Decomposition of Large Look-Up Tables for Fast Homomorphic Evaluation

Sonia Belaïd[1], Nicolas Bon[1,2] and Matthieu Rivain[1]

[1] CryptoExperts, Paris, France, firstname.lastname@cryptoexperts.com
[2] DIENS, Ecole normale supérieure, PSL University, CNRS, Inria, Paris, France, firstname.lastname@ens.fr

Abstract. TFHE is one of the most promising scheme in the literature for an adoption of Fully Homomorphic Encryption (FHE) in practice. The core reason of its good performances is the powerful Programmable Bootstrapping (PBS) operation, that enables to homomorphically evaluate a Look-Up Table (LUT) on a ciphertext while simultaneously reducing its noise. However, the computational cost of running a PBS degrades severely when the size of the plaintext space increases, making it intractable for precision larger than 8 bits. So, evaluating a LUT larger than 2^8 is not considered possible with the "vanilla" TFHE scheme. In this paper, we propose a technique to accelerate LUT evaluation at high precision, that significantly enhances the state of the art. Our method beats the original PBS for spaces larger than 6 bits, and is competitive with the WoP-PBS (the reference of the state of art) while being conceptually simpler. Additionally, our method relies only on the standard PBS of TFHE, and therefore does not require the design of new advanced homomorphic operators, facilitating its integration into larger homomorphic compilation systems.

Keywords: FHE · LUT · TFHE · Efficiency

1 Introduction

Fully Homomorphic Encryption (FHE) is a cryptographic technique that enables computations to be performed on encrypted data, without requiring decryption. This field has seen an impressive development over the past decade, notably with the emergence of increasingly efficient homomorphic schemes. To ensure security, these schemes all introduce random noise into ciphertexts, which increases as homomorphic operations are applied. This noise growth restricts the achievable computational depth, since excessive noise eventually corrupts the ciphertext and prevents correct decryption. The major breakthrough that overcame this limitation was a generic operation called *bootstrapping* [Gen09], which makes it possible to homomorphically reduce ciphertext noise, enabling further computations.

Among the most promising constructions in the literature is the TFHE scheme [CGGI16, CGGI17, CGGI20]. It supports simple linear homomorphisms (*i.e.,* additions of ciphertexts and multiplications by cleartext constants), but its most distinctive feature lies in an operation known as *Programmable Bootstrapping* (PBS). This operation not only refreshes ciphertexts by reducing noise, but also enables the homomorphic evaluation of an arbitrary Look-Up Table (LUT) on the encrypted input, making it possible, in theory, to evaluate any computational circuit. Despite the great flexibility of TFHE, the cost of PBS grows rapidly with the size of the plaintext space. Consequently, the scheme is considered impractical for plaintexts larger than a few bits (typically beyond 8). Besides, it is hampered by the so-called *negacyclicity problem* which restricts the set of evaluable functions, unless some countermeasure is implemented to relax this constraint. Various examples of those have

been proposed, particularly the ones of [YXS+21, KS23, CBSZ23, HLMS25], and they all induce some computational cost.

A typical use case for evaluating large LUTs is the homomorphic evaluation of activation functions in neural networks. Since these functions are intrinsically non-linear, they cannot be computed using only simple linear homomorphisms. A lot of literature on low-precision (thus TFHE-friendly) machine learning exists, notably for 16-bit precision [KWW+17, DMM+18] and 8-bit precision [SCC+19, CBG+20].

Another important use-case is transciphering, a technique designed to mitigate the ciphertext expansion introduced by homomorphic encryption. The idea is to encrypt the data using a symmetric cipher $\mathcal{C}$, and then recover homomorphic ciphertexts by evaluating the decryption function of $\mathcal{C}$ in the encrypted domain. A notable line of work is the evaluation of AES, that has seen impressive improvements in the recent years [TCBS23, WWL+23, WLW+24, BBB+25]. One of the key of the security of block ciphers (including AES) is non-linearity, and the the standard way to compute them homomorphically with TFHE is to represent them as a LUT and to evaluate it with a PBS. Surveys on the topic of homomorphic evaluation of block ciphers [TBS25, NWH+25] show that the input size of S-boxes used in common block ciphers is a major bottleneck for TFHE-based implementations. Consequently, improving the efficiency of LUT evaluation for these sizes would have a direct impact on the performance of homomorphic block cipher implementations and transciphering.

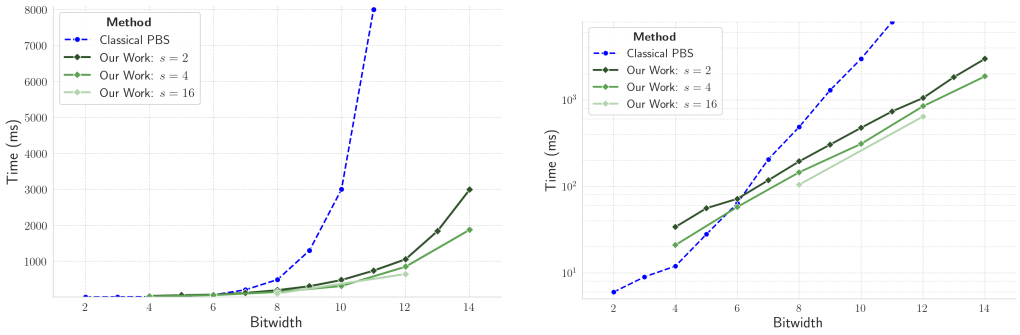
More generally, the ability to efficiently handle large LUT operations constitutes a powerful primitive for the generic compilation of homomorphic programs, as it enables a wider range of non-linear functions to be implemented with minimal circuit depth and without the need for complex gate-level optimizations.

Contributions. In this paper, we introduce an alternative method for efficiently evaluating large LUTs. Our approach is inspired by the masking literature, particularly from the line of work [CRV14, GR16, GRVV17]. These papers focus on generating circuit representations of arbitrary S-boxes to construct efficient masked implementations, under the constraint of minimizing either the number of multiplications or the use of specific simple functions. We adopt similar principles to design a new framework for evaluating large LUTs within TFHE, this time aiming to minimize the number of required PBS calls.

Previous methods to evaluate larger LUTs such as the Tree-Based Method (TBM) [GBA21] or the WoP-PBS [BBB+23] are built on top of advanced homomorphic operators such as sophisticated keyswitches or circuit bootstrapping, that requires larger key material or significant implementation complexity. On the contrary, our method only uses the original PBS of TFHE as a black box primitive, which makes the construction conceptually simpler and easier to integrate. As illustrated in Figure 1, our method already outperforms the vanilla PBS algorithm for LUTs of size 6-to-6 bits, and we extend our experiments up to 14-to-14 bits LUTs. Compared to WoP-PBS, our technique achieves comparable or even better performance for intermediate LUT sizes (between 6 and 12 bits). In particular, for LUTs of 8-to-8 bits, which correspond to the size of the AES S-box, our method surpasses all existing approaches.

Finally, we provide an open-source implementation of our framework, making it directly available for evaluation and integration to any library featuring standard TFHE operators.

Organization. Section 2 introduces the necessary background and notations, together with a brief overview of TFHE. In Section 3, we first formalize the problem of evaluating large LUTs. We then review existing evaluation techniques and their limitations, and explain how decomposition methods from the masking literature motivate our approach under a different cost model. Section 3 also introduces the two main building blocks used in our new decomposition approach: embeddings into small prime fields and the atomic



(a) Linear scale (b) Logarithmic scale

Figure 1: Comparison of the performance of the classical PBS and our method. See Section 6 for details on the experimental setup and comparison to previous works. The reported results correspond to LUTs with identical input and output sizes.

functions that we use to generate auxiliary non-linear values. Section 4 presents our generic LUT decomposition formula, analyzes its homomorphic complexity as a function of its parameters, and describes how to efficiently compute the unknown coefficients. Section 5 introduces further optimizations, with a particular focus on the multi-output setting, where computing the m outputs can be significantly more efficient than running the method m times independently. Finally, Section 6 reports experimental results and compares our approach with vanilla PBS, WoP-PBS, and the Tree-Based method.

2 Preliminaries

2.1 Notations

In this paper, scalars are denoted by lowercase letters (e.g. a). Vectors are written in bold (e.g. $\mathbf{a}$) and matrices in round uppercase type $\mathcal{A}$. The ring of integers modulo q, denoted classically by $\mathbb{Z}/q\mathbb{Z}$, is abbreviated as $\mathbb{Z}_q$. Similarly, finite field are denoted by $\mathbb{F}$, with $\mathbb{F}_p$ representing the finite field of prime order p. The operation of rounding to the closest integer is denoted by $\lfloor \cdot \rceil$, the operation of reduction modulo p is denoted $[\cdot]_p$ and $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes the inner product of vectors $\mathbf{x}$ and $\mathbf{y}$. Finally, for randomness and distributions, if $\mathcal{D}$ is a distribution, the notation $x \xleftarrow{\$} \mathcal{D}$ means that x is sampled according to $\mathcal{D}$. The same notation is used for uniform sampling from a set, for example $x \xleftarrow{\$} \mathbb{Z}_q$ denotes that x is sampling uniformly at random from $\mathbb{Z}_q$.

2.2 Background on TFHE

TFHE [CGGI16, CGGI17, CGGI20] is a homomorphic encryption scheme, designed as a successor to FHEW [DM15]. Its security is based on the Learning With Errors (LWE) problem. Optimized for operations on low-precision data (typically less than 6 bits), TFHE offers a distinctive feature: a *programmable* bootstrapping. This enables the evaluation of any univariate function on a ciphertext while simultaneously resetting its noise to a nominal level, which enable the fully homomorphic property. In what follows, we introduce TFHE, describe its encoding and encryption procedures, and provide an overview of the homomorphic operations it supports.

2.2.1 Plaintext Space and Encryption

The plaintext space is the *discretized torus* $\mathbb{T}_p$, that we identify to the ring $\mathbb{Z}_p$, with $p \in \mathbb{N}$. Let us consider a mapping $\rho : \mathbb{Z}_p \to \mathbb{Z}_q$, defined as $\rho : m \mapsto \left\lfloor \frac{mq}{p} \right\rceil$. The image of this mapping only reaches p elements in $\mathbb{Z}_q$, which take the form $\left\{ \frac{kq}{p} \mid k \in \mathbb{Z}_p \right\}$. These elements are evenly distributed across $\mathbb{Z}_q$ and form what we refer to as *sectors of* $\mathbb{Z}_q$, represented as: $\left\{ \left(\frac{(2k-1)q}{2p}, \frac{(2k+1)q}{2p} \right) \mid k \in \mathbb{Z}_p \right\}$, of which they are the center.

TFHE features two types of encryption that share similar structural patterns but operate within different mathematical spaces: LWE encryption and GLWE encryption. In this paper, we only use the former that we introduce below:

LWE Encryption. Let $m \in \mathbb{Z}_p$ be a message and let $\mathbf{s} = (s_0, \ldots, s_{n-1})$ represents the secret key, sampled uniformly at random from $\mathbb{B}^n$. Let χ_σ be a centered Gaussian distribution of standard deviation σ. First, the message m is encoded in the space $\mathbb{Z}_q$ by $\tilde{m} = \rho(m)$. A small random Gaussian noise $e \xleftarrow{\$} \chi_\sigma$ is then added. Since e is small, the noisy message $\tilde{m} + e$ remains within the same sector as $\tilde{m}$. Next, we construct the LWE ciphertext as a vector $\mathbf{c} = (a_0, \ldots, a_{n-1}, b)$, where the a_i's are sampled uniformly at random from $\mathbb{Z}_q$, and b is defined by $b = \langle \mathbf{a}, \mathbf{s} \rangle + \tilde{m} + e$.

Decryption has two steps: first, we compute $\phi(c) = b - \langle \mathbf{a}, \mathbf{s} \rangle$, referred to as the *phase*. Then we round it to the nearest plaintext value: $\tilde{m} = \left\lfloor \frac{p}{q} \phi(c) \right\rceil$. As long as $|e| < \frac{q}{2p}$, this rounding produces the right sector center.

The security of this problem relies on the assumption of hardness of the *Learning With Errors* problem, introduced in [Reg05], and widely used in the area of Post-Quantum Cryptography.

2.2.2 Homomorphic Operations

In what follows, we refer to the variance of the Gaussian random variable associated to the noise as the *noise variance*. This quantity is a measure of the "noise level" and should stay as low as possible.

Sum of ciphertexts. Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be two ciphertexts encrypting the messages m_1 and m_2, respectively, with noise variances σ_1^2 and σ_2^2. Performing a coordinate-wise sum of the two vectors results in a valid ciphertext c', which encrypts $m_1 + m_2$ with noise variance $\sigma_1^2 + \sigma_2^2$. We denote this operation by $\texttt{SumTFHE}(\mathbf{c}_1, \mathbf{c}_2)$.

