Large Domain Homomorphic Evaluation for BFV-like schemes via Ring Repacking

Jean-Philippe Bossuat[1*] and Malika Izabachene[2]

[1*]GAUSS LABS PTE. LTD.
[2]ETIS, UMR 8051 CY Cergy Paris Université, ENSEA, CNRS.

Abstract

Fully homomorphic encryption allows to evaluate arbitrary functions over encrypted data. A general rule of (R)LWE-based homomorphic encryption is that when the depth of a circuit increases, the quality of the ciphertext decreases. The gate bootstrapping procedure allows to manage the noise growth through the homomorphic evaluation process. FHEW-like bootstrapping enables the evaluation of discretized arbitrary functions at some additional cost compared to gate bootstrapping. One of its major limitations is that, if one wants to keep it efficient, the precision of the message encoding functions needs to be restricted to relatively small sizes. For example, functions with domains larger than 8 bits of precision become difficult to evaluate in a reasonable amount of time. A recent line of work aims to overcome these limitations to improve the functional bootstrapping efficiency over large input domains.

In this paper, we propose a different approach for the homomorphic evaluation of arbitrary functions built from ring-packing techniques that convert a set of LWE ciphertexts into a RLWE ciphertext. As an application, we propose a computational and data-efficient client-server protocol for the homomorphic evaluation of arbitrary functions defined over a large domain.

We obtain a server-side amortized time of **0.12** ms, **0.17** ms per input over batches of 2048 inputs for functions defined over $\mathbb{Z}_{\mathbf{2^{14}}}$ and $\mathbb{Z}_{\mathbf{2^{16}}}$, respectively. Our solution also enables the client to retrieve many point evaluations almost for free.

Keywords: Homomorphic Look-Up-Table, Ring Repacking, Automorphism, Leveled Homomorphic Encryption

1 Introduction

Fully Homomorphic Encryption (FHE) has become a popular encryption technique which enables secure computation on encrypted data. FHE ciphertexts contain noise that grows as homomorphic operations are performed. In order to support the evaluation of arbitrary circuits, FHE schemes rely on a bootstrapping operation, which basically updates the noise component of an FHE ciphertext. FHEW-like bootstrapping is one of the most known efficient bootstrapping techniques. Initially, DM/CGGI bootstrapping [1–3] allowed the sequential evaluation of Boolean gates. Their refined version, called functional bootstrapping [4–7], allows us to evaluate an arbitrary boolean or arithmetic function encoded with *a priori*.

However, the number of bits that can be preserved by the computation, i.e. the precision of the computation, remains relatively small for practical parameters. Moreover, increasing the bootstrapping precision further has a prohibitive impact on its efficiency. This limitation in precision poses a significant challenge when scaling up to real-world scenarios. Many real-world applications involve the processing of a large amount of data and require the evaluation of functions over large domains. This is particularly critical in contexts such as privacy-preserving traffic detection or tracing algorithms during pandemics, where both scale and accuracy are essential. In the latter case, each user is in possession of a set of sensitive attributes, including their encrypted location, which is processed to decide whether they could potentially have been in close contact with another user. In that case, a large amount of encrypted user information lying in a potentially large domain needs to be processed and compared together. In this work, we propose an efficient solution to homomorphically process the evaluation of an arbitrary function over a large discretized domain, i.e. containing a large number of different points.

Related works on high precision homomorphic arbitrary functions evaluations. Homomorphic evaluation of high precision arbitrary functions remains a central challenge in FHE, especially when targeting large-integer arithmetic. The functional bootstrapping within the DM/CGGI framework can be used to evaluate an arbitrary function. Improved strategies such as the *chaining method* [8] and the *tree-based method* [9] introduced sequential and recursive bootstrapping strategies, respectively, for evaluating arbitrary functions. These approaches have later been extended and refined in a series of works [10, 11], [12, 13], [14], which focused on improving precision, reducing the cost of bootstrapping and adapting to various classes of functions. Additionally, decomposition strategies have been explored to enable homomorphic computation over large integers: the radix and Chinese Remainder Theorem (CRT) techniques [15, 16], [13] have been introduced to break down large plaintexts, but require multiple bootstrapping invocations. Circuit bootstrapping [3], which enables switching between ciphertext formats through leveled operations, has been effectively combined with LUT-based packing strategies to evaluate complex functions over large integers [13]. However, making these methods scalable to large integers affects the performance as the parameters need to be increased.

Efficient approaches for integer arithmetic rely on discretized variants of CKKS to bootstrap bits, and small integers [17–19]. Recent progress has enabled highly efficient bootstrapping for relatively large plaintext moduli [20, 21], and even for

arbitrarily large plaintext sizes [22]. [23] make use of multivariate polynomials to evaluate LUTs with high precision in RLWE homomorphic encryption schemes. [23] reports an amortized time of 1.01ms and 0.15ms to evaluate 8 to 8 LUTs using the BFV and CKKS schemes, respectively, and 0.96ms for a 12- to 16 LUT with CKKS. These runtimes correspond to the amortized cost of evaluating 32768 LUTs in parallel. Compared to the solution from [23], our method achieves competitive server-side amortized runtimes of 0.12 ms and 0.17 ms per input for functions defined over $\mathbb{Z}_{2^{14}}$ and $\mathbb{Z}_{2^{16}}$ respectively, over batches of 2048 inputs. However, it is important to note that the functionalities and target applications differ between approaches. Specifically, the runtimes reported in [23] include the runtime for bootstrapping as part of the evaluation pipeline, while in our case we need an additional layer to support further homomorphic operations by first applying packing and homomorphic encoding.

Split Domain Approach. [24] proposed two solutions for the following problem: a client would like to know how many points in a large domain verify some property. In both solutions, a client sends encrypted points to a server and recovers the number of points that verify some criteria. In this case, the function, i.e., the selection criteria is known to both the client and the server. However, no information about the queried inputs should be revealed to the server. Using the methods from [24], the user sends the encryption of an integer to the server, which homomorphically counts how many elements in the domain meet the criteria, whereas in our case, the client retrieves which elements meet which criteria and counts by herself the elements meeting the target criteria. Hence, their solution is designed for a more complex task in mind than ours. Our work is based on the second method, called the *split domain* approach, which enables the evaluation of multivariate functions of the form $f(i_0, \cdots, i_{z-1}) := \sum_{t=0}^{z-1} \alpha_{i_t} f_t(i_t)$. As the complexity of the computation increases super linearly with N, the idea is to run several homomorphic computations with a smaller N rather than performing one computation with N, if one would like to evaluate f. Splitting the domain hence allows us to support larger domain size. We refer to [24, Section 4] for a full description of the method. In this work, we build on the split domain approach to propose a simple and efficient solution based on ring repacking to evaluate arbitrary functions over large domains. We provide more details of our methods next.

Packing techniques. In this paper, we refer to repacking as the function that maps multiple RLWE ciphertexts to one RLWE ciphertext and to packing when doing the same, but for LWE inputs. Without loss of generality, we can restrict ourselves to the setting where LWE ciphertexts are given as input, as converting an RLWE ciphertext to LWE ciphertexts can be done very efficiently using homomorphic extraction. We review below some of the existing packing techniques. Note that an RLWE ciphertext with secret key s has the form $(a, b) \in \mathsf{R}_q^2$ where R_q is the set of integer polynomials of degree less than N whose coefficients are taken modulo q. Note that (a, b) is a structured set of N LWE ciphertexts of $m(X)[i]$ under secret key $\tilde{s} = \phi(s)$, where $\phi : X^i \to X^{-i}$, of the form $(\mathsf{coeff}(X^i \cdot a_i), b[i])$ where coeff is the coefficient embedding map. This set of LWE ciphertexts can be expressed as $C := (\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{N \times N, 1 \times N}$

where $\mathbf{A} \in \mathbb{Z}_q^{N \times N}$ is the anti-circulant Vandermond matrix representation of a and $\mathbf{b} = \mathsf{coeff}(b)$. The packing technique from [3] adapted from [25], and later improved improved by ring switching [26], handles each column of C separately and switches the "column" ciphertext under the secret key $\tilde{s}$ to a new ciphertext under the secret key s. The diagonal packing encodes the diagonal of the matrix A and applies the linear transformation $A \cdot \mathsf{RLWE}(s) - b$ [27], which acts as a homomorphic decryption. The method of [28] encodes each row of the matrix C as a polynomial ciphertext, where each ciphertext is combined with the next row ciphertext using automorphisms. Our solution is built using the row packing technique.

Our contributions are as follows.

