (step 1). Then, exploration analyzes all combinations of schemes and sets of parameters which respect the application requirement (*i.e.* multiplicative depth) in order to find the ones which imply the most interesting execution times and memory costs. Pareto efficiency is used to elect the best sets of parameters among those analyzed. These sets of parameters and their estimations are then shown to the user.

5. Use cases

Previous section describes the automated exploration method. Exploration allows a non-expert in HE to identify the most interesting scheme (implying the lowest execution time and/or memory cost) for a given application.

Exploration can be applied on any kind of application. In order to show exploration capabilities, four applications have been considered, each one with a different numbers of operations and a different multiplicative depth.

The first part of this section introduces the four applications while the second part presents the results obtained by applying our proposed exploration method.

5.1. Applications definition

Four applications have been created in order to process exploration method on different use cases. Applications exhibit different numbers of operations and different multiplicative depth. Applications features are detailed in Table 1.

Table 1: Applications information (number of called HE basic and depth)

Applications	*FiveHB*	*Pedagogic*	*Croissant*	*Medical*
# of *KeyGen*	1	1	1	1
# of *Encrypt*	1	3	35	235
# of *Add*	1	5	80	294
# of *Mult*	1	12	14	106
# of *Decrypt*	1	1	8	4
Total number of HE basics	5	22	138	640
Depth	1	10	6	12

5.1.1. Artificial applications

FiveHB application is a succession of the five HE basic functions in the following order: key generation *KeyGen*, encryption *Encrypt*, homomorphic addition *Add*, homomorphic multiplication *Mult* and decryption *Decrypt*.

Pedagogic application has an arbitrary number of homomorphic operations. It does not process any useful functionality for a user. It was only created to evaluate an application having a more important depth than *FiveHB* application. *Pedagogic* application thus has a depth of 10 against 1 only for *FiveHB*.

Pedagogic application has, as input parameters, three plaintexts which are polynomials. The three plaintexts m_1, m_2 and m_3 are encrypted as c_1, c_2 and c_3. Then, the following operations are processed:

$$c = c_1^3 \cdot c_2^2 \cdot c_3^2 \cdot S \cdot T \cdot (S + c_3) \cdot (S + c_2 + c_3) \qquad (6)$$

with $S = c_1 \cdot T + c_1 + c_2$ and $T = c_1 \cdot c_2 + c_3$. Result c is then decrypted.

5.1.2. Concrete applications

Croissant application allows to know how many decades of kcal a person will consume by eating n croissants. Parameter n is considered has an 8-bit input parameter.

The computations performed by this application are simple: knowing that eating a croissant implies consuming 231 kcal, only a multiplication is necessary to obtain the final quantity. Moreover, to make easier computations, application rounds quantities to the decade of kcal. Thus, a croissant is equivalent to 23 decades of kcal. Final application consists in taking as input a quantity n of croissants and then, this value is multiplied by 23.

FV [1] and YASHE [22] schemes have a variable named t which is the plaintexts module (each plaintext is reduced modulo t). This parameter is one explored parameter in exploration method. Final result of *Croissant* application must be an integer greater than 23. Thus, if t is chosen

Algorithm 1 C++ source code of *Croissant* application

```
1: Integer8 nb_croissant, c;
2: cin>>nb_croissant;
3: c = nb_croissant*Integer8(23);
4: cout<<c;
```

small (*e.g.* $t = 2$), application will not decrypt the good value because final result is reduced modulo t. On an other hand, if t is chosen large (> 23), it must be taken larger enough to correctly respond to the user. For instance, if the user eats 5 croissants, application must return the value $5 \times 23 = 115$ and so, scheme must use $t > 115$. Consequently, in order to guarantee a good functioning, the chosen value of t limits the maximal value of parameter n which is input by the user.

A method to overcome this limitation is to perform homomorphic operations on an application input bits. By directly working on bits, there is no restriction on input parameters. Nevertheless, this implies that t parameter must always be equal to 2 because every output message will be a single bit (either 0, either 1).

In order to apply this method, an application has to be converted into a boolean circuit. The tool Cingulata [2] allows to convert a C++ implemented application into a boolean circuit. This circuit is only composed of OR, XOR, AND and NOT gates. The application circuit is then saved into a BLIF (Berkeley Logic Interchange Format) file [33].

Others tools exist and can be use to perform a conversion from high level application (C/C++ language) to boolean circuit including, for example, the tool Madeo [34].

We implemented *Croissant* application in C++ as described in Algorithm 1.