Product with a cleartext. Let $\mathbf{c}$ be a ciphertext encrypting m with noise variance σ^2. Multiplying each coordinate of $\mathbf{c}$ by a constant $\lambda \in \mathbb{Z}_p$ produces a valid ciphertext $\mathbf{c}'$, which encrypts $m' = \lambda \cdot m$ with noise variance $\lambda^2 \cdot \sigma^2$. We denote this operation as $\texttt{ClearMultTFHE}(\mathbf{c}', \lambda)$.

These linear operations are extremely fast, particularly in comparison to bootstrapping (which we will introduce below). However, they increase the noise variance (noise level), which means that only a limited number of such operations can be performed before the correctness of the results is compromised.

Programmable Bootstrapping (PBS). Bootstrapping, introduced by Gentry in [Gen09], is a generic technique that allows the noise of a ciphertext to be homomorphically reset to a nominal level. While this operation can theoretically be applied to any homomorphic encryption scheme, it is often deemed too slow for practical use. However, in TFHE, bootstrapping is relatively efficient compared to other homomorphic schemes, especially for

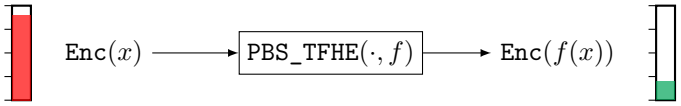


Figure 2: High-level illustration of a PBS. The gauges gives an informal view of the noise level in the ciphertexts.

low-precision messages, and it is implemented in a *programmable* manner. This means that while the noise is being reset, any arbitrary function can be evaluated on the ciphertext. We denote the evaluation of a PBS on a ciphertext $\mathbf{c}$ which evaluates the function f as $\texttt{PBS\_TFHE}(\mathbf{c}, f)$.

The techniques we introduce in this paper use the PBS as a black-box, so explaining its inner workings would be out of scope. Figure 2 shows a schematic model of how to think about the PBS.

3 Problem Formalization, Related Works, and Building Blocks

In this section, we formalize the problem of evaluating large look-up tables (LUTs) in the TFHE framework, review the main existing approaches and their limitations, and introduce the key decomposition ideas that inspire our construction. We also present the building blocks used throughout the rest of the paper.

3.1 Formalization of the Problem

We consider a look-up table over $\mathbb{Z}_s$ for some integer $s \geq 2$. Let $n, m \in \mathbb{N}$ denote the input and output dimensions (i.e., the number of radix-s digits) respectively. The target LUT is modeled as a function

$$F : \mathbb{Z}_s^n \to \mathbb{Z}_s^m \ ,$$

whose input space has cardinality s^n, which quickly becomes prohibitive for direct LUT evaluation techniques (e.g., for $s^n > 2^8$).

In TFHE, the native LUT evaluation primitive is the PBS, which applies an univariate LUT to a single element of $\mathbb{Z}_s$. A standard way to handle multiple inputs is to pack them into a single value over a larger space. For instance, for $(x_1, x_2) \in \mathbb{Z}_s^2$, one can work over $\mathbb{Z}_{s^2}$ and encode the pair as

$$x_1 \cdot s + x_2 \in \mathbb{Z}_{s^2}.$$

More generally, n input elements can be packed into $\mathbb{Z}_{s^n}$. While conceptually simple, this approach severely impacts performance: the cost of PBS grows rapidly with the plaintext modulus, making direct evaluation over $\mathbb{Z}_{s^n}$ impractical for moderately large s^n. Hence, performance is essentially driven by s^n, and one must either keep s small or limit the function arity n.

3.2 Existing Methods and Limitations

The above observation has motivated several lines of work to evaluate large LUTs in TFHE more efficiently. We briefly review the main approaches and highlight their limitations under the TFHE cost model, where PBS dominate runtime.

Methods from the TFHE literature One of the most well-known constructions to improve the efficiency of large LUT in TFHE is the Tree-Based Method (TBM). Introduced in [GBA21], the idea is to split the input message into d ciphertexts. To evaluate a function f with a large input size B^d, the technique uses a tree structure of d layers of PBS, which are carried on $\mathbb{Z}_B$ (and thus are much faster than on $\mathbb{Z}_{B^d}$). The i-th layer of the tree requires B^{d-i} bootstrappings. This method becomes very efficient when combined with the Multi-Value Bootstrapping (MVB) of [CIM19] that allows to parallelize the first layer of the tree, which is by far the most costly. Notably, this construction is studied in details for 8-bit LUT (targeting an homomorphic implementation of AES) in [TCBS23].

More recently, the work [BBB+23] introduces a construction called WoP-PBS (Without-Padding Programmable Bootstrapping), which leverages a technique known as circuit bootstrapping. This process transforms a LWE ciphertext into a GGSW one via ℓ_{GGSW} PBS[1]. The core idea involves extracting each bit of the message and encoding them into individual GGSW ciphertexts. They are then used as selectors in a `CMUX` tree that selects an appropriate accumulator polynomial based on the extracted bits. Finally, a classical PBS is used to rotate the resulting polynomial using the remaining bits of the message and select the appropriate output. WoP-PBS enjoys particularly good asymptotic performances, however the requirement of ℓ_{GGSW} PBS operations per circuit bootstrapping becomes prohibitive at low precisions. WoP-PBS surpasses classical PBS for precisions of 8 bits and above, and have been shown able to evaluate functions over plaintext spaces up to 24 bits, which proves an impressive scalability.

Another interesting work is [NOFN24], which uses an original approach leveraging Time-Memory trade-offs to reduce the amount of computations. While the method shows very good performances in leveled mode (meaning without boostrapping and thus depth-bounded), it does not compete with the rest of the state of the art when it comes to fully-homomorphic mode.

Methods from the masking literature Beyond the TFHE literature, relevant ideas have also been developed in the context of masking, a widely deployed countermeasure against side-channel attacks. Although motivated by different considerations, masking addresses an algorithmic challenge closely related to ours: efficiently evaluating non-linear LUTs by decomposing them into cheaper building blocks. In additive (Boolean) masking, each sensitive variable is replaced by an n-share representation, where the shares are sampled uniformly at random, subject to the constraint that their sum (or XOR, depending on the masking variant) reconstructs the original value. Linear operations can then be applied share-wise at essentially linear cost, whereas non-linear operations require dedicated multiplication gadgets whose complexity typically scales quadratically in the number of shares. This makes multiplications the dominant bottleneck, and has motivated a line of works to efficiently decompose non-linear LUTs using as few multiplications as possible (or more generally, low-algebraic components) [CRV14, CPRR15, GR16, GRVV17].

Their approach typically involves a generic decomposition formula. Some terms of the formula are sampled at random, and then the remaining components are determined by solving a corresponding linear system of equations. In its basic form (see [CRV14]), the decomposition of a univariate function f is expressed as

$$f(x) = \sum_{i=0}^{t-1} h_i(x) \cdot g_i(x) + h_t(x) \ ,$$

where the $\{h_i\}$ and $\{g_i\}$ functions are defined as linear combinations of functions within a precomputed basis (derived with a few multiplications). The g_i functions are chosen

[1] Here, ℓ_{GGSW} refers to the level of the gadget decompositions used in GGSW format, see [CGGI16] for more details.

randomly, while the h_i functions are determined by solving a linear system of equations to ensure the equality holds for all possible inputs. The parameter t is selected to minimize the number of multiplications, which is upper-bounded by t.

While these techniques were designed to minimize multiplications in masked implementations, they provide a useful blueprint for our setting: inject the required amount of non-linearity at low cost, randomize part of the decomposition to gain flexibility, and then complete the construction by solving a linear system. However, directly reusing them in TFHE is not sufficient, since our cost model and constraints differ significantly: PBS calls are the dominant cost, and the computation must remain over small prime fields to leverage efficient native PBS. This motivates a different approach.

In the following, our goal is to to decompose a target function F into a circuit of smaller LUTs over $\mathbb{Z}_s$, each of which can be evaluated efficiently using TFHE's native PBS, while keeping the overall number of PBS calls low. To achieve this, we adapt the above strategy from the masking literature to the TFHE setting: unlike masking, we optimize for PBS calls rather than multiplications, and we design the decomposition to operate over small prime fields (e.g., $\mathbb{F}_3$, $\mathbb{F}_5$, $\mathbb{F}_{17}$), where PBS is particularly efficient in practice.

3.3 Building Blocks

Before presenting our decomposition strategy, we introduce two technical building blocks that will be used throughout the rest of the paper. The first one is an embedding of $\mathbb{Z}_s$ into a small prime field $\mathbb{F}_p$, which allows us to leverage the properties of PBS in prime fields. The second one is the notion of *atomic functions*, which provides a convenient way to generate auxiliary non-linear variables from the inputs at the cost of a single PBS call. These ingredients will be combined in the next sections to construct and analyze our method.

3.3.1 Embedding in a Prime Field

Our method requires working within a finite field, as a consequence the elements of $\mathbb{Z}_s$ must first be embedded into such a field. When s is not prime, we rely on the encoding technique introduced in [BPR24] and later generalized to the arithmetic setting in [BBB+25], known as the (s, p)-encoding. In essence, this encoding defines an embedding of the ring $\mathbb{Z}_s$ into a larger space $\mathbb{Z}_p$, with $p > s$. We select p as the smallest prime number greater than s, so that this new space can naturally be identified with the prime field $\mathbb{F}_p$. We then define an injective mapping $\mathcal{E} : \mathbb{Z}_s \mapsto \mathbb{F}_p$. Apart from the choice of p, which is crucial for the correctness and performance of our method, the definition of $\mathcal{E}$ itself can be chosen arbitrarily. For simplicity, we adopt the natural "identity" embedding:

$$\begin{aligned} \mathcal{E} : \mathbb{Z}_s &\to \mathbb{F}_p \\ x &\mapsto x \end{aligned}$$

As $s < p$, this always defines a valid injective map. In the rest of the paper, we will use the same notation for the original value living in $\mathbb{Z}_s$ and its embedding in $\mathbb{F}_p$. Moreover, the function F that we aim to evaluate is replaced by its embedded counterpart over $\mathbb{F}_p$:

$$\begin{aligned} \bar{F} : \mathrm{Im}(\mathcal{E})^n &\to (\mathbb{F}_p)^m \\ \mathbf{x} = (x_0, \ldots, x_{n-1}) &\mapsto \mathcal{E}(F(\mathcal{E}^{-1}(x_0), \ldots, \mathcal{E}^{-1}(x_{n-1}))) \end{aligned}$$

where $\mathcal{E}^{-1} : \mathrm{Im}(\mathcal{E}) \mapsto \mathbb{Z}_s$ refers to the function mapping the elements of $\mathbb{F}_p$ to the original value of $\mathbb{Z}_s$ they encode.

Hence, instead of encrypting a value $x \in \mathbb{Z}_s$ directly, we encrypt its image under the embedding, $\mathcal{E}(x) \in \mathbb{F}_p$. Conversely, the decryption function produces elements in $\mathbb{F}_p$, which

must then be mapped back to the original plaintext space $\mathbb{Z}_s$ using $\mathcal{E}^{-1}$. Note that this construction additionally solves the negacyclicity problem, as it can be easily circumvented with odd plaintext modulus as shown in [BPR24].

3.3.2 Construction of Atomic Functions

Our method requires constructing a pool of variables derived from the inputs $x_0, \ldots, x_{n-1}$ of the function f. For our method to work, these variables must be *linearly independent* from the x_i's, in the sense that none of them can be expressed as a linear combination of the x_i's. To achieve that, we define the notion of an *atomic function of arity* r as follows[2]:

Definition 1. Let $\mathfrak{F}_p$ be the set of all functions from $\mathbb{F}_p$ to $\mathbb{F}_p$, and r be an integer. An atomic function of arity r is a function ϕ of the form

$$\begin{aligned} \phi : \mathbb{F}_p^r &\to \mathbb{F}_p \\ (x_0, \ldots, x_{r-1}) &\mapsto \psi\left(\sum_{j=0}^{r-1} \alpha_j \cdot x_j\right) \end{aligned}$$

with $\psi \in \mathfrak{F}_p$ and $\boldsymbol{\alpha} = (\alpha_j)_{0 \le j < r} \in \mathbb{F}_p^r$.

Empirically, when we sample an atomic function at random such that ψ is non-linear (and if $\boldsymbol{\alpha}$ is non-zero), the output we obtain is linearly independent of the inputs. Such an atomic function can be evaluated homomorphically using the native operations of TFHE. The procedure is described in Algorithm 1.

Algorithm 1 `EvalAtom` - Homomorphic evaluation of an atomic function

Context:
- $\mathbb{F}_p$: the field of embedding.
- $(x_0, \ldots, x_{r-1}) \in \mathbb{F}_p^r$: the inputs of the atomic function in the clear.