- We build a new and simple method to evaluate arbitrary functions defined over a large domain. Our technique allows a client to request selected point evaluations to a server without leaking any information on the requested inputs. The method proceeds in two steps. In the first step, we demonstrate that, by encoding the function with a well-chosen polynomial, one can directly compute ciphertexts whose constant coefficient gives the targeted evaluations. In the second phase, we merge a set of evaluated points into a single RLWE ciphertext, thus significantly increasing the size of the server response.
- We show that by expressing the circuit as a linear transformation on a structured set of N LWE ciphertexts, rather than a linear transformation over $N \times N$ matrices, we are able to take advantage of the row packing technique and reduce the server-side memory footprint from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ and the number of required switching keys from $\mathcal{O}(N)$ to $\mathcal{O}(\log N)$.
  As a result, our solution reduces the overall number of evaluation keys from $\mathcal{O}(N)$ to $\mathcal{O}(\log(N))$ and brings about a factor of improvement in $2000\times$ throughput per input on the server side over the originally proposed *split domain* approach, while maintaining all the original functionalities.
  Although we do provide a comparison between the original *split domain* method and ours, it should not be interpreted in a strict sense, as they operate under different output representations. In our solution, the target evaluation is returned as polynomial coefficients, rather than in the exponent as in [24]. We implemented our approach using the Lattigo library [29] and empirically verified its performance.
- The most complete proof of the correctness of the row-packing is provided in [28], where the algorithm is presented in its recursive form. However, that proof does not address all aspects of the row-packing algorithm. The way in which ciphertexts are ordered and combined at each step of the recursion is not made explicit. In the present work, we offer a complete bottom-up correctness proof of the iterative version of the algorithm. Although the proof in [28] is sound within its scope, our work seeks to strengthen the correctness argument and could be viewed as consolidating the row packing proof of [28].

Idea of our method. Assume a user would like to obtain the evaluation of $f : \mathsf{Dom} \mapsto \mathsf{Img}$ over a set of inputs $\{i_0, \cdots, i_{z-1}\} \subseteq \mathsf{Dom}(f)$ that the user would like to keep unknown to the server. For the first step, we treat each query input $i \in \mathsf{Dom}(f)$

asked to the server individually. The following step is applied for each query input in parallel. To simplify, we assume that the size of the subset z is less than N but the size can also be larger, as explained later. If the user wants to obtain the evaluation of f on an input i, it computes and sends the RLWE encryption of X^i, $\mathsf{RLWE}(X^i)$. The server precomputes a polynomial representation of the function f, $u_f(X)$ such that the plaintext multiplication with $\mathsf{RLWE}(X^i)$ gives the RLWE encryption of a polynomial (i) whose constant coefficient is $f(i)$ and (ii) whose other coefficients are consecutive evaluations of the function at other $N-1$ inputs in the domain; note that these other inputs are not targeted by the current query and omitted from the response. In a second phase, the server applies a repacking procedure to reduce the response size by up to a factor of N as follows: it takes as input the previous N RLWE ciphertexts whose constant coefficients are respectively $f(i_0), \cdots, f(i_{z-1})$. And it reconstructs a new RLWE ciphertext of the polynomial $f(i_0)X^0 + \cdots + f(i_{z-1})X^{z-1} + \star X^z + \cdots + \star X^{N-1}$, where $\star$ can be any element in the message space. And depending on the format agreed, an LWE encryption of $f(i)$ can be homomorphically extracted and sent back to the user or the repacked RLWE ciphertext is sent to the user who decrypts and parses the polynomial by itself to retrieve the targeted evaluations.

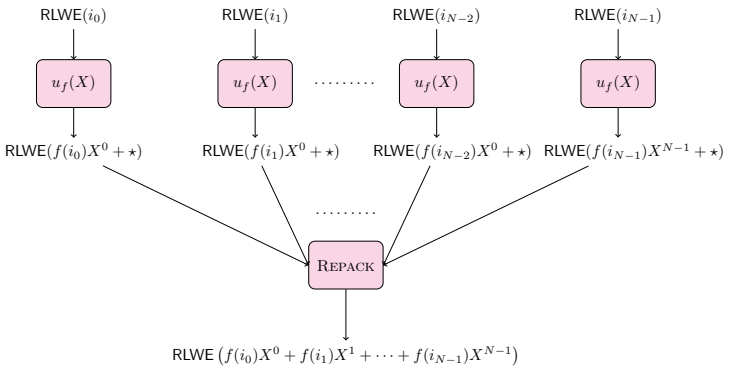

Fig. 1: Steps of the strategy to perform the homomorphic evaluation of f defined over a large domain. At the first step, the function is evaluated on each point i_j of the client query dataset by a homomorphic plaintext multiplication with $u_f(X)$. At the second step, the resulting ciphertext are combined using a repacking procedure so that a RLWE encryption whose coefficients are exactly the consecutive targeted evaluations is obtained as the end.

Practical applications of our approach. In our setting, the design prioritizes minimizing the response size of the server over the query size of the client. Our approach is well-suited to use cases where the client transmits a succinct set of ciphertexts encrypting inputs drawn from a large domain such as Private Set Intersection (PSI)

applications. In PSI protocols, two parties, each holding a private set, jointly compute the intersection of their sets without revealing any additional information about their respective inputs. PSI protocols find applications in a wide range of scenarios, including criminal investigations, cross-institutional fraud detection, or corrupted password verification, to name a few. In the latter case, a client seeks to verify whether her password appears within a list of compromised passwords held by a server without revealing her passwords in clear. To achieve this, the client encrypts her input and transmits it to the server, who then performs the oblivious intersection computation between the two lists. Our method is particularly well-suited to this setting, where the size of the list sent by the client is often much smaller than the size of the list held by the server. Our approach has also been used in the context of statistical analysis of medical data to build predictive models [30]. In this setting, our solution is used as a subroutine within the analysis protocol, operating in a dual model: the function is encrypted while the data remain in cleartext.

2 Preliminaries

2.1 Notations

We denote single elements (polynomials or numbers) in italics, e.g., a, and vectors of such elements in bold, e.g., $\boldsymbol{a}$. We denote $a[i]$ the element at position i of the vector $\boldsymbol{a}$ or the degree-i coefficient of the polynomial a. We will refer to the i-th to j-th coefficients of an interval as $[i : j]$. When the interval $[i : j]$ is a discretized subset of real values, we also denote it as $\mathbb{R}_{[i,j]}$. We will use the operator $+$ for addition between polynomials, vectors, or numbers, the operator $\cdot$ for point-wise multiplication between numbers vectors, or polynomials, $*$ for the regular multiplication between two polynomials and $\lll_k$ for the cyclic rotation by k positions to the left of a vector or a polynomial. We denote $||\cdot||$ the infinity norm, $[\cdot]_Q$, $\lfloor\cdot\rfloor$, $\lfloor\cdot\rceil$ the reduction modulo Q, rounding to the previous and to the closest integer, respectively (coefficient-wise for polynomials), and $\langle\cdot,\cdot\rangle$ the dot product. We denote $\leftarrow \chi_d^n$ the act of sampling a vector (or polynomial) of size n from a given distribution d (omitting n implies $n = 1$). Unless otherwise stated, logarithms are in base 2.

2.2 Cyclotomic Rings

For a fixed power of two $M \geq 4$, let $N = \phi(M) = M/2$ and let the N-th cyclotomic polynomial ring be $\mathsf{R}_q^N[X] = \mathbb{Z}_q[X]/(X^N + 1)$. Elements of $\mathsf{R}_q^N[X]$ are all polynomials in X with integer coefficients taken modulo q and of degree at most $N - 1$. We assume that the coefficients are centered, that is, in $[-\lfloor q/2\rfloor, \lfloor q/2\rfloor)$. For the sake of clarity and to keep notations concise, we will omit the variables N, q and/or X when denoting R when these can be trivially deduced or do not need to be defined.

Note that X has order $2N$ in R and for $N \geq 8$, $\mathbb{Z}_{2N}^\star \cong \mathbb{Z}_{\frac{N}{2}} \otimes \mathbb{Z}_2$ with respective generators 5 and -1.

The ring $\mathsf{R}_q^N[X]$ supports three well-defined operations:

- Addition: $+ : \mathsf{R} \times \mathsf{R} \rightarrow \mathsf{R}$ defined as the coefficient-wise additions between two polynomials of degree at most $N-1$ with coefficients taken modulo q.
- Multiplication: $* : \mathsf{R} \times \mathsf{R} \rightarrow \mathsf{R}$ defined as the multiplication between two polynomials of degree at most $N-1$ modulo X^N+1 with coefficients taken modulo q.
- Automorphism: $\phi_\alpha(\mathsf{R}) \rightarrow \mathsf{R}$ defined as $\phi_\alpha : X \rightarrow X^\alpha \mod (X^N+1)$ with α odd. If $\alpha = (-1)^{k_0} 5^{k_1} \mod 2N$ with k_0, k_1 explicit, ϕ_α will be denoted $\phi_{(k_0,k_1)}$. These automorphisms over R form a group for the composition law. As a morphism, ϕ_α verifies $\phi_\alpha(a) + \phi_\alpha(b) = \phi_\alpha(a+b)$ and $\phi_\alpha(a) * \phi_\alpha(b) = \phi_\alpha(a*b)$.