The tool Cingulata gives boolean circuit of *Croissant* application from its C++ description. Cingulata returns an optimized and a non-optimized version of the boolean circuit. We consider the optimized version that we convert into a succession of HE basics, that is suitable for PAnTHErS. For that purpose, BLIF format is converted in succession of XOR, AND, NOT and OR gates. Then, each XOR gate (resp. AND) is converted in *Add* function (resp. *Mult* function). Furthermore, NOT(a) = a XOR 1 is converted in *Add*(a, *Encrypt*(1)) and OR(a, b) = AND(AND(a,b), XOR(a,b)) is converted in *Mult*(*Mult*(a,b), *Add*(a,b)).

Number of HE basics highly increases when *Croissant* application is converted in a boolean circuit. Indeed, C++ *Croissant* application calculates one multiplication between two 8-bit integers. In homomorphic domain, this application has a succession of 138 HE basics. If we take 16-bit integers as inputs, the application would perform 262 HE basics, roughly twice HE basics with 8-bit integers.

However, 8-bit integers usage reduces number of possible values for parameter n (number of croissants). As it is implemented on 8 bits, *Croissant* application result will not exceed 256 decades of kcal, that is to say 11.13 croissants. Thus, in this use case, n is limited to 11.

Algorithm 2 *Medical* application pseudo-code (described in [35] and available in [2])

Require: gender, age, medicalHistory, smoker, diabetic, HDLcholesterol, highBloodPressure, weight, alcoholConsumption, physicalActivity, height
Ensure: riskFactor (between 0 and 9)

```
1: riskFactor = 0
2: If gender == MAN AND age > 50:
3:         riskFactor += 1
4: If gender == WOMAN AND age > 60:
5:         riskFactor += 1
6: If medicalHistory:
7:         riskFactor += 1
8: If smoker:
9:         riskFactor += 1
10: If diabetic:
11:         riskFactor += 1
12: If highBloodPressure:
13:         riskFactor += 1
14: If HDLcholesterol < 40:
15:         riskFactor += 1
16: If daily physicalActivity < 30 minutes:
17:         riskFactor += 1
18: If weight - 10 ¿ height:
19:         riskFactor += 1
20: If gender == MAN AND alcoholConsumption > 3
    glasses/day:
21:         riskFactor += 1
22: If gender == WOMAN AND alcoholConsumption >
    2 glasses/day:
23:         riskFactor += 1
```

Medical application is presented in [35]. This application aims at computing a risk factor of cardiology disease from medical information of a patient. Calculated risk factor is an integer between 0 (weak risk) and 9 (strong risk). *Medical* application pseudo-code is written in Algorithm 2.

As for *Croissant* application, *Medical* application has been converted into a boolean circuit by the tool *Cingulata* [2] and t is fixed to 2 for equivalent reasons.

5.2. Application exploration results

This section introduces exploration method results of three schemes (FV [1], YASHE [22] and F-NTRU [23]) on the four applications: *FiveHB*, *Pedagogic*, *Croissant* and *Medical*. Hardware and software configuration for experiments is showed in Table 2. For each exploration, we have allocated 2 cores and 128 GB of RAM. The Sage 8.0 [36] open-source computer program has been launched in a Docker container. The ranges of parameters exploration for each scheme of each application are defined in Table 3.

Table 2: Hardware and software configuration of exploration execution environment.

Processors	8 × Intel Xeon E7-8890 v4 @ 2.20 GHz
Number of cores	8 × 24
RAM	192 × 32 Gb / DDR4
Operating system	Red Hat Enterprise Linux 7.5 64-bit
Software environment	Sage 8.0 Python 2.7.10

Table 3: Minimum and maximum bounds reached by input parameters of each scheme.

Application	**Scheme**	$\log(q)$		$\log(w)$		t	
		Min	**Max**	**Min**	**Max**	**Min**	**Max**
FiveHB	FV	43	148	2	99	2	49
	YASHE	45	162	2	99	2	49
	F-NTRU	18	124	2	99	-	-
Pedagogic	FV	297	418	2	99	2	14
	YASHE	308	443	2	99	2	5
	F-NTRU	102	149	2	7	-	-
Croissant	FV	189	287	2	99	2	2
	YASHE	196	308	2	99	2	2
	F-NTRU	61	149	2	19	-	-
Medical	FV	376	476	2	99	2	2
	YASHE	389	499	2	99	2	2
	F-NTRU	121	149	2	4	-	-

Table 4 gives number of explored parameters, exploration execution time and required time to perform all concrete executions for each application. A speedup factor is defined for each application. This factor is the ratio between concrete execution time and our exploration execution time.

