Homomorphically counting elements with the same property

Ilia Iliashenko[1], Malika Izabachène[2], Axel Mertens[3], and Hilder V. L. Pereira[3]

[1] Ciphermode Labs
[2] Cosmian
[3] imec-COSIC, KU Leuven, Leuven, Belgium.
ilia.iliashenko@kuleuven.be, malika.izabachene@cosmian.com, axel.mertens@kuleuven.be, hildervitor.limapereira@kuleuven.be

Abstract. We propose homomorphic algorithms for privacy-preserving applications where we are given an encrypted dataset and we want to compute the number of elements that share a common property. We consider a two party scenario between a client and a server, where the storage and computation is outsourced to the server. We present two new efficient methods to solve this problem by homomorphically evaluating a selection function encoding the desired property, and counting the number of elements which evaluates to the same value. Our first method programs the homomorphic computation in the style of the the functional bootstrapping of TFHE and can be instantiated with essentially any homomorphic encryption scheme that operates on polynomials, like FV or BGV. Our second method relies on new homomorphic operations and ciphertext formats, and it is more suitable for applications where the number of possible inputs is much larger than the number of possible values for the property. We illustrate the feasibility of our methods by presenting a publicly available proof-of-concept implementation in C++ and using it to evaluate a heatmap function over encrypted geographic points.

Keywords: Privacy-preserving Computation, Homomorphic encryption, Ring Learning With Errors

1 Introduction

The need of selecting and counting objects occurs in many security protocols. Counting objects with a same property is a task that occurs in many security protocols and machine learning applications. A typical example is that of the heatmap computation, in which a 2-dimensional map is divided into a grid of cells and the property of interest is the cell in which a point lies in. Thus, counting how many points share the same property, i.e., lie in the same cell, provides a graphical representation of the distribution of the points. The heatmap itself has several applications and can be used, for example, in algorithms designed to count the number of objects in a certain area in order to prevent future congestion or people with a certain behavior.

The goal of this work is to design efficient solutions for selecting and counting dataset elements while preserving the privacy of the inputs. We investigate privacy preserving techniques based on fully homomorphic encryption (FHE) to solve this problem. We show two general approaches and demonstrate how they work with a heatmap example. Before going through our techniques, let us first describe our target scenario.

Scenario. Given an encrypted dataset $X = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ with n inputs, where each input has m variables, i.e., $\mathbf{x}_i \in \mathbb{Z}^m$ and a function $f : \mathbb{Z}^m \to \mathbb{Z}$, we want to homomorphically count how many inputs evaluate to the same value in the image of f. In other words, by defining $X_i := \{\mathbf{x} \in X : f(\mathbf{x}) = i\}$, we want to obtain ciphertexts encrypting $|X_i|$ for each $i \in \mathsf{img}(f)$. By viewing f as some property over dataset elements, we want to know how many points have the same property.

For example, X could contain medical data of n people and we would like to know how many of them have a high risk of developing some disease. Then, the function f can be defined as $f(\mathbf{x}) = 1$ if a patient with data input labeled as $\mathbf{x}$ has a risk of developing the disease and 0 otherwise. At the end, we get a final output of the form $(0, n_0), (1, n_1)$ where n_0 (resp. n_1) is the number low risk (resp. of high risk) patients.

We can imagine more complex functions, like f defined as $f(\mathbf{x}) = k$ if the estimated probability $p_{\mathbf{x}}$ of developing the disease belongs to $[k/10, (k+1)/10[$ for $0 \leq k < 10$, for example. Then, the final output, after decryption, would be of the form $(0, n_0), \cdots, (9, n_9)$, where n_i indicates the number of people having a risk with probability between $\frac{i}{10}$ and $\frac{i+1}{10}$. For instance, $(9, 6)$ would tell us that six people are estimated to have a risk larger than 90% of having the disease.

1.1 Computation challenges

Homomorphically selecting and counting requires the evaluation of a function f on each input of the dataset, then somehow grouping values that are equal and counting how many elements each group has. It seems hard to use fully homomorphic encryption (FHE) schemes like BGV [BGV12] and FV [FV12] for this task, since even simple non-algebraic functions, like $f(x) = \lfloor x/5 \rceil$, are hard to compute with them. Moreover, the "scoring" step, where we count the equal values, seems highly non-trivial with such schemes.