2.3 Learning-With-Errors Assumptions

Learning-With-Errors (LWE). We denote $D^{\mathsf{LWE}}_{N,q,h,\sigma}$ the distribution over $\mathbb{Z}_q^{1+N}$ defined by choosing $\boldsymbol{s} \leftarrow \chi_h^N$, $\boldsymbol{a} \leftarrow U(\mathbb{Z}_Q^N)$, $e \leftarrow \chi_\sigma$, setting $b = -\langle \boldsymbol{a}, \boldsymbol{s} \rangle + e$ and outputting $(\boldsymbol{a}, b) \in \mathbb{Z}_q^{N+1}$, where, $U(\mathbb{Z}_q)$ is the uniform distribution over $\mathbb{Z}_q$, χ_h is a uniform distribution over $\{-1, 0, 1\}$ with exactly h nonzero coefficients; χ_σ a discrete Gaussian distribution with standard deviation σ.

We say that an LWE parameter set $\{N, q, h, \sigma\}$ is λ-secure if the advantage of an adversary $\mathcal{A}$ to distinguish between the distributions $D^{\mathsf{LWE}}_{N,Q,h,\sigma}$ and $U(\mathbb{Z}_Q^{1+N})$ is bounded by $2^{-\lambda}$.

The Ring Learning-With-Errors (RLWE) is defined similarly. We denote $\leftarrow D^{\mathsf{RLWE}}_{N,Q,h,\sigma}$ the act of sampling from this distribution and for the rest of this work, and unless otherwise specified, the symbols a, s and e will refer to polynomials sampled from the distributions $U(\mathbb{Z}_Q^N)$, χ_h^N and χ_σ^N respectively.

Note that a RLWE sample $(a, b) \in (\mathsf{R}_q^N)^2$ can be seen as a structured matrix $\mathbb{Z}_q^{N \times (1+N)}$ composed of N LWE samples under the key $\phi_{(1,0)}(s)$ where the i-th row is $(\mathsf{coeff}(a \cdot X^i), b[i]) \in \mathbb{Z}_q^{1+N}$. In the following, a RLWE encryption of a message $m \in \mathsf{R}^N$ under secret s will be denoted $\mathsf{RLWE}_s(m)$; and a LWE encryption of a message $m \in \mathbb{Z}_q$ under secret key $\tilde{s}$ will be denoted $\mathsf{LWE}_{\tilde{s}}(m)$.

2.4 RLWE Homomorphic Encryption

To generate a RLWE ciphertext of a message m, one first samples $(c_0, c_1) = (a, -as + e) \leftarrow D^{\mathsf{RLWE}}_{N,q,h,\sigma}$. This tuple can be interpreted as the coefficients of a degree 1 polynomial $\mathcal{P}_q(\cdot) \in \mathsf{R}_q^N$ on the secret key $s \in \mathsf{R}^N$, i.e. $\mathcal{P}_q(s) = c_0 + c_1 * s = e$ with $||e||_\infty$ small. More generally, we will denote this RLWE encryption of zero at modulus q of degree k under the indeterminate $s \in \mathsf{R}^N$ as $\mathcal{P}_q^k(s) = \sum_{i=0}^k c_i s^i$. If k is omitted, it is assumed that it is 1 and if q is omitted, it is assumed that the relation holds for any q. Therefore, a RLWE ciphertext of degree k of m will be denoted as $\mathcal{P}^k(s) + m$, and the cyclotomic rings arithmetic (see Section 2.2) carries over in a natural way, allowing the underlying homomorphic properties supported by the construction:

- Addition: $(\mathcal{P}^i(s) + m_0) + (\mathcal{P}^j(s) + m_1) = \mathcal{P}^{\max(i,j)}(s) + m_0 + m_1$;
- Multiplication: $(\mathcal{P}^i(s) + m_0) * (\mathcal{P}^j(s) + m_1) = \mathcal{P}^{i+j}(s) + m_0 * m_1$;
- Automorphism: $\phi(\mathcal{P}(s) + m) \rightarrow \mathcal{P}(\phi(s)) + \phi(m)$.

2.5 RLWE Key-Switching

The RLWE keyswitching operation enables switching from an RLWE encryption under the secret key s to an RLWE encryption of the same message under secret-key s' i.e. $\mathcal{P}_q(s) \to \mathcal{P}_q(s')$ such that $\mathcal{P}(s) \approx \mathcal{P}(s')$. We give below a description of how key switching is implemented and used. Following [31]'s generalized method, we introduce key switching relatively to the combination of a base decomposition and a temporary modulus P:

- $\mathsf{SwitchKeyGen}(s, s', \boldsymbol{w})$: For $s, s' \in \mathsf{R}_{QP}^N$ and $\boldsymbol{w} = (w_0, \cdots, w_{\beta-1})$ an integer decomposition basis of β elements return the key-switch key from s to s': $\mathsf{swk}_{(s\to s')} = (\mathsf{swk}_{(s\to s')}^{(0)}, \ldots, \mathsf{swk}_{(s\to s')}^{(\beta-1)})$, where $\mathsf{swk}_{(s\to s')}^{(i)} = \mathcal{P}_{QP}(s') + w_i P s$;
- $\mathsf{SwitchKey}(d, \mathsf{swk}_{s\to s'})$: decompose $d \in \mathsf{R}_Q^N$ in base $\boldsymbol{w}$ such that $d = \langle \boldsymbol{d}, \boldsymbol{w} \rangle$ and return $\mathcal{P}_Q(s') + ds = \lfloor P^{-1} \cdot \langle \boldsymbol{d}, \mathsf{swk}_{s\to s'} \rangle \rceil \in (\mathsf{R}_Q^N)^2$ for $P^{-1} \in \mathbb{R}$.

The auxiliary modulus P is used to control the noise growth and in practice so that $\lfloor P \cdot \langle \boldsymbol{d}, \boldsymbol{e} \rangle \rceil \in [-0.5, 0.5)^N$. The $\mathsf{SwitchKey}(\cdot)$ procedure is used to perform Automorphism on RLWE ciphertexts: an automorphism ϕ on a RLWE ciphertext $\mathcal{P}(s)$ maps it to a new RLWE ciphertext $\mathcal{P}(\phi(s))$. The keyswitching operation enables to re-encrypt $\mathcal{P}(\phi(s))$ to $\mathcal{P}(s)$ by evaluating $\mathsf{SwitchKey}(\mathcal{P}^{[1]}(\phi(s)), \mathsf{swk}_{\phi(s)\to s}) + \mathcal{P}^{[0]}(\phi(s))$;

3 Ring Repacking and Plaintext Domain Switching

There are several variants of the (re)packing algorithms in the literature. Our constructions will be built from previous packing algorithms [28, 32] that we reviewed in its iterative form. The purpose of repacking is to merge $d \leq N/n$ RLWE ciphertexts of degree N, each encrypting a polynomial message of the form $m_i(Y) = \sum_{j=0}^{n-1} m_{i,j} Y^j$, where $Y = X^{N/n}$, into a single RLWE ciphertext. The resulting ciphertext encrypts a polynomial message of the form $m(X) = \sum_{i=0}^{d-1} \left(\sum_{j=0}^{n-1} m_{i,j} Y^j \right) X^i$.

For simplicity, we take $n = 1$, so that $Y = X$ and the number of input ciphertexts is $d = N$, the degree of the polynomial ring. However, the algorithm easily generalizes to any Y and any number d of inputs such that $d \leq N/n$ just by taking zero values for the remaining coefficients and if $d > N/n$, we can split the d ciphertexts into batches of up to N/n ciphertexts and evaluate one repacking per batch.

To explicitly show how the ciphertexts ordering will be affected by our repacking algorithm, we introduce the following definition, which will be used in our iterative version of the repacking algorithm:

Definition 1 (Ciphertexts Hashmap) A *ciphertext hashmap* is a list of ciphertexts, each ciphertext is associated with a unique index in $[0, n-1]$. We can re-index the elements of a ciphertext hashmap by applying a *permutation* over $[0, n-1]$. We denote as $\mathsf{rev}_n(i)$ the bit-reversal permutation of $i \in [0, n-1]$ over $\log(n)$ bits.

3.1 Iterative Ring Repacking

The algorithm takes as input N/n RLWE ciphertexts, which are divided into two groups of equal size $N/2n$, assuming that n divides N. These two groups of ciphertexts are merged into a single group, thus halving the total number of ciphertexts. This process is iterated until only one ciphertext remains. The parameter n is defined so that each input RLWE ciphertext encrypts a message of the form $m_i(Y) = \sum_{j=0}^{n-1} m_{i,j} Y^j$ for $Y = X^{N/n}$. The algorithm is described in Algorithm 1 and an illustrated example is given in Figure 2.