Speedup factor ranges from 7.4 (*FiveHB*) to 41.8 (*Pedagogic*). Speedup of all gathered applications is 18.6. A study per scheme of application speedup factor shows, in Table 5, that in general F-NTRU (resp. YASHE) has the lowest (resp. the highest) yield. For instance, speedup factor of F-NTRU for *Medical* application is 5.9 and the one of YASHE for the same application is 137.5. This difference is mainly due to the number of analyzed sets of parameters (only 26 for F-NTRU against 2936 for YASHE) and to concrete execution times that are required for calibration: in average, an execution of *Medical* application with F-NTRU scheme is 53 times slower than with YASHE scheme.

5.2.1. Error rates study

Figure 5 illustrates spreading of error rate that were noted between calibrated analysis of exploration and concrete executions. Error rates of Figure 5 are showed by application exploration *i.e.* all schemes combined.

More than 95 % of error rates, for each application, are under 45 %. Moreover, the average of error rates for calibrated complexity (resp. memory cost) is :

- 14.7 % (resp. 16.6 %) for *FiveHB* application,
- 7.5 % (resp. 3.8 %) for *Pedagogic* application,
- 7.4 % (resp. 5.1 %) for *Croissant* application,
- 7.5 % (resp. 1.3 %) for *Medical* application.

All those means combined are under 17 %. Considering application error rates for each separated scheme, the observations on error rates above are also verified except for *Croissant* application with F-NTRU scheme. For this specific case, more than 50 % of error rates produced by the exploration are superior to 45 % for calibrated complexity. Regarding memory cost, only 70 % of values are actually inferior to 45 %.

Those error rates can be decreased by executing application with more referencing points. As we took $p = 2$, the number of referencing points is limited to 4 for F-NTRU and 8 for FV and YASHE (see section 4) but it could be higher, depending on the total number of sets of parameters to explore or depending on the size of interval of varied parameters. By having a large number of referencing points, p-calibration will be thinner and thus will induce lower error rates. However, the more there is referencing points, the longer will be the exploration execution.

5.2.2. Optimal sets of parameters study

Sets of parameters obtained by application exploration are listed in Table 6. Selected sets of parameters by exploration are for YASHE scheme in the case of *FiveHB* application and for FV scheme in the case all the three other applications.

Optimal sets of parameters are obtained after concrete execution of all schemes for each application. As for selected sets of parameters, those optimal points are sets of parameters for YASHE in the case of *FiveHB* application and for FV scheme in the case all the three other applications. FV scheme offers a faster calculation and less memory consumption than YASHE and F-NTRU scheme for more realistic applications.

Figure 6 illustrates spreading of analysis of explored sets of parameters (blue crosses). Each set of parameters is place on graph in function of its concrete execution time (horizontal axis) and its actual memory cost (vertical axis). On the figure, red squares are sets of parameters of optimal points *i.e.* on the Pareto frontier of concrete execution results. Green circles are sets of parameters elected by exploration method *i.e.* on the Pareto frontier of exploration results. Those last points are placed depending on their concrete execution time and memory cost.

A set of parameters or optimal point is a set of parameters implying a weak memory consumption and/or a low execution time or a compromise between both. In other words, optimal points are placed in the bottom-left corner of graphs in Figure 6. Sets of parameters that are obtained by exploration are actually well placed in the bottom-left corner on graphs in Figure 6 and so, are close to optimal points. In order to evaluate the efficiency of the exploration method and the closeness of points, we calculated the distance between selected exploration points and optimal executed points.

The points in $\mathcal{P}$ are optimal if they are on the Pareto frontier. Points that are chosen in second position are points on the second Pareto frontier *i.e.* points on the

Table 4: Information about exploration execution for each application.

Application	Number of sets	Time for finding sets	Exploration time of all sets of parameters	Total time of execution	Speedup
FiveHB	186082	5 h 20 min 3 s	19 h 45 min 46 s	147 h 7 min 26 s	7.4
Pedagogic	22969	2 h 49 min 37 s	24 h 29 min 36 s	1024 h 31 min 17 s	41.8
Croissant	5684	2 h 27 min 41 s	29 h 53 min 17 s	289 h 16 min 2 s	9.7
Medical	5808	1 h 32 min 44 s	319 h 29 min 6 s	5872 h 9 min 56 s	18.4
TOTAL	220543	12 h 10 min 5 s	393 h 37 min 45 s	7333 h 4 min 42 s	18.6

Table 5: Exploration speedup for each application depending on explored schemes.