By using FHE schemes of the so-called third generation, like FHEW [DM15], TFHE [CGGI20], GAHE [Per21], or FINAL [BIP+22], we can express f as a lookup table and efficiently evaluate $f(x)$, which is an advantage over BGV/FV when f cannot be easily represented by polynomials. However, the size of the lookup tables that can be supported by the current techniques with reasonable timing — either in leveled and bootstapped mode — is typically very small, say with 6-bit inputs and outputs. Moreover, the additional aggregation step to count how many $f(\mathbf{x})$ have the same image remains hard with TFHE-like schemes.

Essentially, we need a way to represent bins for each possible value in the set of the image of the function f, $\mathsf{img}(f)$, for instance, one ciphertext for each bin. Then, after computing an encryption of $y \in \mathsf{img}(f)$, we have to homomorphically add 1 only to the correct bin.

To solve this, instead of evaluating the function as usual to obtain encryption of $f(m)$, which we denote $\mathtt{Enc}_{\mathsf{sk}}(f(m))$, and then counting homomorphically, we propose the following general strategy, which exploits the fact that most existing FHE schemes can encrypt polynomials: we evaluate f in the exponent of the monomial X, generating thus $\mathtt{Enc}_{\mathsf{sk}}(X^{f(m)})$. Hence, the counting step reduces to simply adding the ciphertexts. Namely, messages that are evaluated to the same value will produce encryptions of the same power X^y; then adding all the ciphertexts will produce an encryption of a polynomial $a_0 + a_1 X^1 + \cdots + a_{I-1} X^{I-1}$, where $I := |\mathsf{img}(f)|$ and each coefficient a_i tells us how many messages m satisfy $f(m) = i$. For example, if ten different messages m_i satisfy $f(m_i) = 7$, then, we will obtain ten encryptions of X^7 and adding all of them will produce a ciphertext encrypting a polynomial $a(X) = \sum_{i=0}^{I-1} a_i X^i$ such that $a_7 = 10$. This is illustrated in Figure 1.

We extend this technique to compute functions on several variables. Namely, for a function $g(x_1, ..., x_n) := \alpha_1 \cdot f_1(x_1) + ... + \alpha_n \cdot f_n(x_n)$, we first obtain $X^{f_i(x_i)}$, then we use automorphisms to multiply the exponents by some constant α_i and add all of them by using homomorphic multiplications. After this, we can proceed as before and simply add all the ciphertexts to obtain a polynomial whose coefficients count how many points $(x_1, ..., x_n)$ evaluate to the same value in the image of g.

Overall, we present two different ways of implementing this general strategy and as an application, we propose the evaluation of a heatmap over encrypted points. We provide a proof-of-concept implementation to support the correctness of our two techniques and we obtain that for a square grid with $2^{10} \times 2^{10}$ cells, the evaluation of one point with our first method takes around 0.174 seconds and 0.507 seconds with our second method. For a larger number of cells, the second method becomes faster than the first method: for a square grid with $2^{14} \times 2^{14}$ cells for example, the evaluation of one point with our second method takes around 0.635 seconds, while the first one requires 1.061 s. Note that the timing can be easily amortized as the number of points grows, since the first computation step Eval in Figure 1 can be done in parallel for each point, only the homomorphic sum at the end involves several different points.

Related art for private heatmap computation. Bampoulidis et al. [BBH+20] show how to build the heatmap of an infection desease spread using cell phone location data. To protect the privacy of cell phone owners, they use a combination of homomorphic encryption, zero-knowledge proof techniques and differential privacy. However, the authors assume that location-sensitive data aggregation is performed by a cellular network provider in the clear. In our work, we assume that data of individual users or objects is encrypted before aggregation.

Melis et al. [MDD16] show how to produce a heatmap of San Francisco cabs. In [DPP17], Dahl et al. showed a protocol computing a heatmap of popular shops in New York City districts according to Foursquare users. In both papers, it is assumed that users preprocess their input such that the aggregation stage boils down to summation. In our work, we assume that users send a 'raw' data, i.e. they have no prior knowledge of how their input will be aggregated. Our use case can be considered as a special type of private database queries. Namely, we query the number of values satisfying different input conditions.