Algorithm 1 RLWE Repacking

Inputs: h, a hashmap containing up to N/n RLWE ciphertexts with unique indexes in the range $[0, N/n)$, where $c[i]$ denotes the encryption of $m_i(Y) = \sum_{j=0}^{n-1} m_{i,j} Y^j$ for $Y = X^{N/n}$.
Output: a RLWE encryption of $m(X) = \sum_{i=0}^{N/n-1} \left(\sum_{j=0}^{n-1} m_{i,j} X^{jN/n} \right) X^i$.

for $i \leftarrow 0$ **to** $N-1$ **do**
    if $h[i] \neq \emptyset$ **do**
        $h[i] \leftarrow (N/n)^{-1} \cdot \mathsf{h}[i]$

for $i \leftarrow \log(n)$ **to** $\log(N)-1$ **do**
    $t_i \leftarrow N/2^{i+1}$
    if $i = 0$
        $\alpha \leftarrow (1, 0)$
    else
        $\alpha \leftarrow (0, 2^{i-1})$
    for $j \leftarrow 0$ **to** $t_i - 1$
        $\mathsf{h}[j] \leftarrow \mathsf{h}[j] + \mathsf{h}[j + t_i] * X^{t_i} + \phi_\alpha(\mathsf{h}[j] - h[j + t_i] * X^{t_i})$
return $\mathsf{h}[0]$

Lemma 1 *Let* $\phi_{(k_0,k_1)} : X^i \rightarrow X^{i \cdot (-1)^{k_0} 5^{k_1}}$, *then:*

1. *Given* $t = N/2$ *and* $(k_0, k_1) = (1, 0)$,
2. *Given* $t = N/2^{j+1}$ *and* $(k_0, k_1) = (0, 2^{j-1})$ $\forall j \in [1, \log(N/2)]$,

we have $\phi_{k_0,k_1}(X^{it}) = (-1)^i X^{it}$, $\forall i \in [0, N/t)$.

Proof

1. The proof is trivial since $X^{iN/2}$ is mapped to $X^{-iN/2}$ and $X^N \equiv -1 \mod (X^N+1)$.
2. We start by showing that $i\frac{N}{2^{j+1}}(5^{2^{j-1}}) = i\frac{N}{2^{j+1}} + iN \mod 2N$:

$$
\begin{aligned}
5^{2^{j-1}} &= (2^2+1)^{2^{j-1}} \\
&= \sum_{k=0}^{2^{j-1}} \binom{2^{j-1}}{k} (2^2)^k 1^{2^{j-1}-k} \mod 2^{j+1} \\
&= 1 + 2^{j+1} \mod 2^{j+2}
\end{aligned}
$$

which gives

$$
\begin{aligned}
i\frac{N}{2^{j+1}} 5^{2^{j-1}} &= i\frac{N}{2^{j+1}}(1+2^{j+1}) \mod 2N \\
&= i\frac{N}{2^{j+1}} + iN \mod 2N
\end{aligned}
$$

Regarding the sign of X, let k be number of modular reductions by N:

$$
\begin{aligned}
k &= \left(i\frac{N}{2^{j+1}} \cdot (5^{2^{j-1}} \mod 2N) - i\frac{N}{2^{j+1}} \right) / N \\
&= \frac{i}{2^{j+1}} \cdot \left(5^{2^{j-1}} \mod 2N \right) - \frac{i}{2^{j+1}} \\
&= i \cdot \frac{\left(5^{2^{j-1}} \mod 2N \right) - 1}{2^{j+1}}.
\end{aligned}
$$

Thus the sign of X is positive when k is even, else it is negative ($X^N \equiv -1 \mod (X^N+1)$). Since $((5^{2^{j-1}} \mod 2N) - 1)/(2^{j+1})$ is necessarily odd, then the parity of k must the same as the one of i, thus we indeed have the sign being equal to $(-1)^i$.

□

Our proof makes use of this intermediate result that characterizes the coefficients embedded in the ciphertext computed at each iteration of the loop from Algorithm 1. The proof of Lemma 2 is deferred to the appendix.

Lemma 2 *For $i \in [0, \log N)$, define $t = N/2^{i+1}$ and let c, c' be ciphertexts encrypting $\mu(X) = \sum \mu_j X^j$ and $\mu'(X) = \sum \mu'_j X^j$, then line 15 of Algorithm 1 returns a ciphertext encrypting:*

$$
2 \sum_{j=0}^{N/2t-1} (\mu_{2jt} + \mu'_{2jt} \cdot X^t) \cdot X^{2jt} + \sum_{j \in [0,N) \wedge t \nmid j} * \cdot X^j \tag{1}
$$

where $*$ denotes an undetermined value.

The following result can be shown by induction over the loop size. To simplify the notation and w.l.o.g. we take $n = 1$ and $d = N$.

Lemma 3 *Let $c_1, \ldots, c_{N-1}$ be the* RLWE *encryptions of $m_1(X), \cdots, m_{N-1}(X)$ respectively. Algorithm 1 takes as inputs the c_i's and returns a* RLWE *encryption of $\mu(X) = \sum_i m_i(X)[0] \cdot X^j$.*

Proof Let $i \in [0, \log N)$. For $\kappa \in \left[0, N/2^{i+1}\right)$ We denote $\mu[\kappa](X)$ the message corresponding to the κ-th entry in the ciphertexts hashmap at the end of the i-th iteration. We denote $\mu(X)$ instead of $\mu[0](X)$ the message corresponding to the ciphertext outputted at the end of the last iteration. We will show by induction that $\mu[\kappa]$ at the end of the i-th iteration can be written as:

$$\mu[\kappa]\left[j(N/2^{i+1})\right] = \frac{2^i}{N} m_{\kappa + j\frac{N}{2^{i+1}}}[0]$$

$\forall j \in \left[0, 2^{i+1}\right)$ and $\forall \kappa \in \left[0, N/2^{i+1}\right)$.

At the end of the first iteration ($i = 0$), we have $t_0 = N/2$ ciphertexts and $\frac{N}{2t_0} = 1$, $j = 0$. Let $\kappa \in [0, N/2)$. Applying 2 with $\mu(X)$ and $\mu'(X)$ such that:

$$\mu(X)[0] = \frac{1}{N} \cdot m_\kappa(X)[0] \quad \text{and} \quad \mu'(X)[0] = \frac{1}{N} \cdot m_{\kappa + N/2}(X)[0]$$

At the i-th $= 0$-th iteration of Algorithm 1, the κ-th ciphertext can be written as:

$$\begin{aligned} \mu[\kappa]^{(1)}(X) &= 2 \cdot \left(\mu(X) + \mu'(X) \cdot X^{N/2}\right) \\ &= 2 \cdot \left(\frac{1}{N} m_\kappa(X)[0] + \frac{1}{N} m_{\kappa + \frac{N}{2}}(X)[0] \cdot X^{N/2} + \star\right) \end{aligned} \tag{2}$$

And we obtain:

$$\mu[\kappa]\left[j\frac{N}{2}\right] = \frac{2}{N} \cdot m_\kappa[0] + j\frac{N}{2}$$

for $j \in [0, 2)$ and $\forall \kappa \in [0, N/2)$. Let $\kappa \in \left[0, \frac{N}{2^i}\right)$. Following Lemma 2 and focusing on indices that are multiple of t_i, at the end of the i-th iteration, there exist $\mu(X)$ and $\mu(X)'$ such that for the κ-th ciphertext encrypts a message $\mu^{i+1}[\kappa]$ such that for $j' \in [0, 2^i)$:

$$\mu^{(i+1)}[\kappa](X) = 2 \sum_{j'=0}^{N/2t_i - 1} (\mu_{2j't_i} + \mu'_{2j't_i} X^{t_i}) \cdot X^{2j't_i}$$

We set $\mu(X)$ and $\mu'(X)$ such that for all $j' \in \left[0, 2^i\right)$:

$$\begin{aligned} \mu(X)\left[j'(N/2^i)\right] &= \mu^{(i)}[\kappa](X)\left[j'(N/2^i)\right] \quad \text{if } j' \text{ is even} \\ \mu'(X)\left[j'(N/2^i)\right] &= \mu^{(i)}[\kappa](X)\left[(j'(N/2^i)\right] \quad \text{if } j' \text{ is odd} \end{aligned} \tag{3}$$

By the induction assumption, at the end of the $(i-1)$-th iteration, with the observation that $2j'\frac{N}{2^{i+1}} = j'\frac{N}{2^i}$, we have for all $j' \in \left[0, 2^i\right)$:

$$\mu^{(i)}[\kappa](X)\left[j'(N/2^i)\right] = \frac{2^i}{N} m_{\kappa + j'\frac{N}{2^i}}[0]$$

Setting $j = 2j'$, we have for all $j \in \left[0, 2^{i+1}\right)$:

$$\mu^{(i+1)}[\kappa](X)\left[j(N/2^{i+1})\right] = \frac{2^{i+1}}{N} m_{\kappa + j\frac{N}{2^{i+1}}}[0]$$

We finally obtain:

$$\mu^{(i+1)}[\kappa][jt_i] = \frac{2^{i+1}}{N} m_{\kappa + j\frac{N}{2^{i+1}}}[0]$$

for $j \in \left[0, 2^{i+1}\right)$.