Scheme \ Application	FiveHB	Pedagogic	Croissant	Medical
FV	1.8	19.9	16.4	36.2
YASHE	13.3	61.0	73.6	137.5
F-NTRU	4.1	8.8	6.3	5.9

Pareto frontier of all executed sets of parameters $\mathcal{J}_2^{exec}$ minus points in the first Pareto frontier $\mathcal{P}$. And so on, we can define the n-th Pareto frontier, detailed below.

Definition 2 (n-th Pareto frontier)**.** *Let $\mathcal{P}_1$ be the set of points on the Pareto frontier of $\mathcal{J}_2^{exec}$. Let $\mathcal{P}_1 = FrontPareto(\mathcal{J}_2^{exec})$ with $FrontPareto(E)$ which returns the set of points on the Pareto frontier of the set E. The set of points on the n-th Pareto frontier is:*

$$\mathcal{P}_n = FrontPareto\left(\mathcal{J}_2^{exec} \setminus \bigcup_{i=1}^{n-1} \mathcal{P}_i\right). \tag{7}$$

From the index of Pareto frontier explained above, we calculated a ranking position for sets of parameters as defined as below.

Definition 3 (Ranking of a set of parameters)**.** *Let j be a set of parameters in $\mathcal{P}_n$ with $\mathcal{P}_n$ the set of points on the n-th Pareto frontier. The ranking position of j is:*

$$Card\left(\bigcup_{i=1}^{n-1} \mathcal{P}_i\right) + 1, \tag{8}$$

with $Card(E)$ the cardinal number of the set E.

This ranking position c allows to tell if a set of parameters j is the c-th chosen points on a total of $Card(\mathcal{J}_2^{exec})$ points. In other words, the set of parameters j is placed at p % in the total points ranking with:

$$p = \frac{c-1}{Card(\mathcal{J}_2^{exec})} \times 100 \tag{9}$$

Table 6 gives ranking of each exploration result. The ranking of sets of parameters is determined from concrete execution results (execution time and memory cost) and their index of Pareto frontier (Definition 2). This ranking is calculated, on the one hand, with equation 8 (index of ranking on the total of explored sets of parameters) and, on the other hand, with equation 9 (index of ranking in percentages, *e.g.* 0 % is the first ranking place and 100 % the last one).

Table 6 also informs about the distance between:

• execution time of a set of parameters and minimal execution time of optimal points,

• memory consumption of a set of parameters and minimal memory cost of optimal points.

For the four gathered application, 6 sets of parameters on 7 are placed in the six first percents of all explored sets of parameters. Among those sets of parameters, 4 are in the first three percents.

The results presented in this section show that the proposed exploration method provides a good solution for a designer to fastly determine, for a given application, the HE scheme and its input parameters providing interesting performances. Moreover, if required, the p-calibration step can be tuned to provide a configuration even closer from the actual optimal.

6. PAnTHErS usage

The tool PAnTHErS is freely available [37]. It now gathers modeled schemes, theoretical analysis method, p-calibration and exploration method. It is implemented in Python using Sage [36]. PAnTHErS aims at answering to two types of users' profiles: designer and HE expert.

This section presents goals of the two users and how they can use the tool PAnTHErS in order to accomplish their aims.

6.1. Designer

A designer wants to use HE in his future application. This user has no specific skills in cryptography. The application designer/programmer needs HE in order to secure data of his clients.

In order to choose the HE scheme, the designer uses PAnTHErS and its exploration method. The designer must define his application in the tool as a succession of HE basic functions. The designer can use the tool Cingulata [2] to convert his C++ application into a boolean circuit. Each boolean operation can be easily interpreted has HE basic functions (*e.g.* XOR corresponds to homomorphic addition). Finally, the designer launches the exploration

Table 6: Results returned by exploration for each application, ranking of exploration selected points and distance from those selected points to optimal results.