Input:
- $\phi : \mathbb{F}_p^r \mapsto \mathbb{F}_p$: the atomic function of arity r we want to evaluate.
- $(\mathbf{c}_0, \ldots, \mathbf{c}_{r-1})$: the ciphertexts encrypting the inputs
- $\boldsymbol{\alpha} = (\alpha_j)_{0 \le j < r} \in \mathbb{F}_p^r$: the coefficients of the linear combination of ϕ
- $\psi : \mathbb{F}_p \mapsto \mathbb{F}_p$: the outer function of ϕ.

Result: $\mathbf{c}_r$: an encryption of $x_r = \phi(x_0, \ldots, x_{r-1})$

$\mathbf{c}_r \leftarrow 0$
`/* Evaluation of the linear combination */`
for $k \in \{0, \ldots, r-1\}$ **do**
  $\mathbf{c}_r \leftarrow$ `SumTFHE`$(\mathbf{c}_r,$ `ClearMultTFHE`$(\mathbf{c}_k, \alpha_k))$
end
`/* Evaluation of the final function` ψ `with a bootstrapping */`
$\mathbf{c}_r \leftarrow$ `PBS_TFHE`$(\mathbf{c}_r, \psi)$
return $\mathbf{c}_r$

A few remarks on this straightforward algorithm are in order:

- The algorithmic cost of `EvalAtom` can be boiled down to the cost of the final PBS, as the other operations are linear so essentially free.
- The noise in the result is reset to a nominal level by the final PBS.
- The noise growth in the linear combination (before the PBS) is exactly quantified by the norm of $\boldsymbol{\alpha}$. Trivially, the noise variance is multiplied by $\|\boldsymbol{\alpha}\|^2$.

[2] This terminology is inspired by the notion of *atomic pattern* from [BBB+23].

Now, we define a notion of *chain of atomic functions*:

Definition 2. Let $n, \lambda \in \mathbb{N}$. A chain of atomic functions $\Phi_{n,\lambda}$ of basis n and length λ is a function defined as follows:

$$\begin{aligned} \Phi_{n,\lambda} : \mathbb{F}_p^n &\to \mathbb{F}_p^\lambda \\ (x_0, \ldots, x_{n-1}) &\mapsto (x_n, \ldots, x_{n+\lambda-1}) \end{aligned}$$

with

$$\forall k \in \{0, \ldots, \lambda - 1\}, \quad x_{n+k} = \phi_k(x_0, \ldots, x_{n+k-1}) := \psi_k \left(\sum_{j=0}^{n+k-1} \alpha_{k,j} \cdot x_j \right)$$

where ϕ_k is an atomic function of arity $n + k$, i.e., such that $\psi_k \in \mathfrak{F}_p$ and $\boldsymbol{\alpha}_k \in \mathbb{F}_p^{n+k}$.

Concretely, the k-th element of the chain is an atomic function ϕ_k taking in input:

- the n inputs of the chain, and
- the $k - 1$ outputs produced by the previous elements of the chain.

Homomorphically evaluating a chain is done by simply calling `EvalAtom` (Algorithm 1) on each link in the chain.

4 The LUT Decomposition Method

In this section, we present our LUT decomposition method. Our goal is to evaluate a target LUT $F : \mathbb{Z}_s^n \to \mathbb{Z}_s^m$ in TFHE with a small number of PBS calls. Rather than performing a single PBS over a large plaintext space, we aim to rewrite F as a structured expression built from: (i) a limited number of auxiliary non-linear values generated by atomic functions (each involving a single small PBS), (ii) a small number of products (each involving two small PBSs), and (iii) linear combinations (which can be homomorphically evaluated essentially for free). Following the decomposition approach from the masking literature that is overviewed in Subsection 3.2, part of the decomposition is sampled at random so that the full decomposition can be obtained efficiently by solving a linear system.

For clarity, we first describe the method in the single-output setting, and focus on one output coordinate of F, denoted by $f : \mathbb{Z}_s^n \to \mathbb{Z}_s$. The procedure then extends to the multi-output case either by applying the method independently to each output, or by using the dedicated optimization described in Section 5.

4.1 Decomposition Equation

Let $\Phi_{n,\lambda}$ be a chain of λ atomic functions (Definition 2). Given an input $\mathbf{x} = (x_0, \ldots, x_{n-1}) \in \mathbb{Z}_s^n$, we extend it with λ auxiliary variables:

$$(x_n, \ldots, x_{L-1}) := \Phi_{n,\lambda}(x_0, \ldots, x_{n-1}), \qquad \text{where } L = n + \lambda.$$

Our method seeks parameters $\lambda, t \in \mathbb{N}$ and coefficients in $\mathbb{F}_p$ such that the embedded function $\bar{f}$ (defined in Section 3.3.1) can be expressed as

$$\forall \mathbf{x} \in \mathbb{Z}_s^n, \qquad \bar{f}(\mathbf{x}) = \sum_{i=0}^{t-1} \Big(\sum_{j=0}^{L-1} \beta_{i,j} \cdot x_j \Big) \cdot \Big(\sum_{k=0}^{L-1} d_{i,k} \cdot x_k \Big) + \sum_{j=0}^{L-1} \beta_{t,j} \cdot x_j, \tag{1}$$

where $(\beta_{i,j}) \in \mathbb{F}_p$ for $0 \le i \le t$ and $0 \le j < L$, and $(d_{i,k}) \in \mathbb{F}_p$ for $0 \le i < t$ and $0 \le k < L$.

Finding a decomposition. As we will explain in Section 4.3, finding a decomposition for a given LUT reduces to solving a linear system once the parameters (λ, t) and the coefficients $(d_{i,k})$ are fixed. In our procedure, the coefficients $(d_{i,k})$ are sampled uniformly at random, and the coefficients $(\beta_{i,j})$ are then determined so that Equation 1 holds for all inputs. In practice, the main algorithmic task is therefore to choose (λ, t) so as to minimize the homomorphic evaluation cost, and to efficiently compute the corresponding β-coefficients when a solution exists.

4.2 Homomorphic Evaluation of a Decomposition

To homomorphically evaluate a decomposition of $\bar{f}$ as defined in Equation 1, the following FHE operations are required:

- **Generation of intermediate variables.** All variables x_i are obtained by homomorphically evaluating the chain $\Phi_{n,\lambda}$, that is, through successive calls to `EvalAtom` (Algorithm 1).
- **Linear operations.** Since every term of the formula belongs to $\mathbb{F}_p$, plaintext-ciphertext products (*i.e.,* the products $\beta_{ij} \cdot x_j$ and $d_{i,j} \cdot x_j$) are efficiently performed using the native `ClearMultTFHE` operation. Additions are straightforward as well and are computed with `SumTFHE`.
- **Ciphertext-ciphertext multiplications.** Multiplying two ciphertexts is trickier. A direct PBS with two inputs could be used, but this would lead to poor performances. Instead, we adopt the classical trick which leverages the following property:

$$x \cdot y = \left[4^{-1}\right]_p \cdot ((x+y)^2 - (x-y)^2) \ . \tag{2}$$

  Both squaring operations can be evaluated with simple PBS. Evaluating two PBS on inputs of size p is significantly more efficient than evaluating a single PBS of size p^2 (as illustrated by Figure 1, doubling the bit-size of the input is devastating for the performance). The multiplication by $\left[4^{-1}\right]_p$ could be done with `ClearMultTFHE`, but this would slightly increase the noise. To avoid that, the multiplication constant is incorporated directly into the PBS lookup tables. This procedure is formalized in Algorithm 2 (`EncryptedProduct`).

Putting everything together, we obtain Algorithm 3, for the homomorphic evaluation of the decomposition. The cost of this algorithm can be estimated by counting the number of required PBS calls:

- λ PBS to evaluate the chain,
- $2t$ PBS to perform the t ciphertext-ciphertext multiplications using Equation 2.

The total cost in PBS is therefore $\lambda + 2t$.

4.3 Construction of Efficient Decompositions

We have introduced the decomposition structure and its homomorphic evaluation; we now aim to provide a method to find such a decomposition for a given LUT. Given concrete parameters λ and t (we will see later how to choose them optimally), the decomposition method consists in first sampling random values for the $d_{i,k}$'s, and then computing the corresponding values for the $\beta_{i,j}$'s. The goal is to ensure that the decomposition matches the target function f for all possible inputs.

Algorithm 2 `EncryptedProduct` - Homomorphic evaluation of a ciphertext-ciphertext product

Context: $\begin{cases}\mathbb{F}_p\text{: the field of embedding.}\\ x_1, x_2 \in \mathbb{F}_p \text{ : the inputs of the product in the clear.}\end{cases}$
Input: $\mathbf{c}_1, \mathbf{c}_2$: the ciphertexts encrypting the inputs x_1 and x_2.
Result: $\mathbf{c}'$: an encryption of $x_1 \cdot x_2$.

```
/* Computation of the sum and the difference                          */
```
$\boldsymbol{\sigma}_1 \leftarrow \texttt{SumTFHE}(\mathbf{c}_1, \mathbf{c}_2)$
$\boldsymbol{\sigma}_2 \leftarrow \texttt{SumTFHE}(\mathbf{c}_1, \texttt{MulTFHE}(\mathbf{c}_2, [-1]_p))$

```
/* Computation of the squares (the minus sign of the second term is
   precomputed)                                                       */
```
$\boldsymbol{\sigma}'_1 \leftarrow \texttt{PBS\_TFHE}(\boldsymbol{\sigma}_1, x \mapsto x^2 \cdot [4^{-1}]_p)$
$\boldsymbol{\sigma}'_2 \leftarrow \texttt{PBS\_TFHE}(\boldsymbol{\sigma}_2, x \mapsto x^2 \cdot [-4^{-1}]_p)$

```
/* Sum of the squares                                                 */
```
$\mathbf{c}_r \leftarrow \texttt{SumTFHE}(\boldsymbol{\sigma}'_1, \boldsymbol{\sigma}'_2)$

return $\mathbf{c}_r$

To formalize the notation used in the rest of this section, we denote by $\mathbf{x}^{(i)}$ the i-th[3] element of $\mathbb{Z}_s^n$. Formally,

$$\mathbb{Z}_s^n = \left\{\mathbf{x}^{(i)} = (x_0^{(i)}, \ldots, x_{n-1}^{(i)}) \mid 0 \leq i < s^n\right\} .$$

4.3.1 Construction of a Valid Decomposition

We now rewrite the decomposition of our single-output function from Equation 1 in a matrix form, in order to explain the solving method, the constraints it imposes, and its resulting performance.

Matrix representation. More precisely, Equation 1 can be expressed as a matrix-vector equation, where each row of the matrix corresponds to a distinct input of the chain:

$$\mathbf{y} = \mathcal{A} \cdot \boldsymbol{\beta} \tag{3}$$

where:

- $\mathbf{y} = (y_0, \ldots, y_{s^n-1})^\top \in \mathbb{F}_p^{s^n}$ is a vector containing the s^n outputs of the function $\bar{f}$, corresponding to all s^n possible inputs:
$$\forall i \in \mathbb{Z}_{s^n}, y_i = \bar{f}(x_0^{(i)}, \ldots, x_{n-1}^{(i)})$$
- $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_{(t+1)L-1})^\top \in \mathbb{F}_p^{(t+1)L}$ is a flattened container for the β coefficients, such that:
$$\forall i, j \in \mathbb{Z}_{t+1} \times \mathbb{Z}_L, \beta_{iL+j} = \beta_{i,j} .$$
- $\mathcal{A} \in \mathbb{F}_p^{s^n \times tL}$ is the matrix defined by block as:

[3] The order does not matter, so we can arbitrarily pick the lexicographical order for example.