After the $(\log N - 1)$-th iteration, (after the last iteration), $i = \log N - 1$, $t_{\log N - 1} = \frac{N}{2^{\log N}} = 1$ and $\kappa = 0$, there is one ciphertext such that for all $j \in \left[0, 2^{\log N - 1 + 1} = N\right)$:

$$\mu^{(\log N)}[\kappa][j] = \frac{2^{\log N}}{N} m_{\kappa + j}[0]$$

for $j \in \left[0, 2^{\log N}\right)$.

□

Lemma 4 (Complexity of Algorithm 1) *If Algorithm 1 takes N/n RLWE ciphertexts as input, it requires $2^{\log N - \log n} - 1$ automorphism evaluations and $\log N - \log n$ distinct switching keys.*

Lemma 5 (Noise growth in Algorithm 1) *Let ϑ_{ct} be the variance of the noise of the initial ciphertext and ϑ_{ks} be the variance of the noise of the key switching operation, the final noise variance is actually $\approx \vartheta_{ct} + (N/n)\vartheta_{ks}$.*

Proof The noise behavior in Algorithm 1 is predictable and can be easily explained. In each of the $0 \leq i < \log(N/n)$ steps of the algorithm, the $X^{j(N/2^{i+1})}$ ciphertext coefficients double when j is even and vanish when j is odd. This observation easily follows from the proof of Lemma 2.

Additionally, one of the ciphertexts goes through a key switching operation. Therefore, given ϑ_{ct} the noise output of the previous step and e_{ks} the noise of the key switching operation, the new noise in the $X^{i(N/2^j)}$ coefficients after each iteration becomes $2\vartheta_{\mathsf{ct}} + \vartheta_{\mathsf{ks}}$. Over $\log(N/n)$ steps, this gives us an estimated noise growth of:

$$\begin{aligned} 2(\dots 2(2\frac{n}{N}\vartheta_{\mathsf{ct}} + \vartheta_{\mathsf{ks}}) + \vartheta_{\mathsf{ks}} \dots) + \vartheta_{\mathsf{ks}} &= 2^{\log \frac{N}{n}} \frac{n}{N}\vartheta_{\mathsf{ct}} + \sum_{i=0}^{\log(N/n)-1} 2^i \vartheta_{\mathsf{ks}} \\ &= \vartheta_{\mathsf{ct}} + (2^{\log \frac{N}{n}} - 1)\vartheta_{\mathsf{ks}} \\ &\approx \vartheta_{\mathsf{ct}} + \frac{N}{n}\vartheta_{\mathsf{ks}} \end{aligned}$$

The noise growth is therefore independent of the initial noise and depends solely on the additive noise of the key switching operation, which can be parameterized through the gadget decomposition to achieve the desired bounds. □

4 Large Domain Private-Function Evaluation

In this section, we consider a two-party protocol where a server holds a function defined on a large domain, $f : \mathsf{Dom}_f \mapsto \mathsf{Img}_f$ and a client holds a set of points. As an

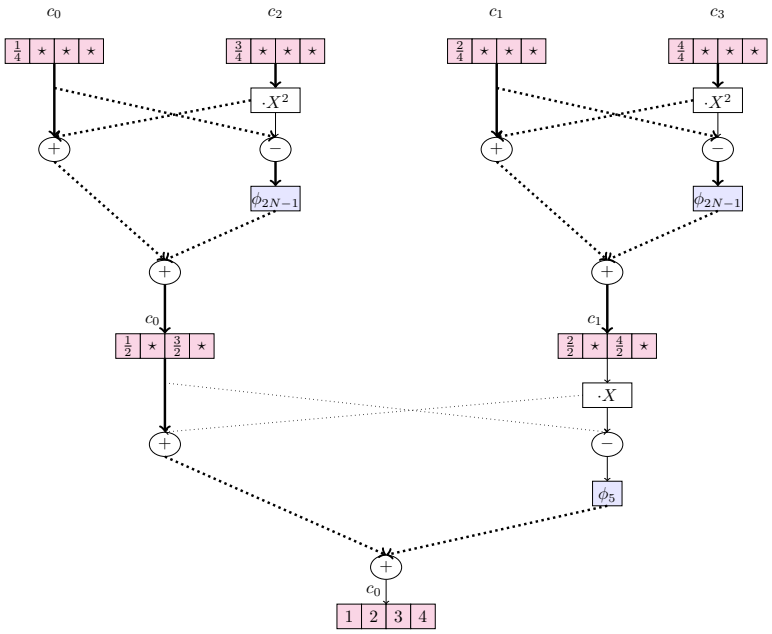

Fig. 2: Illustrated example of Algorithm 1 for $d = 4$ with input ciphertexts (c_0, c_1, c_2, c_3) which are RLWE encryptions of $m_i(X)$ such that $m_i[0] = i + 1$. The ciphertexts are assumed to have been pre-scaled by N^{-1} and are given in bit-reversed order for the sake of clarity. At the initial step ($t_0 = \frac{N}{2^1}$), the input ciphertexts are divided into two groups, those with odd indices (c_1 and c_3) are multiplied by $X^{N/t} = X^2$ and combined with those of even indices (c_0 and c_2) which results in two ciphertexts taken as input to the second step ($t_1 = \frac{N}{2^2}$). The same process is applied and results in a single output ciphertext. The symbols $\star$ inside the ciphertexts represent unknown values.

example, the function f can be a Heatmap function as in [24], asked by a client and processed on a large dataset. In the end, the client receives the evaluation of f in a target subset I of elements $\mathbb{R}_{[a,b]} \subseteq \mathsf{Dom}_f$ taken from the domain of f such that the following privacy constraints are supported:

- The server does not learn which indices were requested by the client.
- The client does not learn more about the function than the result of the function evaluated on the set of selected points.

4.1 Large Domain Function Evaluation

The solution we propose for the problem of evaluating a function with high precision works as follows:

Step 1. The user encrypts $c_i = \mathsf{RLWE}_s(X^i)$ for each targeted element $i \in \mathcal{I}$ and sends the ciphertexts to the server.

Step 2. The server defines a polynomial representation of the function f to be evaluated as follows:

$$u_f := f(0) - f(N-1)X - f(N-2)X^2 - \cdots - f(1)X^{N-1}.$$

For each ciphertext sent by the client, we have:

$$c_i \cdot u_f = \mathsf{RLWE}_s(X^i) \cdot u_f = \mathsf{RLWE}_s(f(i)X^0 + \star)$$

The server obtains $|\mathcal{I}|$ RLWE ciphertexts encrypting each message polynomial whose constant coefficient is equal to $f(i)$.

The server then applies the ring repacking from Algorithm 1 over the $|\mathcal{I}|$ ciphertexts to compress the response. It obtains $\mathsf{RLWE}_s(f(i_0) + f(i_1)X + \cdots + f(i_{|\mathcal{I}|-1})X^{|\mathcal{I}|-1})$ which is sent to the client.

In total, the server performs $2|\mathcal{I}|$ homomorphic plaintext and ciphertext multiplications and $|\mathcal{I}|$ automorphism evaluations and requires $\log N$ distinct key-switching keys for the repacking step.

Step 3. The user decrypts and retrieves $f(i_0) + f(i_1)X + \cdots + f(i_{|\mathcal{I}|-1})X^{|\mathcal{I}|-1}$ which gives the evaluation of $f(i_z)$ for each i_z asked to the server. Then it obtains the consecutive input evaluation requested from the server. Depending on the scenario, the server can alternatively perform the extraction on his side before sending the result so that no parsing would be needed on the user side.

Remark 1 In the above solution, the server computes the RLWE encryption of a message whose coefficient appears with a query-dependent ordering. In order to mask the ordering of the computations, the server can shuffle the LWE ciphertexts obtained after Step 1 and then apply the repacking over the shuffled ciphertexts. Upon receiving the calculation from the server, the user decrypts and rearranges the decrypted value by grouping the elements that match.

4.2 Improved Split Domain Approach through ring repacking

4.2.1 Review of the Split Domain Approach.

In the *split domain* approach of [24], in order to perform the evaluation of an arbitrary function f on the ciphertexts c_i's, the server computes $\sum_i \mathsf{coeff}(c_i) \cdot u$ where $u \in \mathbb{Z}_1^{N \times N}$ is a matrix representation of f (the i-th row being $\mathsf{coeff}(X^{f(i)})$, that is, the one hot encoding vector of $f(i)$). The original split domain approach of [24] expresses a RLWE ciphertext input $(a, b) \in \mathsf{R}_q^2$, as a structured set of N $\mathsf{LWE}_{\vec{\mathbf{s}}}(m(X)[i])$ ciphertexts $(\mathsf{coeff}(X^i \cdot a_i), b[i])$. This matrix can be expressed as $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{N \times N, 1 \times N}$ where $\mathbf{A} \in \mathbb{Z}_q^{N \times N}$ is the anti-circulant Vandermond matrix representation of a and $\mathbf{b} = \mathsf{coeff}(b)$. They make use of this expression of the ciphertext $\mathsf{RLWE}_s(m(X))$ as

$(\mathbf{A}, \mathbf{b})$ and multiply it by the matrix u representing the function f to evaluate. However, the resulting ciphertext $(\mathbf{A}u, \mathbf{b}u) \in \mathbb{Z}_q^{N \times N, 1 \times N}$ is not a valid RLWE ciphertext because $\mathbf{A}u$ is no longer an anti-circulant Vandermond matrix and thus cannot be collapsed back into an element of R_q. To fix the format of the ciphertext, they need a *format fixing* key which is composed of a set of N switching keys, the format fixing procedure being equivalent to the evaluation of N key switches, one for each row of the matrix $\mathbf{A}u$.