Application	Scheme	$\log(q)$	$\log(w)$	t	Time (s)			Memory cost (MiB)			Ranking		Distance with optimal point	
					Estim.	Exec.	Error	Estim.	Exec.	Error	Position	In %	Time (s)	Memory cost (MiB)
FiveHB	YASHE	46	10	2	0.13	0.13	10.8 %	2.77	2.5	10.9 %	1/186082	0	0	0
	YASHE	75	38	3	1.77	1.73	1.79 %	2.77	4.5	38.51 %	867/186082	0.47	1.61	2
Pedagogic	FV	297	23	2	10.42	10.26	1.62 %	99.37	93.8	5.94 %	119/22969	0.51	0.12	16.8
	FV	339	68	2	10.75	11.53	6.78 %	83.93	80.2	4.66 %	736/22969	3.2	1.39	3.2
Croissant	FV	243	81	2	12.91	13.15	1.82 %	39.26	39.8	1.37 %	309/5684	5.42	1.63	4.3
Medical	FV	393	44	2	167.25	175.23	4.55 %	269.55	265.5	1.53 %	1734/5808	29.84	24.27	30.9
	FV	437	88	2	171.02	167.45	2.14 %	237.29	240	1.13 %	133/5808	2.27	16.49	5.4

Table 7: Returned results after concrete execution of all sets of parameters of FV, YASHE and F-NTRU for the four studying applications.

Application	Scheme	$\log(q)$	$\log(w)$	t	Time (s)	Memory cost (MiB)
FiveHB	YASHE	46	10	2	0.13	2.5
Pedagogic	FV	307	35	2	10.32	83.6
	FV	310	35	2	10.14	84.1
	FV	312	40	2	10.39	79.9
	FV	313	38	2	10.26	83.6
	FV	314	40	2	10.37	81.1
	FV	315	40	2	10.38	80.3
	FV	315	43	2	10.37	80.6
	FV	320	46	2	10.51	79.4
	FV	328	56	2	11.37	77
	FV	331	57	2	10.87	79.3
	FV	337	57	3	10.95	78.3
	FV	338	63	2	11.08	77
	FV	342	70	2	11.01	77.4
Croissant	FV	253	89	2	11.80	35.5
	FV	265	94	2	11.51	37.2
	FV	276	99	2	11.68	36.5
Medical	FV	412	58	2	150.95	258.7
	FV	449	84	2	153.47	245.3
	FV	449	90	2	155.00	234.6
	FV	451	98	2	154.39	235.8
	FV	460	99	2	153.50	237.3

method which will return the most interesting scheme and input parameters.

PAnTHErS is accessible and easy to use (especially for exploration) thanks to its graphical interface, implemented with TKinter library [38].

6.2. HE expert

A HE expert aims at analyzing and/or improving his scheme. Indeed, the analysis of a scheme allows the expert to profile his internal functions in terms of execution time and memory consumption. Thus, the expert could optimize those functions, for example through parallelization or hardware acceleration. Moreover, HE expert could compare his scheme performances versus those of other schemes which are already available in PAnTHErS library.

In order to analyze and/or compare schemes, HE expert can use the PAnTHErS graphical interface to perform fine grained analyses. Moreover, tutorials [39] are available to help an expert who would like to add a new scheme into PAnTHErS.

Adding HE schemes consists in implementing classical algorithms (for instance those referred in section 2) but by using Atomic functions of PAnTHErS. Those functions act as substitutes for the implemented algorithm's elementary operations (*e.g.* addition, multiplication,...). Sage library [36] allows the user to create mathematics sets such as polynomial rings or fields. PAnTHErS supports user design of Atomic (or Specific) functions that are supported by Sage library. Thus, PAnTHErS easily allows the modeling of all kinds of HE schemes (based on LWE, NTRU, approximate-GCD) as well as algorithms from other application domains.

The HE expert will not find in PAnTHErS a turnkey solution, but, instead an open and extensible platform: new metrics, new analyses, new external modules can be added or plugged in, with potential reuse of existing ones.

A first new metric that would make sense is noise analysis to help calculating multiplicative depth of new schemes without even having the mathematical equation. This analysis could help HE experts in their scheme creation with function per function analysis.

7. Conclusion and future work

Through a modeling and analysis tool named PAnTHErS, a user (designer or HE expert) is now able to analyze a HE scheme or an application. PAnTHErS offers the possibility to launch several types of analysis: theoretical or concrete, for a scheme or an application, partial exploration (one parameter variation) or complete exploration (all parameters variation). The user chooses what kind of analysis best fits his needs.

The new functionality of PAnTHErS, the exploration method, allows, for a given application, to analyze every HE schemes and sets of input parameters implying the application multiplicative depth. Thus, it enables to elect scheme inducing the most interesting results (a low execution time, a weak memory consumption or any tradeoff in between).

This method has been tested on four applications, exhibiting different features. Exploration highlights YASHE [22] for *FiveHB* application and FV [1] for the three other applications. Comparing to application concrete executions with scheme and sets of parameters to explore, exploration of all sets of parameters is from 7.4 to 41.8 times faster (see Table 4 in section 5.2).