Algorithm 3 Homomorphic evaluation of a decomposition

Context: $\begin{cases} \bar{f} : \mathbb{F}_p^n \mapsto \mathbb{F}_p : \text{the function to evaluate.} \\ (x_0, \ldots, x_{n-1}) \in \mathbb{F}_p^n : \text{the clear inputs of the function.} \\ \Phi_{n,\lambda} = (\phi_0, \ldots, \phi_{\lambda-1}) : \text{the chain used to generate the decomposition.} \\ L = n + \lambda. \end{cases}$

Input: $\begin{cases} (\mathbf{c}_0, \ldots, \mathbf{c}_{n-1}) : \text{the ciphertexts encrypting the inputs.} \\ (\beta_{i,j})_{\substack{0 \le i \le t \\ 0 \le j < L}} \in \mathbb{F}_p^{(t+1) \times L} : \text{coefficients of the left-hand linear combination (and the linear term) of Equat} \\ (d_{i,j})_{\substack{0 \le i < t \\ 0 \le j < L}} \in \mathbb{F}_p^{t \times L} : \text{coefficients of the right-hand linear combination of Equation 1.} \end{cases}$

Result: $\mathbf{c}_y$: an encryption of $y = f(x_0, \ldots, x_{n-1})$

```
/* Evaluation of the chain of atomic functions                          */
for k ∈ {n, ..., L − 1} do
    c_k ← EvalAtom(φ_k, (c_0, ..., c_{k−1}))
end
c_y ← 0
for i ∈ {0, ..., t − 1} do
    /* Computation of the linear combinations with the β's and d's      */
    for j ∈ {0, ..., L − 1} do
        γ_ij ← ClearMultTFHE(c_j, β_ij)
        γ'_ij ← ClearMultTFHE(c_j, d_ij)
    end
    σ_i ← SumTFHE(γ_i0, ..., γ_{i,L−1})
    σ'_i ← SumTFHE(γ'_i0, ..., γ'_{i,L−1})
    /* Computation of the encrypted product from Algorithm 2            */
    c_y ← SumTFHE(c_y, EncryptedProduct(σ_i, σ'_i))
end
/* Computation of the linear term                                       */
for j ∈ {0, ..., L − 1} do
    γ_tj ← ClearMultTFHE(c_j, β_tj)
end
c_y ← SumTFHE(c_y, γ_t,0, ..., γ_{t,L−1})
return c_y
```

$$\mathcal{A} = (\mathcal{A}_0 \mid \mathcal{A}_1 \mid \ \ldots \ \mid \mathcal{A}_{t-1} \mid \mathcal{A}_t) \quad \text{with:}$$

$$\forall 0 \le i < t, \mathcal{A}_i = \begin{pmatrix} x_0^{(0)} \cdot \langle \mathbf{d}_i, \mathbf{x}^{(0)} \rangle & \ldots & x_{L-1}^{(0)} \cdot \langle \mathbf{d}_i, \mathbf{x}^{(0)} \rangle \\ x_0^{(1)} \cdot \langle \mathbf{d}_i, \mathbf{x}^{(1)} \rangle & \ldots & x_{L-1}^{(1)} \cdot \langle \mathbf{d}_i, \mathbf{x}^{(1)} \rangle \\ \vdots & \vdots & \vdots \\ x_0^{(s^n-1)} \cdot \langle \mathbf{d}_i, \mathbf{x}^{(s^n-1)} \rangle & \ldots & x_{L-1}^{(s^n-1)} \cdot \langle \mathbf{d}_i, \mathbf{x}^{(s^n-1)} \rangle \end{pmatrix} \in \mathbb{F}_p^{s^n \times L}$$

$$\text{and: } \mathbf{d}_i = (d_{i,0}, \ldots, d_{i,L-1}) \in \mathbb{F}_p^L$$

$$\text{and: } \mathcal{A}_t = \begin{pmatrix} x_0^{(0)} & \ldots & x_{L-1}^{(0)} \\ x_0^{(1)} & \ldots & x_{L-1}^{(1)} \\ \vdots & \vdots & \vdots \\ x_0^{(s^n-1)} & \ldots & x_{L-1}^{(s^n-1)} \end{pmatrix} \in \mathbb{F}_p^{s^n \times L} \tag{4}$$

Putting everything together, we can rewrite Equation 3 as:

$$s^n \left\{ \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ y_{s^n-1} \end{pmatrix} \right. = \overbrace{\left(\begin{array}{c|c|c|c} & & & \\ \mathcal{A}_0 & \mathcal{A}_1 & \dots & \mathcal{A}_t \\ & & & \end{array}\right)}^{(t+1)\cdot L} \cdot \left(\begin{array}{c} \beta_{0,0} \\ \vdots \\ \hline \beta_{0,L-1} \\ \vdots \\ \vdots \\ \hline \beta_{t,0} \\ \vdots \\ \beta_{t,L-1} \end{array}\right) \tag{5}$$

To formalize the connection with the notion of a chain, we define the *matrix representation* of a chain:

Definition 3. Let $\Phi_{n,\lambda}$ a chain of atomic functions of basis n and length λ. The *matrix representation* of the chain $\Phi_{n,\lambda}$ with respect to $\mathbb{Z}_s$ is a matrix of size $s^n \times (n+\lambda)$ whose i-th row stores the concatenation of the inputs and the outputs of the evaluation of $\Phi_{n,\lambda}$ on the input $\mathbf{x}^{(i)} \in \mathbb{Z}_s^n$. Thus, it can be written:

$$\texttt{Mat}(\Phi_{n,\lambda}, \mathbb{Z}_s) = \begin{pmatrix} x_0^{(0)} & \dots & x_{n-1}^{(0)} & x_n^{(0)} & \dots & x_{n+\lambda-1}^{(0)} \\ x_0^{(1)} & \dots & x_{n-1}^{(1)} & x_n^{(1)} & \dots & x_{n+\lambda-1}^{(1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0^{(s^n-1)} & \dots & x_{n-1}^{(s^n-1)} & x_n^{(s^n-1)} & \dots & x_{n+\lambda-1}^{(s^n-1)} \end{pmatrix} \in \mathbb{F}_p^{s^n \times (n+\lambda)}$$

where $x_{n+k}^{(i)} = \phi_k(x_0^{(i)}, \dots, x_{n+k-1}^{(i)})$ for $0 \leq k < \lambda$ and $0 \leq i < s^n$. This matrix can be seen as the concatenation of two blocks:

$$\texttt{Mat}(\Phi_{n,\lambda}, \mathbb{Z}_s) = \left(\begin{array}{c|c} \texttt{Triv} & \texttt{Eval} \end{array} \right)$$

where `Triv` is the trivial writing of all possible inputs of the chain (so all the elements of $\mathbb{Z}_{s^n}$) and each row of `Eval` corresponds to evaluation of the chain with the corresponding row of `Triv` as input.

Using this notation, the matrix $\mathcal{A}$ can alternatively be defined as $\mathcal{A} = \mathcal{V} \,\square\, \mathcal{U}$, where:

- $\mathcal{U}$ is the matrix representation of the chain $\Phi_{n,\lambda}$ with respect to $\mathbb{Z}_s$.
- $\mathcal{V}$ is the matrix defined by:
$$\mathcal{V} = \left(\mathcal{U} \cdot \mathcal{D}^T \mid \mathbf{1}\right) \tag{6}$$
where $\mathcal{D}$ is the matrix containing all the d_{ij}'s:
$$\mathcal{D} = \begin{pmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_{t-1} \end{pmatrix} = \begin{pmatrix} d_{0,0} & \dots & d_{0,L-1} \\ d_{1,0} & \dots & d_{1,L-1} \\ \vdots & \ddots & \vdots \\ d_{t-1,0} & \dots & d_{t-1,L-1} \end{pmatrix} \in \mathbb{F}_p^{t \times L} \tag{7}$$
and $\mathbf{1}$ is a column vector filled with 1.
- $\square$ denotes the *face-splitting* product of Slyusar [Sly99], illustrated on Figure 3.

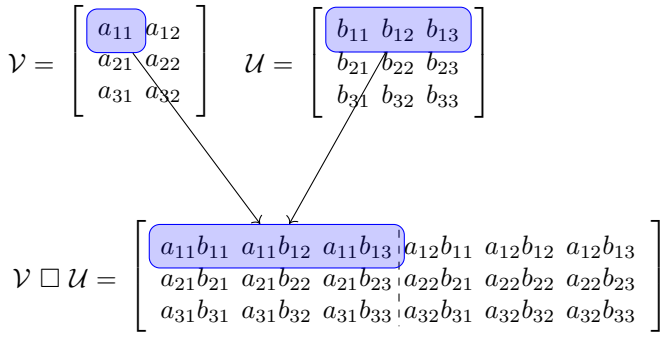


Figure 3: Illustration of the face-splitting product.

Now that we have described the decomposition of the function $\bar{f}$, we can move on to the explanation of the method to find a good decomposition.

Solving method. In practice, we sample uniformly at random a chain of atomic functions $\Phi_{n,\lambda}$ (with coefficients α drawn uniformly at random) and a matrix $\mathcal{D}$. The vector $\boldsymbol{\beta}$ is then treated as the unknown. To check whether the pair $(\Phi_{n,\lambda}, \mathcal{D})$ yields a valid decomposition of $\bar{f}$, we aim to find a vector $\boldsymbol{\beta}$ satisfying Equation 5. This is possible whenever $\texttt{Rank}(\mathcal{A}) = s^n$, *i.e.,* when $\mathcal{A}$ has full row rank, which allows us to obtain $\boldsymbol{\beta}$ with a Gaussian elimination. If the matrix is rank-deficient, we simply resample a new chain and matrix and repeat the procedure.

One of the strength of this method is that the pair $(\Phi_{n,\lambda}, \mathcal{D})$ and the underlying matrix $\mathcal{A}$ can be used for *any* function f. Indeed, the definition of f only impacts the target vector $\mathbf{y}$ in the system. So once a pair $(\Phi_{n,\lambda}, \mathcal{D})$ yielding a full row-rank matrix $\mathcal{A}$ has been found, it can be used for any function f of the same arity n.

Induced constraints. As shown above, obtaining a valid decomposition requires the matrix $\mathcal{A}$ to be *full row rank*. A necessary condition for this is that $\mathcal{A}$ must have at least as many columns as rows, which leads to the following simple inequality on the decomposition parameters:

$$s^n \leq (t+1) \cdot (n+\lambda). \tag{8}$$

Concretely, the method for finding a valid decomposition consists in sampling random chains $\Phi_{n,\lambda}$ and random matrices $\mathcal{D} \in \mathbb{F}_p^{t \times L}$ until $\mathcal{A}$ becomes full row rank. Since the columns of $\mathcal{A}$ are formed from the outputs of random functions, linear dependencies between them are highly unlikely. In practice, a suitable decomposition is typically found quickly whenever the parameters satisfy Equation 8.

Performances of the search algorithm. Our experiments reveal a significant variation in the time required to find a valid solution, depending on the value of n. Two phenomenons explain this variability:

- The Gaussian elimination algorithm has a cubic complexity in the number of rows of $\mathcal{A}$ (*i.e.,* in s^n). Consequently, as n increases, the computational cost of testing a single chain grows rapidly. Figure 4 reports the execution time of the algorithm for one chain. All the experiments were conducted on a server equipped with an AMD Ryzen Threadripper PRO 7995WX (96 cores).

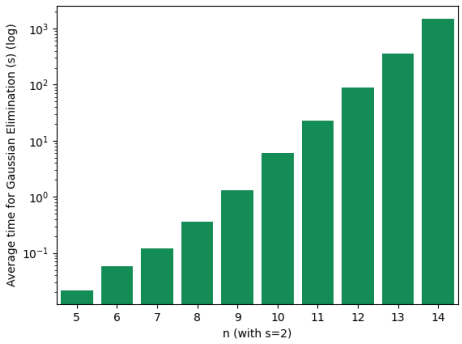


Figure 4: Average time of the Gaussian elimination with respect to n, for $s = 2$ and $p = 3$.

- The probability that a random chain (together with a random $\mathcal{D}$) yields a valid decomposition varies significantly, and seems to be inversely correlated with the tightness of the bound of Equation 8. A natural way to accelerate the search procedure is therefore to relax this bound by introducing a multiplicative factor $\gamma > 1$ on the left-hand side of Equation 8, providing a larger margin:

$$\gamma \cdot s^n \leq (t+1) \cdot (n+\lambda).$$

  Our experimental results show that setting $\gamma = 1.05$ is enough to obtain a solution in less than 200 iterations for any choice of parameters s, p and n. The impact of γ can be visualized on Figure 5. Increasing γ slightly increases the required values of λ and t, which in turn slows down a bit the homomorphic evaluation of the decomposition (see Table 2 for a quantitative illustration of this slowdown).

Our experiments further show that the probability of finding a valid decomposition for a given set of parameters depends on the margin in the bound of Equation 8, normalized by the number of rows of the matrix. More formally, we observe a correlation coefficient of 0.89 between the quantity

$$\frac{(t+1)(n+\lambda) - s^n}{s^n}$$

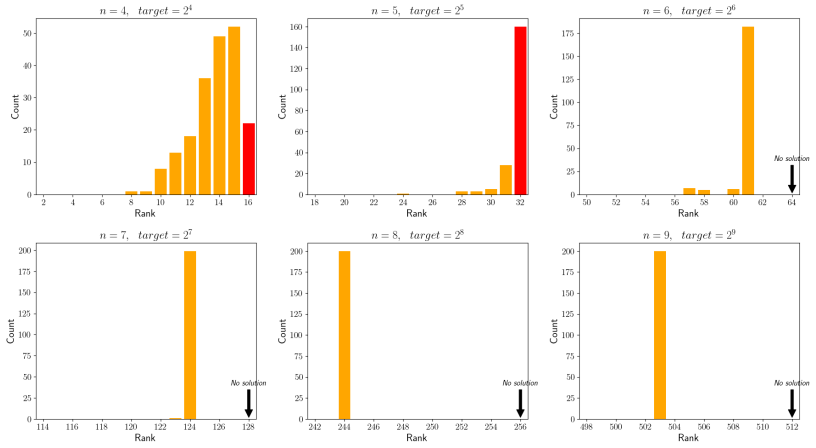
and the probability of obtaining a full-rank matrix with these parameters. This correlation was measured by generating 10,000 matrices of various sizes, sorting them by margin, dividing them into equal-width bins, and computing the empirical probability of full-rankness within each bin. The resulting correlation is illustrated in Figure 6.