Cheon et al. [CKK15] provided a framework for the design of queries with equality and range conditions using SHE/FHE. For example, this framework turns a heatmap use case to the computation of four comparison functions per each pair (object, region) and then counting the results per region. If N is the number of regions, K is the number of objects and μ is the bitsize of object coordinates, this algorithm requires $\mathcal{O}(\mu NK)$ ciphertext-ciphertext multiplications and a circuit of multiplicative depth $\log \mu + 4$.

In our work, the same task can be done with $(\log N - 1)K$ ciphertext-ciphertext multiplications, $(\log N)^2 K$ homomorphic automorphisms and a circuit of multiplicative depth $\log \log N$. If $\mu \leq \log N$, our approach is better in all respects.

Why we are using homomorphic encryption. Since homomorphic encryption schemes are known for not being very efficient in practice, one could wonder if other cryptographic primitives could be used to perform such queries that evaluate a function then count, while maintaining both the database and the results secret. Hence, we consider a few alternatives and discuss possible advantages and also their limitations.

– *Searchable encryption* (SE) allows a user to store an encrypted database in a server, then securely retrieve elements that match some keyword. Thus, SE is not designed to evaluate the same type of queries that we are considering in this work, because the server cannot compute a function f on the encrypted database using SE. To overcome this, the user would have to precompute the values $f(\mathbf{m})$ for every $\mathbf{m}$ in the database, encrypt $f(\mathbf{m})$ and include it in the encrypted database. Then, for each $k \in \mathsf{img}(f)$, the client would perform a query using k as

keyword, retrieving the elements that match $f(\mathbf{m}) = k$. However, with this solution, the client would be essentially downloading the whole database, thus, the communication cost would be as bad as the one of the trivial solution, which consists in downloading the encrypted database, decrypting all the column corresponding to $f(\mathbf{m})$ and counting by itself. We stress that in our proposed solutions, the client just downloads one ciphertext.

- *CryptDB* [PRZB11] is a tool that can perform search on encrypted database and is particularly designed to support SQL queries. The architecture assumes a semi-honest proxy and is built on the combination of property preserving encryption, additive homomorphic encryption and searchable encryption. Because of the primitives used by CryptDB, the server cannot evaluate a function f on the encrypted database, thus, again, the client would have to compute $f(\mathbf{m})$ beforehand, encrypt it and include it in the database. This is not ideal, since f can be hard to compute or can even be private (e.g., f corresponds to machine learning model trained by the server and which the server does not want to reveal). Notice that in our solution, the client just needs to know some upper bounds on the size of the domain and the image of f. But assuming that the client could compute f on the entire database, then, using CryptDB, they could run select and sum queries for each $k \in \mathsf{img}(f)$ and obtain ciphertexts corresponding the number of elements $\mathbf{m}$ such that $f(\mathbf{m}) = k$, as desired. Such queries take less that 1 ms which is much faster than what we obtain using our FHE-based solution. However, while we provide data privacy as long as the underlying FHE scheme is IND-CPA secure and the server is semi-honest, CryptDB uses order-preserving encryption (OPE) to enable these select-and-count queries, but OPE is deterministic and does not offer the same security level as our solutions. In particular, [NKW15] presented a series of attacks showing that sensitive information can be recovered from the encrypted database when OPE is used.
- *Private information retrieval* (PIR) protocols offer the following functionality: suppose that a server stores a database with n elements. Then, the client can choose an index i and download the i-th element of the database without revealing the value i to the server. Also, the communication cost is sublinear in the size of the database. It is clear that PIR does not give the server the ability to compute a function f over encrypted data. Moreover, it is not possible for the client to select elements based on the value of any chosen column, just based on the index. In particular, the client would not be able to select elements $\mathbf{m}$ that satisfy $f(\mathbf{m}) = k$.

Since we want the server to compute f and we do not want to leak information about the data or the results of the queries, homomorphic encryption is a natural solution. Nevertheless, simply using FHE schemes in a black box manner would not produce efficient solutions as previously discussed in this section. As a concrete example, in Appendix A, we show how schemes like BGV and FV could be used to evaluate a homomorphic heatmap and argue that the running times one would obtain would be between 57 and 219 times slowers than the timings we obtain with our solutions.