Among 7 sets of parameters selected by exploration (four gathered applications), 6 are close to optimal *i.e.* placed in the first six percents of sets of parameters ranking. The ranking is determined from concrete execution

results. Only one set of parameters has a deceptive ranking: it is out of the 20 first percents.

In the short term, future work will focus on refining calibration step on the exploration method by increasing the number of referencing points. In the long term, an interfacing between PAnTHErS and Cingulata [2] could be first created to easily add new applications into PAnTHErS in order to explore and/or analyze them. Secondly, execution of applications and schemes could benefit from hardware acceleration. For instance, Migliore et al. [40] created a co-design approach to speedup polynomials multiplication. PAnTHErS could take benefit from this work for faster evaluations.

References

[1] J. Fan, F. Vercauteren, Somewhat Practical Fully Homomorphic Encryption, IACR Cryptology ePrint Archive 2012 (2012) 144.

[2] CEA-LIST Crypto Team, Cingulata: open-source compiler toolchain, https://github.com/CEA-LIST/Cingulata/, commit fbc40d9e9948630047e4a60312c925594d14fc46, 2017.

[3] R. L. Rivest, L. Adleman, M. L. Dertouzos, On data banks and privacy homomorphisms, Foundations of secure computation 4 (1978) 169–180. URL: https://pdfs.semanticscholar.org/3c87/22737ef9f37b7a1da6ab81b54224a3c64f72.pdf.

[4] R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (1978) 120–126. URL: http://dl.acm.org/citation.cfm?id=359342.

[5] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 1999, pp. 223–238. URL: http://link.springer.com/chapter/10.1007/3-540-48910-X_16.

[6] C. Gentry, Fully homomorphic encryption using ideal lattices., in: STOC, 2009, pp. 169–178.

[7] Z. Brakerski, C. Gentry, V. Vaikuntanathan, (Leveled) Fully Homomorphic Encryption without Bootstrapping, in: ITCS, 2012.

[8] Z. Brakerski, Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP, in: Proc. CRYPTO, Santa Barbara, CA, USA, 2012, pp. 868–886. doi:10.1007/978-3-642-32009-5_50.

[9] Z. Brakerski, V. Vaikuntanathan, Efficient Fully Homomorphic Encryption from (Standard) LWE, FOCS 2011 (2011) 97–106. doi:10.1109/FOCS.2011.12.

[10] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, R. Sirdey, Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression, in: FSE, 2016.

[11] M. V. Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan, Fully Homomorphic Encryption over the Integers, in: Proc. EUROCRYPT, French Riviera, 2010, pp. 24–43. doi:10.1007/978-3-642-13190-5_2.

[12] J. Coron, A. Mandal, D. Naccache, M. Tibouchi, Fully Homomorphic Encryption over the Integers with Shorter Public Keys, in: Proc. CRYPTO, Santa Barbara, CA, USA, 2011, pp. 487–504. doi:10.1007/978-3-642-22792-9_28.

[13] J. Coron, D. Naccache, M. Tibouchi, Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers, in: Proc. EUROCRYPT, Cambridge, UK, 2012, pp. 446–464. doi:10.1007/978-3-642-29011-4_27.

[14] J. H. Cheon, J. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, A. Yun, Batch Fully Homomorphic Encryption over the Integers, in: Proc. EUROCRYPT, Athens, Greece, 2013, pp. 315–335. doi:10.1007/978-3-642-38348-9_20.

[15] J. Coron, T. Lepoint, M. Tibouchi, Scale-Invariant Fully Homomorphic Encryption over the Integers, in: Proc. Public-Key Cryptography - PKC, Buenos Aires, Argentina, 2014, pp. 311–328. doi:10.1007/978-3-642-54631-0_18.

[16] R. Lindner, C. Peikert, Better Key Sizes (and Attacks) for LWE-Based Encryption, in: Proc. - CT-RSA 2011, San Francisco, CA, USA, 2011, pp. 319–339. doi:10.1007/978-3-642-19074-2_21.

[17] Z. Brakerski, V. Vaikuntanathan, Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages, in: Proc. CRYPTO, Santa Barbara, CA, USA, 2011, pp. 505–524. doi:10.1007/978-3-642-22792-9_29.

[18] O. Regev, On lattices, learning with errors, random linear codes, and cryptography, in: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005, 2005, pp. 84–93. URL: http://doi.acm.org/10.1145/1060590.1060603. doi:10.1145/1060590.1060603.

[19] C. Gentry, A. Sahai, B. Waters, Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based, in: Proc. CRYPTO, Santa Barbara, CA, USA, 2013, pp. 75–92. doi:10.1007/978-3-642-40041-4_5.