Overall, the cost of finding one valid decomposition remains reasonable for values of n up to 14, which we chose as an arbitrary upper bound for the arity of the functions used in our experiments (Section 6).

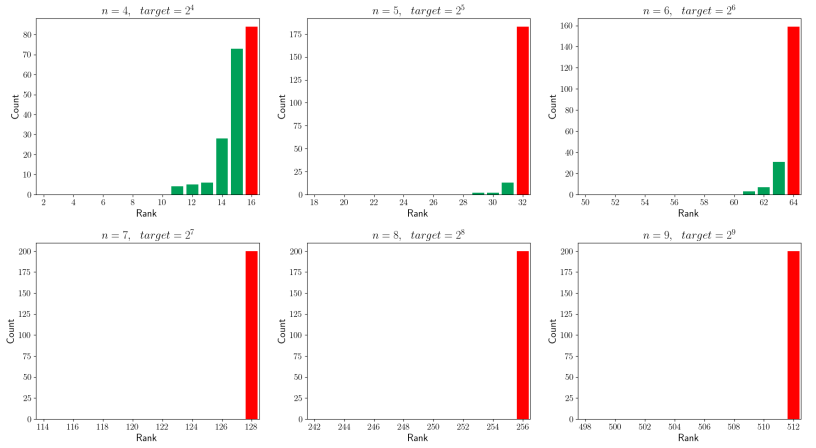
However, in order to optimize the efficiency of the homomorphic evaluation of the decomposition, we need a *cost model* to quantify the quality of a given decomposition and select the most efficient one. We define this cost model in the next section.

4.3.2 Selection of the Best Decomposition

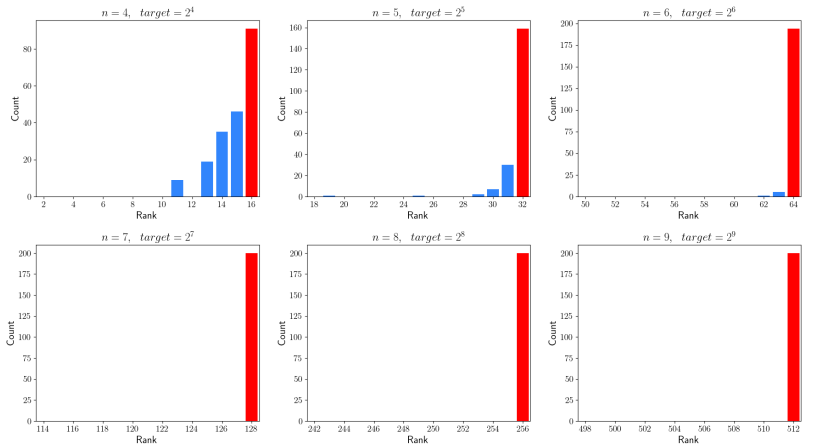
In TFHE, the PBS is by far the slowest operation by several orders of magnitude. Thus, it may seem that finding an efficient decomposition can be boiled down to reducing the number of PBS. However, the speed of the PBS itself is dependent the parameters picked for the TFHE scheme. Thankfully, extensive work has been done to try to optimize the



(a) $\gamma = 1$



(b) $\gamma = 1.05$



(c) $\gamma = 1.1$

Figure 5: Distribution of the ranks of 200 matrices $\mathcal{A}$ generated using our algorithm, for different values of n and γ. Increasing γ to 1.05 almost guarantees finding a full-rank matrice quickly. The experiments were carried out for $s = 2$, $p = 3$, and a single-output function. Full-rank matrices are shown in red.

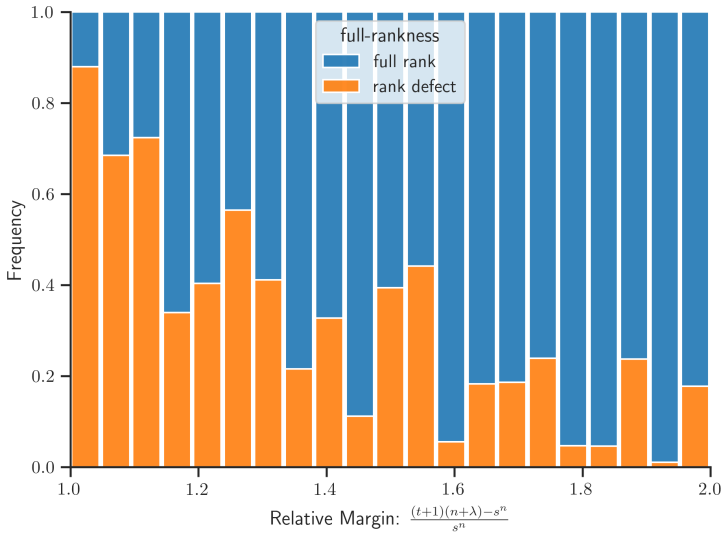


Figure 6: Correlation between the (normalized) margin on the bound of Equation 8 $\Delta = \frac{(t+1)(n+\lambda)-s^n}{s^n}$ and the proportion of full-rank matrices generated by the algorithm. Data acquired on 10 000 matrices, with $s = 2$ and $p = 3$.

parameters of TFHE in a given context to minimize the running time of the PBS operation while maintaining the correctness of the homomorphic computation. In the following, we explain how we select parameters in our own optimization framework.

In [BBB+23], the authors define the notion of an *atomic pattern*. It refers to a suite of homomorphic operations whose output noise is *independent* from the input noise. Evaluating an atomic function (introduced in Definition 1) consists in computing a linear combination on the input ciphertexts, then performing a PBS. This match exactly the simple atomic pattern denoted by $\mathcal{A}^{(\mathcal{CJP})}$ in [BBB+23] (that has been defined for the use-case of [CJP21]). Some $\mathcal{A}^{(\mathcal{CJP})}$ patterns also arise in the evaluation of the decomposition: a linear combination is performed between the vectors β_i and the ciphertexts produced by the chain, then a product is evaluated using two PBS.

To construct a cost model for the PBS, the elements to take into account are:

- the target security level λ, that corresponds to the computational hardness of the underlying LWE instance,
- the error probability ϵ, which is the probability that the noise overflows over some bits of the message during the computation,
- the size of the embedding field p,
- the norms of the linear combinations the ciphertexts go through. Such linear combinations happen in the computations of the atomic functions ($\|\boldsymbol{\alpha}_i\|$), and in the evaluation of the decomposition itself ($\|\boldsymbol{\beta}_i\|$ and $\|\mathbf{d}_i\|$). In practice, we consider the worst one of all the circuit ν_{max}.

We used an internal tool to generate the parameter sets for TFHE, although our methodology would remain applicable with any other cost model that takes these four parameters as input. For the purposes of our experiments, we relied on this tool to dynamically derive a set of parameters for each candidate decomposition we generated.

Conceptually, the use of this tool can be abstracted by modeling the computational cost of each PBS within a given decomposition by an oracle (which, in practice, corresponds to a call to our tool):

$$Cost(PBS) = \mathcal{O}(\lambda, \epsilon, p, \nu_{max}) \ .$$

Having established a model for the cost of a single PBS, we can estimate the cost of the entire decomposition by simply multiplying this cost by the total number of PBS operations required:

$$Cost = Cost(PBS) \times \#PBS$$

where

$$\#PBS = \lambda + 2t$$

For clarity, this last equation can be broken down as follows: the evaluation of the chain requires λ PBS, while the t products are each evaluated using two PBS, hence the term $2t$.

For the sake of clarity, we can break down this last equation: the evaluation of the chain takes λ PBS, and the t products are evaluated in $2t$ PBS.

In summary, we now have both a procedure to construct valid decompositions for single-output functions by sampling random chains, and a means to estimate their cost.

5 Optimization of the Method

In this section, we present two optimizations: the first improves the performance of the resulting decomposition, while the second increases the probability of successfully finding one.

5.1 Optimization of the Multi-Output Case

Having established a method to compute a single-output function

$$f : \mathbb{Z}_s^n \mapsto \mathbb{Z}_s,$$

we now extend our approach to handle functions with multiple outputs:

$$F : \mathbb{Z}_s^n \mapsto \mathbb{Z}_s^m.$$

As already mentioned, a straightforward approach is to treat the m outputs independently and apply the method separately to each of them. However, better performance for the homomorphic evaluation can be achieved by leveraging the optimization introduced in this section.

The main idea is to first evaluate the first output (denoted f_0) using the approach introduced in Section 4, and then reuse as much intermediate computation as possible to speed-up the evaluation of the remaining $m-1$ outputs of F (namely, the functions f_k for $k \in 1, \ldots, m-1$). A straightforward optimization consists in reusing the same chain to evaluate the λ intermediate values $(x_n, \ldots, x_{n+\lambda-1})$, which saves $(m-1) \times \lambda$ PBS operations during the homomorphic evaluation. Inspired by the approach of [GR16, GRVV17], we can in fact push this optimization further.

After completing the evaluation of the first output, we have access to several variables that are, with high probability, linearly independent:

- the n fresh inputs of the function f_0,
- the λ random variables computed through the chain,

- the t_0 product results $\{\rho_i\}_{0 \leq i < t_0}$ defined as:

$$\forall 0 \leq i < t_0, \rho_i = \left(\sum_{j=0}^{L_0-1} \beta_{ij} \cdot x_j \right) \cdot \left(\sum_{j=0}^{L_0-1} d_{ij} \cdot x_j \right) \tag{9}$$

  with $L_0 = n + \lambda$.

Altogether, this provides a total of $L_1 = L_0 + t_0$ variables that can serve as a new basis. We can then choose a new value t_1, which is constrained by Equation 10:

$$s^n \leq L_1 \cdot (t_1 + 1) \tag{10}$$

Compared to Equation 8, this allows t_1 to be smaller than the previous value t_0. Recall that t_1 represents the number of ciphertext-ciphertext products in the decomposition, each requiring two PBS. Hence, as more outputs are evaluated, the number of PBS required per output progressively decreases.

We then follow the same procedure than for the first output: we sample a new random matrix $\mathcal{D}^{(1)} \in \mathbb{F}_p^{t_1 \times L}$ and we construct a new matrix $\mathcal{A}^{(1)} = \mathcal{V}^{(1)} \square \mathcal{U}^{(1)} \in \mathbb{F}_p^{s^n \times (t_1+1) \cdot L_1}$ where $\mathcal{U}^{(1)}$ is the matrix representation of the chain $\Phi_{n,\lambda}$ augmented with t_0 columns containing the corresponding results of products $(\rho_i)_{0 \leq i < t}$ and $\mathcal{V}^{(1)} = \mathcal{U}^{(1)} \cdot (D^{(1)})^T$. The matrix $\mathcal{U}^{(1)}$ can be defined as:

$$\mathcal{U}^{(1)} = \begin{pmatrix} \mathcal{U}^{(0)} & \mathcal{P}^{(0)} \end{pmatrix} \in \mathbb{F}_p^{s^n \times L_1} \tag{11}$$

where $\mathcal{P}^{(0)} \in \mathbb{F}_p^{s^n \times t_0}$, as explained previously, compiles the results of each products:

$$\mathcal{P}^{(0)} = \begin{pmatrix} \rho_0^{(0)} & \cdots \rho_{t_0-1}^{(0)} \\ \rho_0^{(1)} & \cdots \rho_{t_0-1}^{(1)} \\ \vdots & \ddots \quad \vdots \\ \rho_0^{(s^n-1)} & \cdots \rho_{t_0-1}^{(s^n-1)} \end{pmatrix}.$$

If $\mathcal{A}^{(1)}$ is full-rank, it is then possible to solve the equation:

$$\mathbf{y}^{(1)} = \mathcal{A}^{(1)} \cdot \boldsymbol{\beta}^{(1)} \tag{12}$$

and construct a decomposition. The same procedure can then be carried out iteratively on the rest of the outputs of the function F. We summarize this procedure in Algorithm 4, which relies on several subroutines detailed in the following paragraphs.

`OptimalLambda`. Given a configuration (s, n) and a γ parameter, the shape of the decomposition solely depends on the parameter λ. Indeed, according to Equation 8 and the definition of the γ parameter, the successive parameters can be defined recursively as:

$$\begin{cases} L_0 = n + \lambda \\ t_k = \left\lceil \frac{\gamma \cdot s^n}{L_k} - 1 \right\rceil \\ L_{k+1} = L_k + t_k \end{cases}$$

The initial chain needs λ PBS, and then each decomposition takes $2t_k$ PBS. So we run an exhaustive search on the value of λ to minimize the number of PBS, which is given by:

$$\min_{\lambda \in \{0,\ldots,s^n-1\}} \lambda + 2 \sum_{k=0}^{m-1} t_k(\lambda) \ .$$

Table 1 shows the optimal shapes (*i.e.*, with $\gamma = 1$) for different (s, n) configurations. Table 2 shows the degradation of the number of PBS required to evaluate the decomposition when increasing γ.