4.2.2 Optimized Split Domain Approach.

Omitting the action of u on $\mathbf{A}$, observe that the action of u on $\mathbf{b}$ can be expressed with three well-defined homomorphic operations: addition, scalar multiplication, and permutation. Therefore if we first perform a coefficient extraction on the ciphertext $\mathsf{RLWE}_s(m(X)) = (a, b) \in \mathsf{R}_q^2$ for each of its N coefficients, the evaluation can now be seen as:

$$(\mathsf{LWE}_{\tilde{\mathbf{s}}}(\mathsf{coeff}(X^0 \cdot a), b[0]), \ldots, \mathsf{LWE}_{\tilde{\mathbf{s}}}(\mathsf{coeff}(X^{N-1} \cdot a), b[N-1])) \cdot u$$

which is actually a new set of N LWE ciphertexts of the form $\mathsf{LWE}_{\tilde{\mathbf{s}}}(b'[i])$. Since an $\mathsf{LWE}_{\tilde{\mathbf{s}}}(m)$ ciphertext $(\mathbf{a}, b) \in \mathbb{Z}_q^{N+1}$ can be expressed as a RLWE_s ciphertext whose constant coefficient decrypts to m by setting $(\mathbf{a}, (b, 0, \ldots, 0)) \in \mathbb{Z}_q^{2N}$ and taking it as an element of R_q, we can then use Algorithm 1 to pack the d LWE ciphertexts into a single N dimensional RLWE ciphertext. In that case, we take the number of RLWE inputs d equal to N. Algorithm 1 requires at most $d-1$ automorphism evaluations and only $\log d$ switching keys, which is an asymptotic improvement over the d switching keys required by the original approach. In addition, using our new repacking allows to reduce the memory consumption from $O(d)$ to $O(\log d)$ ciphertexts.

4.2.3 Larger Domains

Domain sizes larger than N can be handled in the same way as in [24]: if $k := \lceil \frac{\mathsf{Dom}(f)}{N} \rceil$, $\mathsf{Dom}(f)$ can be written as a union of disjoint sets D_i, that is, $\mathsf{Dom}(f) = \cup_{i=1}^{k} D_i$. Then, in the first step, the client computes k ciphertexts for each input, encrypting $X^{j \mod N}$ or 0 depending on whether the input j belongs to D_i.

4.3 Query, response sizes, computation complexity and privacy

The following subsections provide a detailed comparison of how the client encryption and key generation, server evaluation, and client decryption steps of our approach differ from the original split domain approach. We summarize these differences in Table 1.

4.3.1 Client Key materials and Encryption.

Given a target set $\mathcal{I}$, the original split domain approach, as ours, encrypts each input $i \in \mathcal{I}$ as $\mathsf{Enc}(X^i)$. Similarly, up to $k = \lceil |\mathsf{Dom}(f)|/N \rceil$ ciphertexts per input are required. Our approach only requires $\mathcal{O}(\log d)$ automorphism keys, instead of $\mathcal{O}(N)$, with $d \leq N$. We will see next that this results in a total size of public key material of

only 1.5MB for the considered cryptographic parameters (which are sufficient for all the considered sizes $|\mathsf{Dom}(f)|$ and $|\mathsf{Img}(f)|$).

4.3.2 Server Evaluation in the original Split Domain Approach.

The original solution aims to retrieve the result as $X^{f(i)}$. To do so, the server performs the evaluation on each input in two distinct steps. As explained above, each ciphertext $\mathsf{RLWE}(X^{f(i)})$ needs to be expanded into its LWE $N \times (N+1)$ matrix representation. This significantly increases the memory footprint of each input with respect to the ring degree N from $\mathcal{O}(N)$ to $\mathcal{O}(N^2)$. The server then evaluates k $N \times N$ matrix products between the expanded ciphertext and the sparse matrix representation of the function f, which requires $O(kN^2)$ additions/multiplications in $\mathbb{Z}_q$ per input. The server then has to evaluate N key-switching operations to fix the format of the ciphertext and retrieve a valid RLWE ciphertext, which requires $\mathcal{O}(N)$ key-switching operations per input. Since the results are stored in the exponent, the original solution requires that $N \geq |\mathsf{Img}(f)|$, so the response size is $\mathcal{O}(|\mathsf{Img}(f)|)$ for a single input. However, each coefficient can store the count of $\mathcal{O}(q/N)$ [1] inputs. Therefore, the response size for a set of d inputs is $\mathcal{O}(\lceil dN/q \rceil |\mathsf{Img}(f)|)$.

4.3.3 Server Evaluation in our approach.

Our solution aims to retrieve the result as $f(i)X^0$ instead. This considerably simplifies server evaluation as we can simply encode f on elements of R_q and perform each input evaluation as k plaintext multiplications in R_q. This enables to keep the memory footprint with respect to the ring degree N to $\mathcal{O}(N)$ and the evaluation complexity per input is $\mathcal{O}(kN)$ operations in $\mathbb{Z}_q$ since elements of R_q are by default in the NTT domain. Finally, we can repack a set of $d \leq N$ RLWE inputs into a single RLWE ciphertext using Algorithm 1. The complexity of such repacking is $d-1$ automorphisms for d RLWE input ciphertexts, which is amortized by $\mathcal{O}(1)$ key-switching per input.

This enables reducing the data complexity of the response from $\mathcal{O}(dN \log |\mathsf{Img}(f)|)$ to $\mathcal{O}(\lceil d/N \rceil N \log |\mathsf{Img}(f)|)$. This provides an interesting trade-off with respect to the original solution. In fact, in our solution N is $\mathcal{O}(\log |\mathsf{Img}(f)|)$ however d is divided by N instead of q, with $\log q$ being $\mathcal{O}(N)$. The result is that the original solution provides a more compact response when $|\mathsf{Img}(f)|$ is small and N is very large, while our solution provides a more compact response than the original solution when $|\mathsf{Img}(f)|$ is large and $N \leq |\mathsf{Img}(f)|$.

4.3.4 Multivariate functions.

If the function is of the form $g = \sum \alpha_i f_i$, then the server combines the different f_i using automorphisms. Note that since α_i must be an element $\mathbb{Z}^\star_{2N}$ for the automorphism to be well defined, it is necessarily odd, which limits the class of functions that can be evaluated. Although [33] proposed several solutions to handle even α (either by setting α to the next/previous odd integer or by using auxiliary blind rotation keys),

[1] For a fixed error and secret distribution, the dominant factor of the noise growth is the ring expansion factor, which is N in our case.

	Query Size	Switching Keys	Evaluation	Response Size
Original [24]	$\mathcal{O}(dkN)$	$\mathcal{O}(N)$	$\mathcal{O}_{\mathbb{Z}_q}(dkN^2) + \mathcal{O}_{\text{KeySwith}}(dN)$	$\mathcal{O}(\lceil dN/q \rceil \lvert \mathsf{Img}(f) \rvert)$
Ours	$\mathcal{O}(dkN)$	$\mathcal{O}(\log N)$	$\mathcal{O}_{\mathbb{Z}_q}(dkN) + \mathcal{O}_{\text{KeySwith}}(d)$	$\mathcal{O}(\lceil d/N \rceil N \log \lvert \mathsf{Img}(f) \rvert)$

Table 1: Comparison summary of our *split domain* approach with the original *split domain* for a batch evaluation of d inputs. We let $k = |\mathsf{Dom}(f)|/N$, where N is the ring degree of R and q is the ciphertext modulus.

they are all specific to homomorphic decryption of an LWE sample, which can tolerate error, and thus cannot be translated to this use case.

Since $f(i)$ appears in the coefficient of each monomial and not in the exponent as in [24], the server only needs to perform a homomorphic scalar product between each α_i and the f_i evaluation obtained to obtain the combined result. We can thus combine the intermediate results with addition and scalar multiplications between repacked ciphertexts. This step does not require evaluation keys and is essentially free with respect to the previous computation. Note that, contrary to the approach that requires automorphisms to combine results, ours is not limited to odd α_i.