[20] Z. Brakerski, V. Vaikuntanathan, Lattice-based FHE as secure as PKE, in: Proc. Innovations in Theoretical Computer Science, Princeton, NJ, USA, 2014, pp. 1–12. doi:10.1145/2554797.2554799.

[21] J. Hoffstein, J. Pipher, J. H. Silverman, NTRU: A ring-based public key cryptosystem, in: Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings, 1998, pp. 267–288. doi:10.1007/BFb0054868.

[22] J. W. Bos, K. E. Lauter, J. Loftus, M. Naehrig, Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme, in: Cryptography and Coding IMA, 2013.

[23] Y. Doröz, B. Sunar, Flattening NTRU for evaluation key free homomorphic encryption, IACR Cryptology ePrint Archive (2016).

[24] S. Halevi, V. Shoup, HElib - An implementation of homomorphic encryption, https://github.com/shaih/HElib, 2014.

[25] J.-S. Coron et al., An implementation of the DGHV fully homomorphic scheme, https://github.com/coron/fhe, 2012.

[26] C. A. Melchor, J. Barrier, S. Guelton, A. Guinet, M. Killijian, T. Lepoint, Nfllib: Ntt-based fast lattice library, in: Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings, 2016, pp. 341–356. doi:10.1007/978-3-319-29485-8_20.

[27] C. Aguilar-Melchor, J. Barrier, S. Guelton, A. Guinet, M.-O. Killijian, T. Lepoint, Nfllib: Ntt-based fast lattice library, in: Cryptographers' Track at the RSA Conference, Springer, 2016, pp. 341–356.

[28] CryptoExperts, FV-NFL : Library implementing the Fan-Vercauteren homomorphic encryption scheme, https://github.com/CryptoExperts/FV-NFLlib#fv-nfllib, 2016.

[29] C. Feron, V. Lapotre, L. Lagadec, PAnTHErS: A Prototyping and Analysis Tool for Homomorphic Encryption Schemes, in: Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT, Madrid, Spain, 2017, pp. 359–366.

[30] C. Feron, V. Lapotre, L. Lagadec, Fast evaluation of homomorphic encryption schemes based on ring-lwe, in: New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on, IEEE, 2018, pp. 1–5.

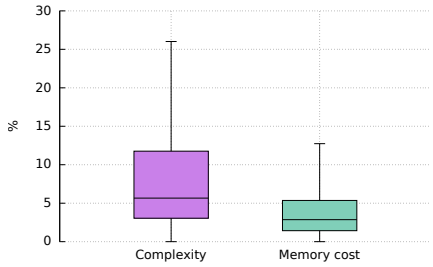
[31] M. Albrecht, S. Bai, L. Ducas, A subfield lattice attack on overstretched ntru assumptions, in: Annual Cryptology Conference, Springer, 2016, pp. 153–178.

[32] V. Pareto, Cours d'économie politique, volume 1, Librairie Droz, 1964.

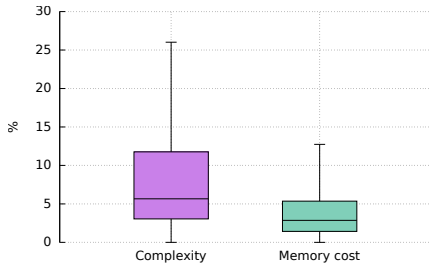
[33] U. o. C. Berkeley, Berkeley logic interchange format (blif), Oct Tools Distribution 2 (1992) 197–247.

[34] L. Lagadec, B. Pottier, O. Villellas-Guillen, An lut-based high level synthesis framework for reconfigurable architectures,

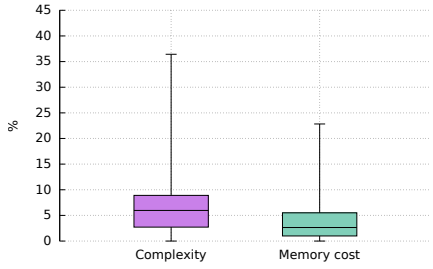
Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation (2003) 19–39.
[35] S. Carpov, T. H. Nguyen, R. Sirdey, G. Constantino, F. Martinelli, Practical privacy-preserving medical diagnosis using homomorphic encryption, in: Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on, IEEE, 2016, pp. 593–599.
[36] W. A. Stein, T. Abbott, M. Abshoff, SageMath, `http://www.sagemath.org/`, 2016.
[37] C. Feron, PAnTHErS (Prototyping and Analysis Tool for Homomorphic Encryption Schemes), `https://github.com/cferon/PAnTHErS`, 2018.
[38] J. W. Shipman, Tkinter 8.4 reference: a gui for python, New Mexico Tech Computer Center (2013).
[39] C. Feron, PAnTHErS: User Guide, Technical Report, ENSTA Bretagne ; Lab-STICC, 2018. URL: `https://hal.archives-ouvertes.fr/hal-01867913`.
[40] V. Migliore, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, G. Gogniat, Hardware/software co-design of an accelerator for FV homomorphic encryption scheme using karatsuba algorithm, IEEE Trans. Computers 67 (2018) 335–347. doi:`10.1109/TC.2016.2645204`.