Algorithm 4 `CreateDecompositions` - Generation of decompositions for a function F

Context: $\mathbb{F}_p$: The finite field of embedding
Input: $F : \mathbb{Z}_s^n \mapsto \mathbb{Z}_s^m$: the function to evaluate.
Result: $\left(\boldsymbol{\beta}^{(0)}, \ldots, \boldsymbol{\beta}^{(m-1)}\right) \in \prod_{k=0}^{m-1} \mathbb{F}_p^{(t_k+1)\cdot L_k}$: the coefficients of each decomposition

$\lambda \leftarrow$ `OptimalLambda`(s, n, γ)
$L_0 \leftarrow n + \lambda$
$\Phi^{(base)} \xleftarrow{\$} \Phi_{n,\lambda}$; `/* Sample a random chain */`
$\mathcal{U}^{(0)} \leftarrow$ `Mat`$(\Phi^{(base)}, \mathbb{Z}_s)$
$t_0 = \left\lceil \frac{s^n}{L_0} - 1 \right\rceil$
for $k \in \{0, \ldots, m-1\}$ **do**
  `/* Sample random chains and matrices` $\mathcal{D}$`, until it produces a full-rank matrix */`
  do
    $\mathcal{D}^{(k)} \xleftarrow{\$} \mathbb{F}_p^{t_k \times L_k}$
    $\mathcal{V}^{(k)} \leftarrow \left(\mathcal{U}^{(k)} \cdot (D^{(k)})^T \mid \mathbf{1}\right)$
    $\mathcal{A}^{(k)} \leftarrow \mathcal{V}^{(k)} \square \mathcal{U}^{(k)}$
  while `Rank`$(A^{(i)}) < s^n$;
  `/* Solve the linear system to get the coefficients of the decomposition */`
  $\boldsymbol{\beta}^{(k)} \leftarrow$ `GaussElimination`$(\mathcal{A}^{(k)}, \mathbf{y}^{(k)})$
  `/* Compute intermediary products */`
  $\mathcal{P}^{(k)} \leftarrow$ `ComputeProducts`$(\mathcal{A}^{(k)}, \boldsymbol{\beta}^{(k)})$
  `/* Append them to the pool of available variables */`
  $\mathcal{U}^{(k+1)} \leftarrow \left(\mathcal{U}^{(k)} | \mathcal{P}^{(k)}\right)$
  `/* Compute the shape of the decomposition of the next output */`
  $L_{k+1} \leftarrow L_k + t_k$
  $t_{k+1} \leftarrow \left\lceil \frac{s^n}{L_k} - 1 \right\rceil$
end
return $\left(\boldsymbol{\beta}^{(0)}, \ldots, \boldsymbol{\beta}^{(m-1)}\right)$

`Rank` and `GaussianElimination`. In concrete implementations, both procedures are actually run at the same time: a Gaussian elimination is performed on $\mathcal{A}^{(k)}$, and a solution $\boldsymbol{\beta}^{(k)}$ is found only if $\mathcal{A}^{(k)}$ is full rank.

`ComputeProducts`. This subroutine produces the matrix $\mathcal{P}^{(k)}$, already introduced above. This is a matrix of shape $s^n \times t_k$, defined by:

$$\mathcal{P}^{(k)} = \left(\mathcal{U}^{(k)} \cdot (\mathcal{B}^{(k)})^T\right) \odot \left(\mathcal{U}^{(k)} \cdot (D^{(k)})^T\right) \tag{13}$$

where $\mathcal{B}^{(k)}$ is a matrix containing the coefficients of $\boldsymbol{\beta}^{(k)}$ rearranged in two dimensions as:

$$\mathcal{B}^{(k)} = \begin{pmatrix} \beta_{0,0}^{(k)} & \beta_{0,1}^{(k)} & \cdots & \beta_{0,L_k-1}^{(k)} \\ \beta_{1,0}^{(k)} & \beta_{1,1}^{(k)} & \cdots & \beta_{1,L_k-1}^{(k)} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{t_i-1,0}^{(k)} & \beta_{t_i-1,1}^{(k)} & \cdots & \beta_{t_i-1,L_k-1}^{(k)} \end{pmatrix}$$

Table 1: Optimal parameters for the shapes of the matrices for different values of s and n (with $\gamma = 1$).

(a) $s = 2$

s	p	n	λ	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	#PBS
2	3	4	2	2	1	1	1	-	-	-	-	-	-	-	-	-	-	12
2	3	5	7	2	2	1	1	1	-	-	-	-	-	-	-	-	-	21
2	3	6	10	3	3	2	2	2	2	-	-	-	-	-	-	-	-	38
2	3	7	19	4	4	3	3	3	2	2	-	-	-	-	-	-	-	61
2	3	8	29	6	5	5	4	4	4	3	3	-	-	-	-	-	-	97
2	3	9	50	8	7	6	6	5	5	5	5	4	-	-	-	-	-	152
2	3	10	73	12	10	9	8	8	7	7	7	6	6	-	-	-	-	233
2	3	11	110	16	14	13	12	11	10	10	9	9	9	8	-	-	-	352
2	3	12	177	21	19	17	16	15	14	14	13	12	12	11	11	-	-	527
2	3	13	253	30	27	25	23	22	20	19	18	18	17	16	16	15	-	785
2	3	14	368	42	38	35	32	30	29	27	26	25	24	23	22	22	21	1160

(b) $s = 4$

s	p	n	λ	t_0	t_1	t_2	t_3	t_4	t_5	t_6	#PBS
4	5	2	4	2	1	-	-	-	-	-	10
4	5	3	13	3	3	2	-	-	-	-	29
4	5	4	25	8	6	5	5	-	-	-	73
4	5	5	64	14	12	10	9	8	-	-	170
4	5	6	128	30	24	21	19	17	16	-	382
4	5	7	281	56	47	41	37	34	32	30	835

(c) $s = 16$

s	p	n	λ	t_0	t_1	t_2	t_3	#PBS
16	17	2	27	8	6	-	-	55
16	17	3	108	36	27	23	-	280
16	17	4	514	126	101	87	78	1298

and $\odot$ is the coefficient-wise product. It can be checked that $(\mathcal{P}^{(k)})_{jj'}$ corresponds to the j'-th term of the decomposition (Equation 1), when evaluated with the vector $\mathbf{x}^{(j)}$ as input.

Overview of the Algorithm. We provide in Figure 7 an overview of the first steps of the algorithm, and we detail them below:

1. A chain $\Phi_{n,\lambda}$ is sampled and its matrix representation $\mathcal{U}^{(0)}$ is constructed.
2. A matrix $\mathcal{D}^{(0)}$ is sampled.
3. Using these two matrices, we compute the matrices $\mathcal{V}^{(0)}$ and $\mathcal{A}^{(0)} = \mathcal{V}^{(0)} \square \mathcal{U}^{(0)}$.
4. Using a Gaussian elimination, the vector $\boldsymbol{\beta}^{(0)}$ corresponding to the LUT first output $\mathbf{y}^{(0)}$ is computed. If $\mathcal{A}^{(0)}$ is not full rank, then we go back to Step 1. and sample a new chain and a new matrix.
5. We construct the t_0 variables products (that will be summed in the decomposition formula of Equation 1). Those products are re-used in the base of the next chain.

Table 2: Evolution of the number of PBS required to evaluate a decomposition with respect to the value of γ.

(a) $s = 2$ and $p = 3$

n	#PBS ($\gamma = 1$)	#PBS ($\gamma = 1.05$)	#PBS ($\gamma = 1.1$))
4	12	13	13
5	21	22	23
6	38	39	40
7	61	63	65
8	97	100	103
9	152	156	160
10	233	238	244
11	352	361	370
12	527	541	554
13	785	802	825
14	1160	1188	1217

(b) $s = 4$ and $p = 5$

n	#PBS ($\gamma = 1$)	#PBS ($\gamma = 1.05$)	#PBS ($\gamma = 1.1$))
2	10	11	11
3	29	30	31
4	73	75	77
5	170	174	179
6	382	392	401
7	835	855	876

(c) $s = 16$ and $p = 17$

n	#PBS ($\gamma = 1$)	#PBS ($\gamma = 1.05$)	#PBS ($\gamma = 1.1$))
2	55	57	58
3	280	287	293
4	1298	1330	1362

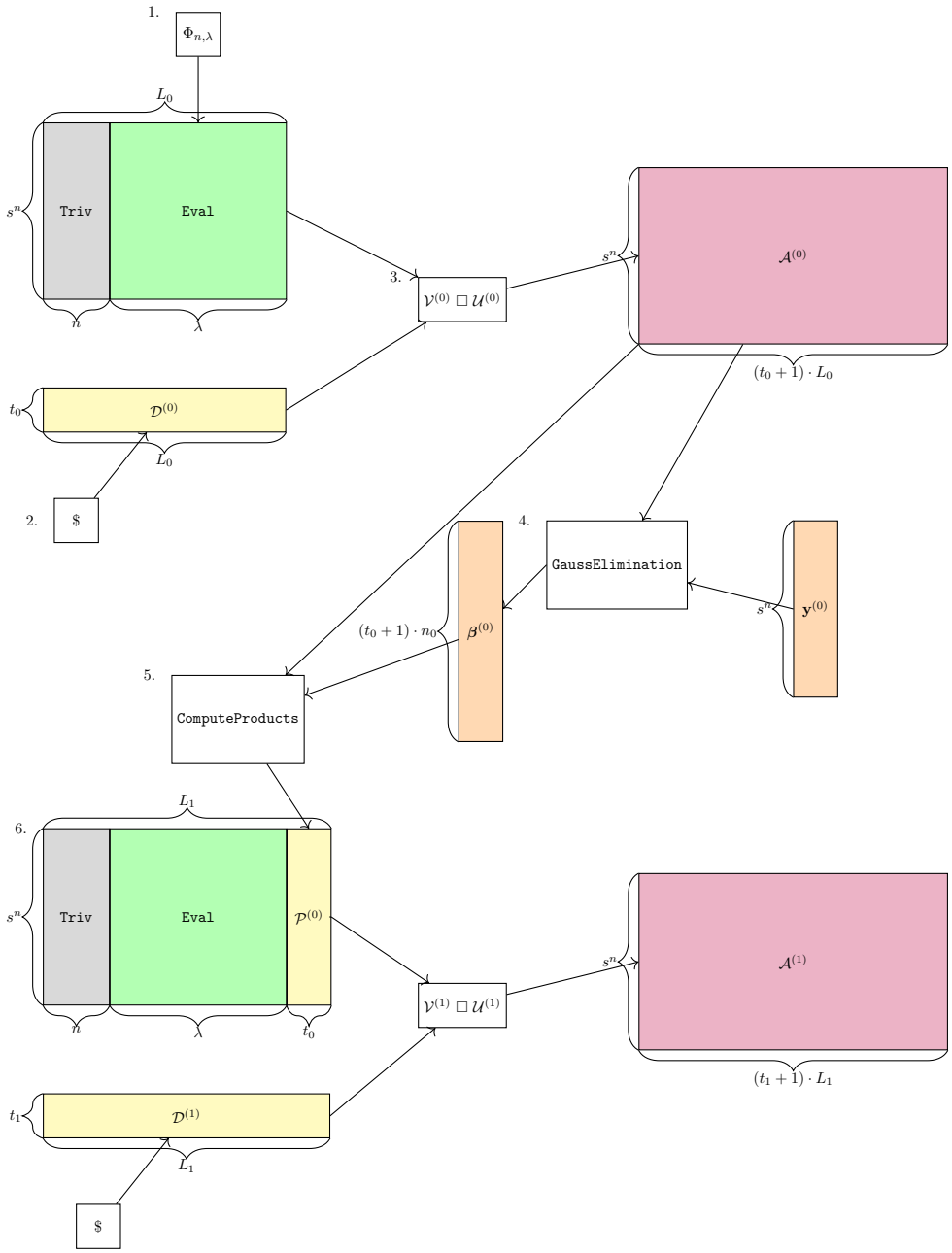


Figure 7: Overview of an iteration of the decomposition search algorithm. The outputs are the vectors $\boldsymbol{\beta}^{(i)}$, which define the decomposition of the function F represented by the vectors $\mathbf{y}^{(i)}$. The figure also illustrates the construction of the subsequent matrices $\mathcal{U}^{(1)}$ and $\mathcal{A}^{(1)}$.