4.3.5 Circuit Privacy

By default, the function-evaluations described in Section 4.1 and Section 4.2 do not offer circuit privacy. However, the server knows the distribution of u_f and e (the noise of $\mathsf{RLWE}_s(X^i)$), and therefore the distribution of $u_f e$, which is the part that needs to be hidden from the client. As such, following the technique proposed by Castro et al. [34], it first re-randomize the ciphertexts before the repacking step and then perform a modulus switching to hide $u_f e$ after the repacking step, before sending the result back to the client.

5 Performances

$\mathsf{Dom}(f)$	$\mathsf{Img}(f)$	Encryption [sec]	Query [MB]	Evaluation [sec]	Response [KB]	Keys [MB]
$\mathbb{Z}_{2^{12}}$	$\mathbb{Z}_{2^{12}}$	0.64	129	0.219		
$\mathbb{Z}_{2^{13}}$	$\mathbb{Z}_{2^{13}}$	1.25	258	0.240		
$\mathbb{Z}_{2^{14}}$	$\mathbb{Z}_{2^{14}}$	2.46	516	0.268	32	1.5
$\mathbb{Z}_{2^{15}}$	$\mathbb{Z}_{2^{15}}$	4.92	1033	0.320		
$\mathbb{Z}_{2^{16}}$	$\mathbb{Z}_{2^{16}}$	9.95	2066	0.487		

Table 2: Performance of our *Split-Domain* approach for an univariate function $f(x)$. Timings and data size are reported for batches of 2048 inputs.

In this section, we empirically evaluate the performance of our strategy. We implemented it using the Lattigo library [29]. The code can be found in the following repository https://github.com/Pro7ech/fhe-org-2024/tree/main/large-domain. All benchmarks were conducted on the following hardware: Go 1.21, Windows 11, i9-12900K, 32GB DDR4, single threaded.

All experiments make use of the same cryptographic parameters, which provide a security of 128-bit: $N = 2^{11}$, $\log_2(Q) = 54$, $\sigma_e = 3.2$ (0-centered discrete Gaussian with standard deviation 3.2), $\sigma_s = \sqrt{2/3}$ (uniform ternary distribution). We performed two experiments, one for an univariate function $f : \mathbb{Z}_t \mapsto \mathbb{Z}_t$ and one for a bivariate function $g : \mathbb{Z}_t \times \mathbb{Z}_t \mapsto \mathbb{Z}_t$ of the form $\alpha_1 f_1 + \alpha_2 f_2$. The results of our experiments can be found in Table 2 and Table 3. We observe that in both cases, the client encryption time and query size are linear with $|\mathsf{Dom}(f)|$, which is expected. In fact, by improving the performance on the server side, the client side is now the dominant cost of the computation, and how to improve this aspect is an open question that we leave for future work.

Regarding server evaluation time, our solution shows an improvement of three-orders-of-magnitude for both univariate and bivariate functions over the original *split domain* approach of [24]. As an example, the server can evaluate $g(x, y) = \alpha_1 f_1(x) + \alpha_2 f_2(y)$ with $g(x, y), x, y \in \mathbb{Z}_{2^{15}}$ in 0.783 seconds for a batch of 2048 inputs, while the original *split domain* approach requires 0.820 seconds per input, which gives 2000 higher throughput improvement per input.

Finally, the response size is 32KB for all cases, which is in fact the size of a ciphertext for the cryptographic parameters used. This provides an amortized size of 16 bytes per input, which is reasonably small in the context of homomorphic encryption. In fact, depending on the ratio ciphertext-modulus/plaintext-modulus q/t, the size of the response could be further reduced by truncating the lower bits of each coefficient of the ciphertext, as all that is required during the decryption is that the ratio q/t is large enough to store the decryption error.

$\mathsf{Dom}(g)$	$\mathsf{Img}(g)$	Encryption [sec]	Query [MB]	Eval [sec]	EvalPriv [sec]	Response [KB]	Keys [MB]
$\mathbb{Z}_{2^{12}} \times \mathbb{Z}_{2^{12}}$	$\mathbb{Z}_{2^{12}}$	0.658	258	0.255	0.742		
$\mathbb{Z}_{2^{13}} \times \mathbb{Z}_{2^{13}}$	$\mathbb{Z}_{2^{13}}$	1.25	516	0.283	0.784		
$\mathbb{Z}_{2^{14}} \times \mathbb{Z}_{2^{14}}$	$\mathbb{Z}_{2^{14}}$	2.42	1033	0.343	1.23	32	1.5
$\mathbb{Z}_{2^{15}} \times \mathbb{Z}_{2^{15}}$	$\mathbb{Z}_{2^{15}}$	6.31	2066	0.783	1.32		
$\mathbb{Z}_{2^{16}} \times \mathbb{Z}_{2^{16}}$	$\mathbb{Z}_{2^{16}}$	15.02	4133	1.15	2.10		

Table 3: Performance of our *Split-Domain* approach for a bivariate function $g(x, y) = \alpha_1 f_1(x) + \alpha_2 f_2(y)$. Timings and data size are reported for batches of 2048 inputs. EvalPriv refers to a circuit private evaluation, where the ciphertexts are re-randomized (by adding an encryption of zero) before the repacking step, and then quantized after the repacking step.

In this Section, we have considered that the function is evaluated over encrypted inputs but remains known to the server through the evaluation process. If the client

would like to hide the function instead, which happens when the function holds valuable intellectual properties, the process works exactly the same, except that the test vector is encrypted and the computation is processed over cleartext data.

6 Conclusion

This paper presents a very efficient strategy for evaluating arbitrary functions over encrypted inputs defined over a large domain. For this target task, our work significantly improves the previous *split domain* approach from [24] by three orders of magnitude, reducing the evaluation key size from $O(N)$ to $O(logN)$ through the use of ring repacking. To illustrate the comparison with [24], we consider a two-party protocol in which a client sends encrypted points lying in a large domain to the server; the latter returns the homomorphic evaluation of an arbitrary function over these points. Still, we believe that our strategy could be useful in many other privacy-enhancing use-cases. Finally, as an independent contribution, we consolidate previous work on the ring (re-)packing algorithm from [28, 32] by providing a complete proof of the correctness of all the steps of the repacking algorithm in its iterative form.

Appendix

Proof of Lemma 2

Proof Let $\mu(X) = \sum \mu_j X^j$ and $\mu'(X) = \sum \mu'_j X^j$ and let some $i \in [0, \log N)$, $t = N/2^{i+1}$ and $\alpha = (1, 0)$ if $i = 0$ else $\alpha = (0, 2^{i-1})$. Using Lemma 1, we only focus on the coefficients of $\mu(X)$ and $\mu'(X)$ whose index is a multiple of t, thus we have:

$$
\begin{aligned}
&= \sum_{j=0}^{N/t-1} \mu_{jt} \cdot X^{jt} + \sum_{j=0}^{N/t-1} (\mu'_{jt} \cdot X^{jt}) \cdot X^t \\
&+ \phi \left(\sum_{j=0}^{N/t-1} \mu_{jt} \cdot X^{jt} - \left(\sum_{j=0}^{N/t-1} \mu'_{jt} \cdot X^{jt} \right) \cdot X^t \right) \\
&= \sum_{j=0}^{N/t-1} \mu_{jt} \cdot X^{jt} + \sum_{j=0}^{N/t-1} (\mu'_{jt} \cdot X^{jt}) \cdot X^t \\
&\quad + \phi \left(\sum_{j=0}^{N/t-1} \mu_{jt} \cdot X^{jt} \right) - \phi \left(\sum_{j=0}^{N/t-1} \mu'_{jt} \cdot X^{jt} \right) \cdot \phi(X^t) \\
&= \sum_{j=0}^{N/t-1} \mu_{jt} \cdot X^{jt} + \sum_{j=0}^{N/t-1} (\mu'_{jt} \cdot X^{jt}) \cdot X^t \\
&+ \sum_{j=0}^{N/t-1} (-1)^j \mu_{jt} \cdot X^{jt} + \sum_{j=0}^{N/t-1} ((-1)^j \mu'_{jt} \cdot X^{jt}) \cdot X^t
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{j=0}^{N/t-1} \mu_{jt}(-1^j+1)\cdot X^{jt} + \sum_{j=0}^{N/t-1} \mu'_{jt}((-1^j+1)\cdot X^{jt})\cdot X^t \\
&= 2\sum_{j=0}^{N/2t-1} \mu_{2it}\cdot X^{2jt} + 2\sum_{j=0}^{N/2t} (\mu_{2jt}\cdot X^{2jt})\cdot X^t \\
&= 2\sum_{j=0}^{N/2t-1} (\mu_{2jt}+\mu'_{2jt}\cdot X^t)\cdot X^{2jt}
\end{aligned}
$$

□

References

[1] Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 617–640. Springer, Sofia, Bulgaria (2015). https://doi.org/10.1007/978-3-662-46800-5_24

[2] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 3–33. Springer, Hanoi, Vietnam (2016). https://doi.org/10.1007/978-3-662-53887-6_1

[3] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 377–408. Springer, Hong Kong, China (2017). https://doi.org/10.1007/978-3-319-70694-8_14

[4] Biasse, J.-F., Ruiz, L.: FHEW with efficient multibit bootstrapping. In: Lauter, K.E., Rodríguez-Henríquez, F. (eds.) Progress in Cryptology - LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America. Lecture Notes in Computer Science, vol. 9230, pp. 119–135. Springer, Guadalajara, Mexico (2015). https://doi.org/10.1007/978-3-319-22174-8_7

[5] Bonnoron, G., Ducas, L., Fillinger, M.: Large FHE gates from tensored homomorphic accumulator. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 18: 10th International Conference on Cryptology in Africa. Lecture Notes in Computer Science, vol. 10831, pp. 217–251. Springer, Marrakesh, Morocco (2018). https://doi.org/10.1007/978-3-319-89339-6_13

[6] Carpov, S., Izabachène, M., Mollimard, V.: New techniques for multi-value input homomorphic evaluation and applications. In: Matsui, M. (ed.) Topics in Cryptology – CT-RSA 2019. Lecture Notes in Computer Science, vol. 11405, pp.