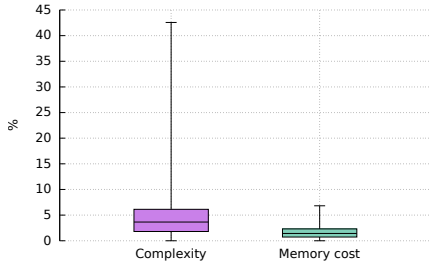
(a) *FiveHB* (98 %).



(b) *Pedagogic* (99 %).

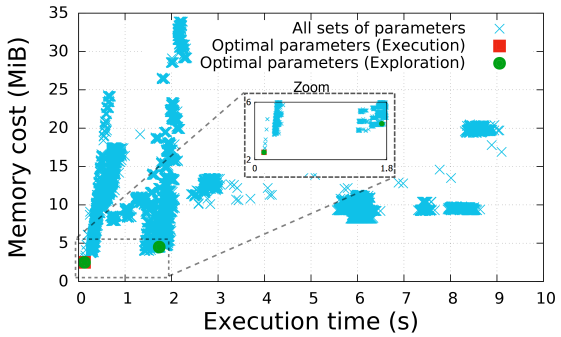


(c) *Croissant* (97 %).

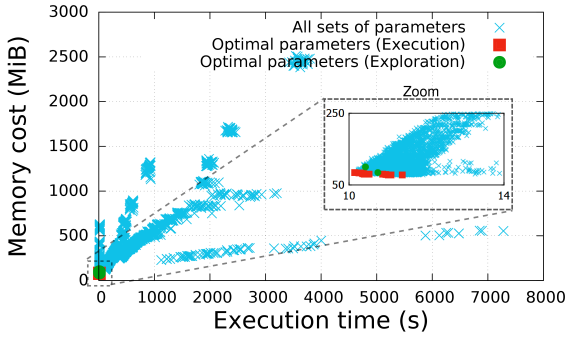


(d) *Medical* (99 %).

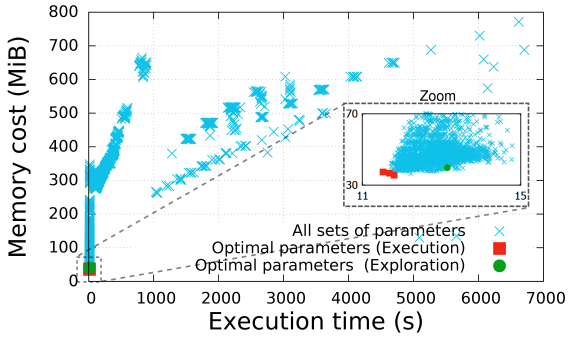
Figure 5: Boxplot of error rates representing differences between exploration results and concrete execution results, for each application and each calibrated theoretical analysis (complexity and memory cost).



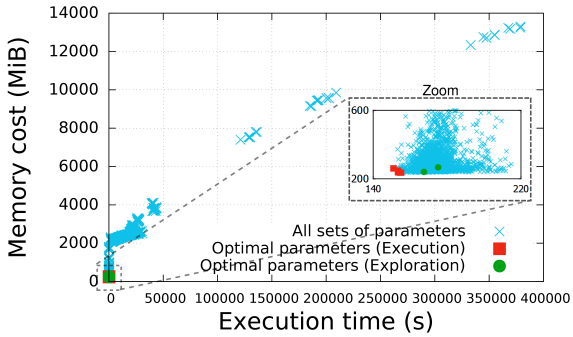
(a) *FiveHB*.



(b) *Pedagogic*.



(c) *Croissant*.



(d) *Medical*.

Figure 6: Identification of returned exploration points. Blue crosses are analysis results that are obtained after concrete execution of sets of parameters. Red squares are optimal points and green circles are exploration selected points.