6. Then we process the next iteration: we construct the new matrice $\mathcal{U}^{(1)}$ by appending the products $\mathcal{P}^{(0)}$ and we sample a new matrix $D^{(1)}$. The algorithm then keep on iterating to process all the m outputs.

Remark 1. Using our decomposition method requires solving a linear system once at homomorphic-program compilation time. This system must be solved separately for each

look-up table used in the homomorphic computation. When large look-up tables make this compilation step costly, one possible trade-off is to skip the multi-output optimization described above, that is, to avoid reusing intermediary results in the pool of random variables. In that setting, the same matrix inversion can be reused for any look-up table of a given size, since the same matrix $\mathcal{A}$ applies to every output chunk. As a result, for a fixed size, a single matrix $\mathcal{A}$ needs to be inverted, rather than one per output chunk and per look-up table with the multi-output optimization. The downside is an increase in online homomorphic evaluation time. A quick experiment shows that, for $s = 2$ and $n = 14$, the number of PBS operations rises from 1160 to 1313, corresponding to a 13% increase.

5.2 Improving the Search by Encoding Switching

Figure 5 shows that increasing the coefficient γ tends to produce matrices with higher ranks. However, this comes at the cost of slightly larger values of t and L, which results in additional PBS during the homomorphic evaluation. Table 2 illustrates the corresponding loss in performance as γ increases. Consequently, it is preferable to keep γ close to one in order to maintain the decomposition as small as possible.

In this section, we introduce a technique to increase the probability of finding a valid decomposition without increasing γ. The main idea is to use an alternative encoding for the output of the decomposition.

Consider a chain $\Phi_{n,\lambda}$, that produces a matrice $\mathcal{A} \in \mathbb{F}_p^{s^n \times (t+1)L}$. In the standard approach, if this matrix is not full rank, we would simply discard it and restart the process with a new random chain. Here, instead, we perform an additional step that allows us to derive a valid solution from the same chain, even when $\mathcal{A}$ is rank-deficient.

Let $\delta > 0$ denote the rank defect of $\mathcal{A}$, so that $\texttt{Rank}(\mathcal{A}) = s^n - \delta$. When performing Gaussian elimination, we obtain a vector β in which δ coefficients remain undetermined (they are not assigned during the pivoting process) and are thus arbitrarily set to zero. Computing the product $\mathcal{A} \cdot \beta$ yields a vector $\tilde{\mathbf{y}}$ that matches the target $\mathbf{y}$ on all but the last δ coordinates. We can model these coordinates as uniformly random variables. As noted in [GR16], we then have a probability of $1/p^\delta$ that these coefficients match the target $\mathbf{y}$.

We can go further by trying to find a valid output encoding $\mathcal{E}_{\mathsf{out}}$ such that:

$$(\mathcal{E}_{\mathsf{out}}(y_{s^n-\delta}), \ldots, \mathcal{E}_{\mathsf{out}}(y_{s^n-1})) = (\tilde{y}_{s^n-\delta}, \ldots, \tilde{y}_{s^n-1}) \ .$$

As a concrete example, assume we have $\delta = 3$, and the last three outputs of the S-box are $(y_{s^n-3}, y_{s^n-2}, y_{s^n-1}) = (0, 1, 0)$, while the algorithm produces $(\tilde{y}_{s^n-3}, \tilde{y}_{s^n-2}, \tilde{y}_{s^n-1}) = (0, 1, 2)$. By defining an encoding $\mathcal{E}_{\mathsf{out}} = \begin{cases} 0 \mapsto 0, 2 \\ 1 \mapsto 1 \end{cases}$, we obtain a correct match. During homomorphic evaluation, the output can be mapped back to the original encoding using a single PBS evaluating $\mathcal{E}_{\mathsf{out}}^{-1}$, which operates on an input of size p. The output encoding $\mathcal{E}_{\mathsf{out}}$ can be deterministically constructed directly from the vectors $\mathbf{y}$ and $\tilde{\mathbf{y}}$. The procedure is given in Algorithm 5.

We conducted experiments to evaluate the improvement in success probability brought by this "encoding switching" technique. The results are reported in Table 3. In these experiments, we sampled 200 matrices and output vectors $\mathbf{y}$ at random in $\mathbb{Z}_{s^n}$ corresponding to n-bit LUTs (with $s = 2$ and $p = 3$) for several values of γ. The results show a clear increase in the algorithm's success rate across all configurations.

Algorithm 5 `EncodingConstruction` - Construction of an output encoding

Context: $\begin{cases} \mathbb{Z}_s^n\text{: the input space of the LUT.} \\ \mathbb{F}_p\text{: the field of embedding.} \\ (y_0, \ldots, y_{s^n-1}) \in \mathbb{F}_p^{s^n}\text{: the actual outputs of the LUT.} \end{cases}$

Input: $\begin{cases} \delta\text{: the rank defect of } \mathcal{A}. \\ (\tilde{y}_0, \ldots, \tilde{y}_{s^n-1}) \in \mathbb{F}_p^{s^n}\text{: the outputs of the algorithm.} \\ \forall i \in \{0, \ldots, s^n - \delta - 1\},\ y_i = \tilde{y}_i \end{cases}$

Result: $\mathcal{E}_{\mathsf{out}}$: the output encoding (can be $\perp$ if the algorithm fails).

$$\mathcal{E}_{\mathsf{out}} \leftarrow \begin{cases} 0 \mapsto \{0\} \\ \vdots \\ s-1 \mapsto \{s-1\} \end{cases}$$

for $i \in \{s^n - \delta, \ldots, s^n - 1\}$ **do**
  if $\tilde{y}_i \in \mathcal{E}_{out}(y_i)$ **then**
    pass
  end
  else if $\tilde{y}_i \in \bigcup_{\substack{x \in \mathbb{Z}_s \\ x \neq y_i}} \mathcal{E}_{out}(x)$ **then**
    return $\perp$
  end
  else
    $\mathcal{E}_{\mathsf{out}}(y_i) \leftarrow \mathcal{E}_{\mathsf{out}}(y_i) \cup \{\tilde{y}_i\}$
  end
end
return $\mathcal{E}_{out}$

Table 3: For each experiment, 200 random matrices were generated. For each matrix, we computed its rank, and in case of a rank defect we ran `EncodingConstruction` to determine whether a valid encoding could be found. We report both the proportion of full-rank matrices and the success rate of `EncodingConstruction`. All experiments were conducted with $s = 2$ and $p = 3$.

n	$\gamma = 1$		$\gamma = 1.05$		$\gamma = 1.1$	
	Full rank	Success (enc. switch)	Full rank	Success (enc. switch)	Full rank	Success (enc. switch)
4	9 %	66 %	48 %	90 %	50 %	89 %
5	76 %	97 %	90 %	98 %	66 %	98 %
6	0 %	47 %	84 %	96 %	99 %	100 %
7	0 %	24 %	100 %	100 %	100 %	100 %
8	0 %	0 %	100 %	100 %	100 %	100 %
9	0 %	1 %	100 %	100 %	100 %	100 %

6 Experimental Results

In the following, we compare our work to the WoP-PBS technique [BBB+23], in its version with 1, 2 and 4 messages (called *blocks* in the WoP-PBS paper). We also present a comparison with the Tree-Based Method of [GBA21], in its version with 2 and 3 messages.

To compare our method with the WoP-PBS and the TBM, we performed some benchmarks on the same machine for different sizes of LUT. Our implementation can be found at https://github.com/CryptoExperts/artifact-HLUT-TCHES, and relies on the library

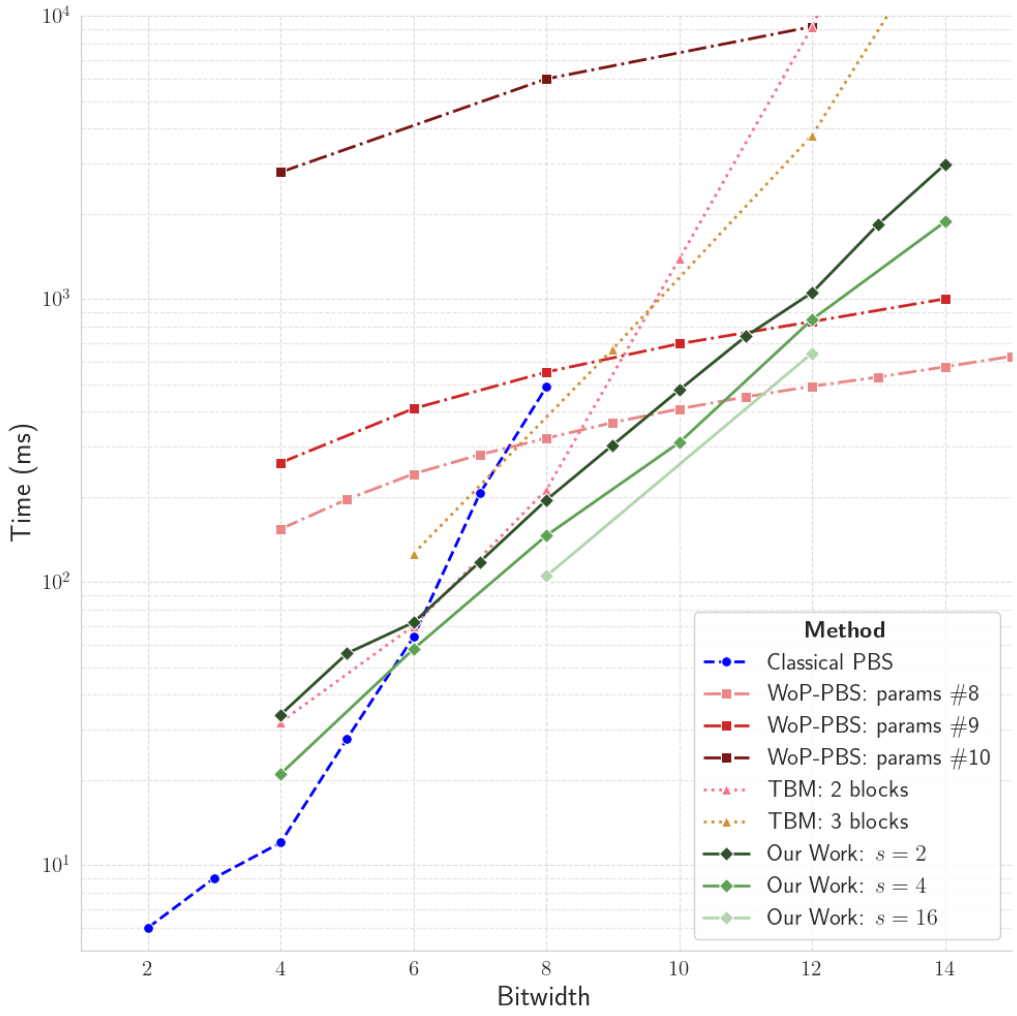


Figure 8: Comparison of timings of the classical PBS, the WoP-PBS, TBM and our work. For our work, we plot here the results for LUT of output size equal to the input size. TBM and our method use parameters ensuring a failure probability of 2^{-40}. The WoP-PBS parameters are taken from [BBB+23] and ensure a failure probability of $2^{-13.9}$. Experiments run on a server equipped with an AMD Ryzen Threadripper PRO 7995WX with 96 cores, with a maximal frequency of 5.4 GHz and 528 GB of RAM.

`tfhe-rs` [Zam22]. For the WoP-PBS, no code is provided directly with the paper [BBB+23], but an implementation of the WoP-PBS operator can be found in the artifact[4] of the follow-up paper [BCL+25]. We used this code to benchmark the WoP-PBS technique. For the TBM, we implemented it using `tfhe-rs` as well.

Using our approach, we generated decompositions for random LUTs of size 2^n, where $4 \leq n \leq 14$ and measured the time of execution of the LUT in the homomorphic domain. We evaluated multiple chunk sizes ($s = 2$, 4 and 16), all followed by a small prime (resp. 3, 5 and 17) that is used for the value of p. Parameters were selected to ensure an error probability of 2^{-40}.

Figure 8 summarizes our experimental results. It shows the runtime for homomorphic

[4]The artifact for WoP-PBS can be found at https://artifacts.iacr.org/tches/2025/a16/

LUT evaluation using our technique with respect to the input size of the LUT, and compare it with the classical PBS, the WoP-PBS and the TBM. Note that in these experiments, the output size of the LUT is the same as the input size.

For the WoP-PBS, picking cryptographic parameters was tricky because they are not provided for all the LUT sizes in the [BBB+23] paper. We have considered several sets of parameters: the ones directly provided in the artifact of [BCL+25] and the ones provided in Table 2 of [BBB+23] (that target the use-case of 16-bits and 32-bits operations). We kept the ones that showed the best performances, namely the sets #8, #9 and #10 in the latter table. Note that those parameters have been dimensioned for larger LUT sizes that in our experiments, and that they ensure a maximum failure probability of $2^{-13.9}$ (vs. the 2^{-40} that ours guarantee). For the TBM, we used the hardcoded parameters `PARAM_MESSAGE_X_CARRY_1_KS_PBS` in the `tfhe-rs` library, that are dimensioned for a failure probability of 2^{-40}.