1.2 Summary of our techniques

In both methods, instead of simply encrypting the inputs denoted as z_i, we actually encrypt the inputs in the exponent as X^{z_i}, as this allows us to homomorphically evaluate $f(z_i)$ and obtain directly encryption of $X^{f(z_i)}$. If the final function we want to evaluate is simply f, then we can add all the obtained ciphertexts $X^{f(z_i)}$ for all the possible inputs z_i's to obtain an encryption of the desired polynomial whose coefficients count the elements.

First method: the general idea is to use the so-called "test-vector polynomials" akin to the ones employed in the TFHE bootstrapping [CGGI20] to obtain encryptions of the binary decomposition

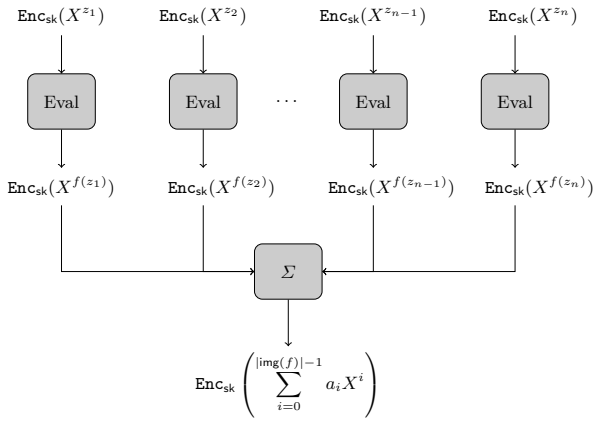

Fig. 1: General strategy to perform the homomorphic counting, where $\mathsf{Enc}_{\mathsf{sk}}(m)$ denotes the homomorphic encryption of m under secret key sk. The function f is evaluated in each of the n elements of the dataset, then the counting step is performed by a simple addition. Each coefficient a_i of the output is equal to number of elements z such that $f(z) = i$.

of $f(z)$, then, use these encryptions to construct an encryption $\mathsf{Enc}_{\mathsf{sk}}(X^{f(z)})$. In more details, consider a function f with ℓ-bit output. Then for $0 \leq i \leq \ell - 1$, we define a polynomial $T_i(X)$ that encodes the i-th bits of all target outputs of f. In particular, we define $T_i(X)$ such that the constant term of $T_i(X) \cdot X^z$ is equal to the i-th bit of $f(z)$. We show that given $T_i(X)$ and an encryption of X^z, we can homomorphically compute the encryption of the i-th bit of $f(z)$, which we denote by $f(z)[i]$.

Then, since $b(X^{2^i} - 1) + 1 = X^{b \cdot 2^i}$ for any bit b, we can obtain encryptions of $X^{2^i \cdot f(z)[i]}$. Multiplying all these ciphertexts homomorphically, we finally obtain the encryption of $X^{\sum 2^i \cdot f(z)[i]} = X^{f(z)}$. If we want to compute more complex functions like $g(x_0, \cdots, x_{m-1}) = \sum_{i=0}^{m-1} \alpha_i f_i(x_i)$, we just need to repeat this process to obtain encryptions of $X^{f_i(x_i)}$, then use the automorphisms and multiplications in order to obtain the homomorphic encryption of $\alpha_i f_i(x_i)$ and $\sum_{i=0}^{m-1} \alpha_i f_i(x_i)$ encoded in the exponent.

In this approach, the parameter N defining the "degree" of the polynomial ring $\mathsf{R} := \mathbb{Z}[x]/\langle X^N + 1\rangle$ used in the FHE scheme, has to be larger than or equal to the size of all the domains and images of f_i and g. However, sometimes they have very different sizes and it would be desirable to work with a smaller N for some of the functions, since all the homomorphic operations become more expensive as N increases. In particular, it may happen that $|\mathsf{dom}(f)| \gg |\mathsf{img}(f)|$ and thus, it would be better to choose N as the size of the image instead of the domain.