106–126. Springer, San Francisco, CA, USA (2019). https://doi.org/10.1007/978-3-030-12612-4_6

[7] Boura, C., Gama, N., Georgieva, M., Jetchev, D.: CHIMERA: combining ring-lwe-based fully homomorphic encryption schemes. J. Math. Cryptol. **14**(1), 316–338 (2020)

[8] Bourse, F., Sanders, O., Traoré, J.: Improved secure integer comparison via homomorphic encryption. In: Jarecki, S. (ed.) Topics in Cryptology – CT-RSA 2020. Lecture Notes in Computer Science, vol. 12006, pp. 391–416. Springer, San Francisco, CA, USA (2020). https://doi.org/10.1007/978-3-030-40186-3_17

[9] Guimarães, A., Borin, E., Aranha, D.F.: Revisiting the functional bootstrap in TFHE. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(2), 229–253 (2021) https://doi.org/10.46586/tches.v2021.i2.229-253

[10] Lee, K., Yoon, J.W.: Discretization error reduction for high precision torus fully homomorphic encryption. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part II. Lecture Notes in Computer Science, vol. 13941, pp. 33–62. Springer, Atlanta, GA, USA (2023). https://doi.org/10.1007/978-3-031-31371-4_2

[11] Kluczniak, K., Schild, L.: FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(1), 501–537 (2023) https://doi.org/10.46586/tches.v2023.i1.501-537

[12] Liu, Z., Micciancio, D., Polyakov, Y.: Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022, Part II. Lecture Notes in Computer Science, vol. 13792, pp. 130–160. Springer, Taipei, Taiwan (2022). https://doi.org/10.1007/978-3-031-22966-4_5

[13] Bergerat, L., Boudi, A., Bourgerie, Q., Chillotti, I., Ligier, D., Orfila, J.-B., Tap, S.: Parameter optimization and larger precision for (T)FHE. Journal of Cryptology **36**(3), 28 (2023) https://doi.org/10.1007/s00145-023-09463-5

[14] Yang, Z., Xie, X., Shen, H., Chen, S., Zhou, J.: TOTA: Fully Homomorphic Encryption with Smaller Parameters and Stronger Security. Cryptology ePrint Archive, Report 2021/1347 (2021). https://eprint.iacr.org/2021/1347

[15] Chillotti, I., Ligier, D., Orfila, J.-B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021, Part III. Lecture Notes in Computer Science, vol. 13092, pp. 670–699. Springer, Singapore (2021). https://doi.org/10.1007/978-3-030-92078-4_23

[16] Chartier, P., Koskas, M., Lemou, M., Méhats, F.: Homomorphic sign evaluation with a RNS representation of integers. In: Chung, K.-M., Sasaki, Y. (eds.) Advances in Cryptology – ASIACRYPT 2024, Part I. Lecture Notes in Computer Science, vol. 15484, pp. 271–296. Springer, Kolkata, India (2024). https://doi.org/10.1007/978-981-96-0875-1_9

[17] Drucker, N., Moshkowich, G., Pelleg, T., Shaul, H.: BLEACH: Cleaning errors in discrete computations over CKKS. Journal of Cryptology **37**(1), 3 (2024) https://doi.org/10.1007/s00145-023-09483-1

[18] Bae, Y., Cheon, J.H., Kim, J., Stehlé, D.: Bootstrapping bits with CKKS. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024, Part II. Lecture Notes in Computer Science, vol. 14652, pp. 94–123. Springer, Zurich, Switzerland (2024). https://doi.org/10.1007/978-3-031-58723-8_4

[19] Bae, Y., Kim, J., Stehlé, D., Suvanto, E.: Bootstrapping small integers with CKKS. In: Chung, K.-M., Sasaki, Y. (eds.) Advances in Cryptology – ASIACRYPT 2024, Part I. Lecture Notes in Computer Science, vol. 15484, pp. 330–360. Springer, Kolkata, India (2024). https://doi.org/10.1007/978-981-96-0875-1_11

[20] Geelen, R., Vercauteren, F.: Fully Homomorphic Encryption for Cyclotomic Prime Moduli. Cryptology ePrint Archive, Report 2024/1587 (2024). https://eprint.iacr.org/2024/1587

[21] Kim, J., Noh, T.: Modular Reduction in CKKS. Cryptology ePrint Archive, Report 2024/1638 (2024). https://eprint.iacr.org/2024/1638

[22] Boneh, D., Kim, J.: Homomorphic Encryption for Large Integers from Nested Residue Number Systems. Cryptology ePrint Archive, Report 2025/346 (2025). https://eprint.iacr.org/2025/346

[23] Chung, H., Kim, H., Kim, Y.-S., Lee, Y.: Amortized Large Look-up Table Evaluation with Multivariate Polynomials for Homomorphic Encryption. Cryptology ePrint Archive, Report 2024/274 (2024). https://eprint.iacr.org/2024/274

[24] Iliashenko, I., Izabachène, M., Mertens, A., Pereira, H.V.L.: Homomorphically counting elements with the same property. Proceedings on Privacy Enhancing Technologies **2022**(4), 670–683 (2022) https://doi.org/10.56553/popets-2022-0127

[25] Halevi, S., Shoup, V.: Algorithms in HElib. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 554–571. Springer, Santa Barbara, CA, USA (2014). https://doi.org/10.1007/978-3-662-44371-2_31

[26] Bae, Y., Cheon, J.H., Kim, J., Park, J.H., Stehlé, D.: HERMES: Efficient ring

packing using MLWE ciphertexts and application to transciphering. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part IV. Lecture Notes in Computer Science, vol. 14084, pp. 37–69. Springer, Santa Barbara, CA, USA (2023). https://doi.org/10.1007/978-3-031-38551-3_2

[27] Lu, W.-j., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: 2021 IEEE Symposium on Security and Privacy, pp. 1057–1073. IEEE Computer Society Press, San Francisco, CA, USA (2021). https://doi.org/10.1109/SP40001.2021.00043

[28] Chen, H., Dai, W., Kim, M., Song, Y.: Efficient homomorphic conversion between (ring) LWE ciphertexts. In: Sako, K., Tippenhauer, N.O. (eds.) ACNS 21: 19th International Conference on Applied Cryptography and Network Security, Part I. Lecture Notes in Computer Science, vol. 12726, pp. 460–479. Springer, Kamakura, Japan (2021). https://doi.org/10.1007/978-3-030-78372-3_18

[29] EPFL-LDS, T.I.S.: Lattigo v5. Online: https://github.com/tuneinsight/lattigo (2023)

[30] Izabachène, M., Bossuat, J.: TETRIS: composing FHE techniques for private functional exploration over large datasets. Proc. Priv. Enhancing Technol. **2025**(2), 23–38 (2025) https://doi.org/10.56553/POPETS-2025-0047

[31] Han, K., Ki, D.: Better bootstrapping for approximate homomorphic encryption. In: Jarecki, S. (ed.) Topics in Cryptology – CT-RSA 2020. Lecture Notes in Computer Science, vol. 12006, pp. 364–390. Springer, San Francisco, CA, USA (2020). https://doi.org/10.1007/978-3-030-40186-3_16

[32] Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., Yoo, D.: General bootstrapping approach for RLWE-based homomorphic encryption. IACR Cryptol. ePrint Arch. **2021**, 691 (2021)

[33] Lee, Y., Micciancio, D., Kim, A., Choi, R., Deryabin, M., Eom, J., Yoo, D.: Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption, pp. 227–256 (2023). https://doi.org/10.1007/978-3-031-30620-4_8

[34] Castro, L., Juvekar, C., Vaikuntanathan, V.: Fast Vector Oblivious Linear Evaluation from Ring Learning with Errors. WAHC@ACM (2021). https://doi.org/10.1145/3474366.3486928 . https://eprint.iacr.org/2020/685