Our method outperforms the state of the art for precisions between 6 and 10 bits, and is asymptotically better than the TBM. Moreover, its implementation is conceptually much simpler than the other methods, as it simply requires an implementation of the standard PBS and not any advanced homomorphic operators. While [BBB+23] reports results up to 24 bits, our method does not realistically scale that far. Although there is no theoretical barrier, generating a decomposition at such precision would require solving a linear system of size 2^{24}, which is computationally impractical with our experimental setup.

Moreover, our method is the fastest for 8-to-8-bit LUTs, which corresponds to the size of the AES S-box, which has been a very active research area in the recent years. Interestingly, it could be used as a building block to replace the `SubBytes` step in [BBB+25] (that is currently done with a TBM). With a block size $s = 1$, the entire AES algorithm could be evaluated using only Boolean representation, which would remove the need for the costly conversions between Boolean and arithmetic layers used in this previous work.

Finally, since our technique computes each output block sequentially, the evaluation of a LUT with smaller output width (for a fixed input size) is faster. Figure 9 displays the execution time for all possible output sizes.

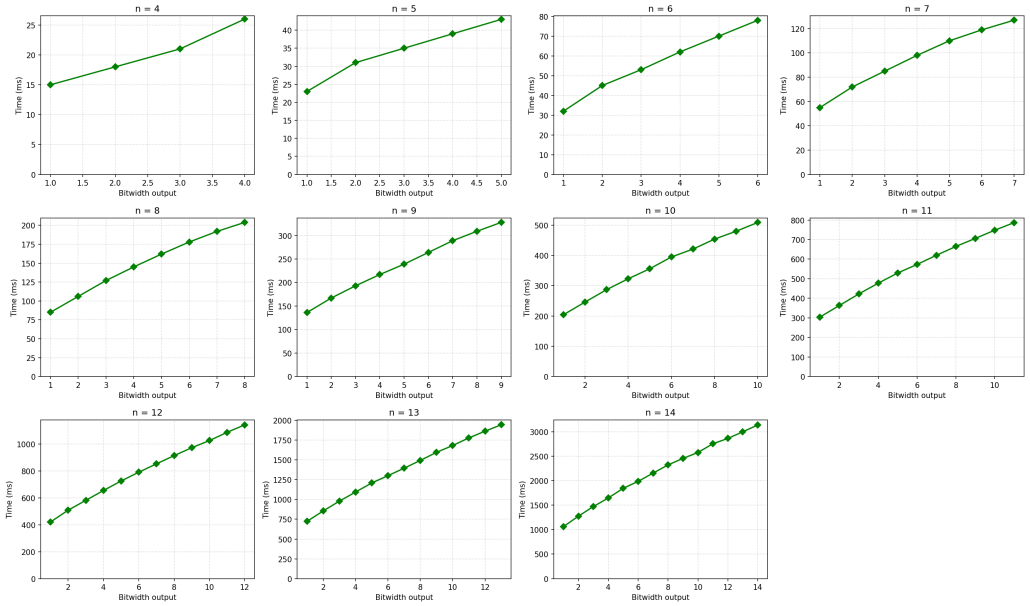
7 Conclusion

With this work, we have shown that homomorphic evaluations of look-up tables over large plaintext spaces can be accelerated by decomposing them into several smaller ones. We provided algorithms to generate such decompositions for arbitrary look-up tables, along with metrics demonstrating significant performance improvements. Moreover, we released an open-source tool that implements these decomposition techniques which is, to the best of our knowledge, the only available tool able to compute LUTs with large precisions.

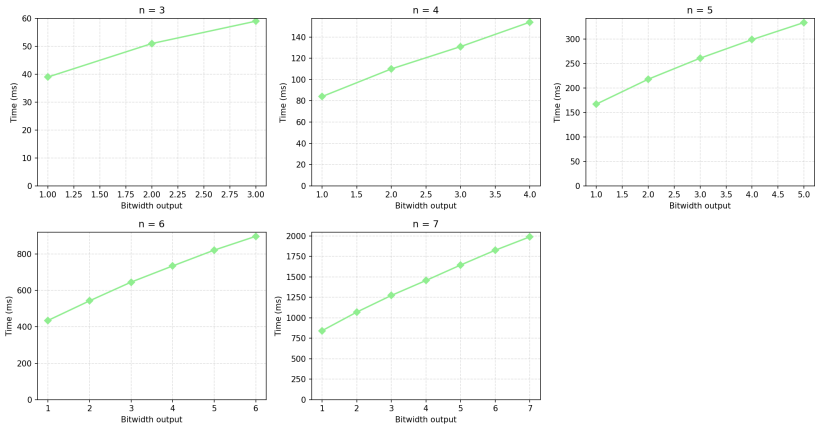
A promising direction for future work is to develop a deeper theoretical understanding of the probability of finding a valid decomposition as a function of the chosen parameters. In this work, we relied on a randomized trial-and-error approach, which proved effective in practice. However, it would be valuable to estimate the expected time to find a valid decomposition for arbitrarily large input spaces.

Acknowledgements

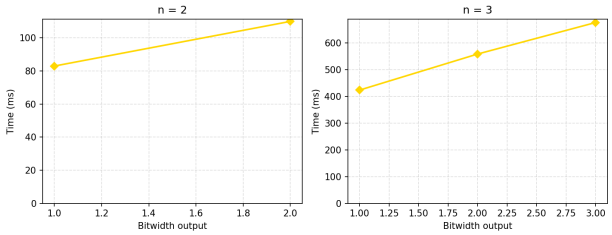
This work was supported by the Project ANR-21-CE39-0012-06 SWAP.



(a) $p = 3$



(b) $p = 5$



(c) $p = 17$

Figure 9: Comparison of running times for different configurations and output sizes. The results show that evaluating LUTs with smaller output sizes is more efficient than with larger ones, which is not the case for PBS and WoP-PBS.

References

[BBB+23] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization and larger precision for (T)FHE. *Journal of Cryptology*, 36(3):28, July 2023.

[BBB+25] Sonia Belaïd, Nicolas Bon, Aymen Boudguiga, Renaud Sirdey, Daphné Trama, and Nicolas Ye. Further improvements in AES execution over TFHE. *CiC*, 2(1):39, 2025.

[BCL+25] Loris Bergerat, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. TFHE gets real: an efficient and flexible homomorphic floating-point arithmetic. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(2):126–162, 2025.

[BPR24] Nicolas Bon, David Pointcheval, and Matthieu Rivain. Optimized homomorphic evaluation of boolean functions. *IACR TCHES*, 2024(3):302–341, 2024.

[CBG+20] Léopold Cambier, Anahita Bhiwandiwalla, Ting Gong, Mehran Nekuii, Oguz H. Elibol, and Hanlin Tang. Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks. *CoRR*, abs/2001.05674, 2020.

[CBSZ23] Pierre-Emmanuel Clet, Aymen Boudguiga, Renaud Sirdey, and Martin Zuber. ComBo: A novel functional bootstrapping method for efficient evaluation of nonlinear functions in the encrypted domain. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 317–343. Springer, Cham, July 2023.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Berlin, Heidelberg, December 2016.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 377–408. Springer, Cham, December 2017.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.

[CIM19] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New techniques for multi-value input homomorphic evaluation and applications. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 106–126. Springer, Cham, March 2019.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. *Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks*, volume 12716 of *Lecture Notes in Computer Science*, page 1–19. Springer International Publishing, Cham, 2021.

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 742–763. Springer, Berlin, Heidelberg, August 2015.

[CRV14] Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 170–187. Springer, Berlin, Heidelberg, September 2014.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Berlin, Heidelberg, April 2015.

[DMM+18] Dipankar Das, Naveen Mellempudi, Dheevatsa Mudigere, Dhiraj D. Kalamkar, Sasikanth Avancha, Kunal Banerjee, Srinivas Sridharan, Karthik Vaidyanathan, Bharat Kaul, Evangelos Georganas, Alexander Heinecke, Pradeep Dubey, Jesús Corbal, Nikita Shustrov, Roman Dubtsov, Evarist Fomenko, and Vadim O. Pirogov. Mixed precision training of convolutional neural networks using integer operations. *CoRR*, abs/1802.00930, 2018.

[GBA21] Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in TFHE. *IACR TCHES*, 2021(2):229–253, 2021.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GR16] Dahmun Goudarzi and Matthieu Rivain. On the multiplicative complexity of Boolean functions and bitsliced higher-order masking. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 457–478. Springer, Berlin, Heidelberg, August 2016.

[GRVV17] Dahmun Goudarzi, Matthieu Rivain, Damien Vergnaud, and Srinivas Vivek. Generalized polynomial decomposition for S-boxes with application to side-channel countermeasures. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 154–171. Springer, Cham, September 2017.

[HLMS25] Intak Hwang, Shinwon Lee, Seonhong Min, and Yongsoo Song. Efficient full domain functional bootstrapping from recursive LUT decomposition. *IACR Cryptol. ePrint Arch.*, page 1255, 2025.

[KS23] Kamil Kluczniak and Leonard Schild. FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. *IACR TCHES*, 2023(1):501–537, 2023.

[KWW+17] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K. Bansal, William Constable, Oguz Elibol, Stewart Hall, Luke Hornof, Amir Khosrowshahi, Carey Kloss, Ruby J. Pai, and Naveen Rao. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. *CoRR*, abs/1711.02213, 2017.

[NOFN24] Shintaro Narisada, Hiroki Okada, Kazuhide Fukushima, and Takashi Nishide. Time-memory trade-off algorithms for homomorphically evaluating look-up table in TFHE. *IACR Cryptol. ePrint Arch.*, page 1114, 2024.

[NWH+25] Chao Niu, Benqiang Wei, Zhicong Huang, Zhaomin Yang, Cheng Hong, Meiqin Wang, and Tao Wei. Sok: Fhe-friendly symmetric ciphers and transciphering. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(3):583–613, Jun. 2025.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[SCC+19] Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4901–4910, 2019.

[Sly99] V. I. Slyusar. A family of face products of matrices and its properties. *Cybernetics and Systems Analysis*, 35(3):379–384, May 1999.

[TBS25] Daphné Trama, Aymen Boudguiga, and Renaud Sirdey. AES is not enough: the block ciphers zoo goes homormorphic (over TFHE). *IACR Cryptol. ePrint Arch.*, page 782, 2025.

[TCBS23] Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. A homomorphic AES evaluation in less than 30 seconds by means of TFHE. In Michael Brenner, Anamaria Costache, and Kurt Rohloff, editors, *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Copenhagen, Denmark, 26 November 2023*, pages 79–90. ACM, 2023.

[WLW+24] Benqiang Wei, Xianhui Lu, Ruida Wang, Kun Liu, Zhihao Li, and Kunpeng Wang. Thunderbird: Efficient homomorphic evaluation of symmetric ciphers in 3GPP by combining two modes of TFHE. *IACR TCHES*, 2024(3):530–573, 2024.

[WWL+23] Benqiang Wei, Ruida Wang, Zhihao Li, Qinju Liu, and Xianhui Lu. Fregata: Faster homomorphic evaluation of AES via TFHE. In Elias Athanasopoulos and Bart Mennink, editors, *ISC 2023*, volume 14411 of *LNCS*, pages 392–412. Springer, Cham, November 2023.

[YXS+21] Zhaomin Yang, Xiang Xie, Huajie Shen, Shiying Chen, and Jun Zhou. TOTA: Fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive, Report 2021/1347, 2021.

[Zam22] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.

A Tables of Experimental Results

We report in Tables 4 and 5 the timings obtained in our experiments using our technique, for parameters targeting failure probabilities of 2^{-40} and 2^{-128}, respectively. Note that the timings in Table 4 correspond to the ones displayed in Figure 8.

Table 4: Timings achieved for failure probability of 2^{-40}.

(a) Our work: $s = 2$

Bitwidth	**Timing (ms)**
4	34
5	56
6	72
7	118
8	195
9	305
10	477
11	738
12	1054
13	1841
14	2991

(b) Our work: $s = 4$

Bitwidth	**Timing (ms)**
4	21
6	58
8	146
10	311
12	847
14	1874

(c) Our work: $s = 16$

Bitwidth	**Timing (ms)**
8	105
12	644

Table 5: Timings achieved for failure probability of 2^{-128}.

(a) $s = 3$

Bitwidth	**Timing (ms)**
4	112
5	171
6	303
7	493
8	781
9	1241
10	1935
11	3022
12	4859
13	7293
14	11066

(b) $s = 5$

Bitwidth	**Timing (ms)**
4	86
6	233
8	582
10	1403
12	3226
14	7641

(c) $s = 16$

Bitwidth	**Timing (ms)**
8	464
12	2428