Second method: with this approach, we are able to work with different rings. This method is specially suitable for the cases where the domain is larger than the image. It starts with the following observation: for a power $X^z \in \mathsf{R}$, consider the indicator vector $\boldsymbol{\phi}(X^z) := (0, \cdots, 0, 1, 0, \cdots, 0) \in \{0,1\}^N$ with a single 1 at position $z + 1$, i.e., a vector indicating the position of the corresponding exponent. Also, for a given function f, define

$$\mathbf{u}_f := (X^{f(0)}, X^{f(1)}, \cdots, X^{f(N-1)}).$$

Then, if we want to obtain $f(z)$ for some $0 \leq z \leq N - 1$, it is clear we can compute $X^{f(z)} = \boldsymbol{\phi}(X^z) \cdot \mathbf{u}_f$. Thus encrypting all the monomials X^{z_i} for each possible dataset input z_i and somehow

multiplying each ciphertext by $\mathbf{u}_f$ gives us encryptions of $X^{f(z_i)}$, and it remains to sum all the encryptions to count the number of inputs with the same property.

However, ideally, we would like to evaluate more general functions, such as multivariate functions of the form $g(z_0, \cdots, z_{d-1}) = \sum_{i=0}^{d-1} \alpha_i f_i(z_i)$. In order to do so, we need to multiply the encryptions of $X^{f_i(z_i)}$, but after multiplication by $\mathbf{u}_{f_i}$, we obtain ciphertexts whose format does not allow us to perform efficient homomorphic multiplications, because multiplication by $\mathbf{u}_{f_i}$ is not a native operation in any existing FHE scheme. Thus, we propose a new key-switching procedure to obtain again regular Ring-LWE ciphertexts encrypting $X^{f_i(z_i)}$, allowing us also to switch the dimension from N to a bigger N, so that we are able to perform usual homomorphic multiplications and get an encryption of $X^{g(x_0,\ldots,x_{d-1})}$. An example with $d = 2$ and $\alpha_1 = \alpha_2 = 1$ is shown in Figure 2.

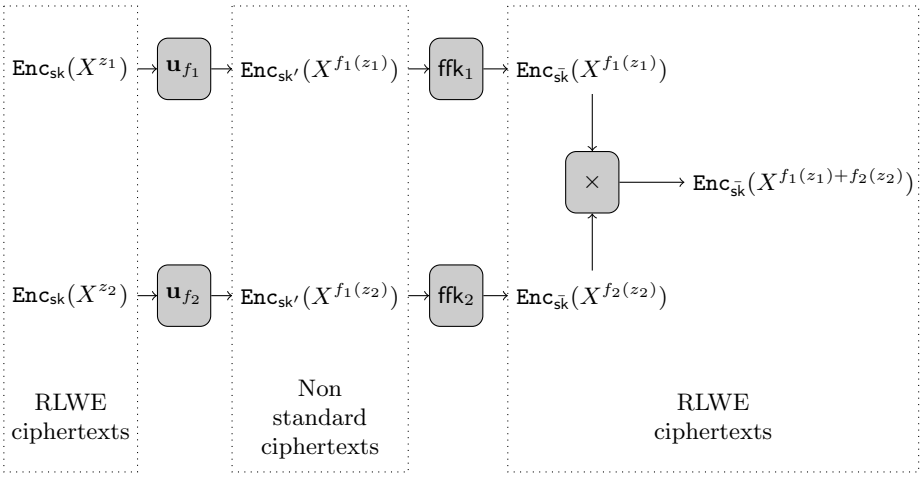

Fig. 2: Computing a function $g(x_1, x_2) := f_1(x_1) + f_2(x_2)$. The ciphertexts in the first layer are defined over $\mathsf{R} := \mathbb{Z}[X]/\langle X^N + 1\rangle$, the ones in the middle are not regular RLWE ciphertexts, and the ones in the last layer are defined over $\bar{\mathsf{R}} := \mathbb{Z}[X]/\langle X^{\bar{N}} + 1\rangle$, where we can set $\bar{N} > N$ if the image of g is larger than the image of both f_1 and f_2. The format-fixing key-switching keys are denoted by ffk_1 and ffk_2.

2 Preliminaries

2.1 Notations

We let $\mathsf{R} = \mathbb{Z}[X]/\langle X^N + 1\rangle$, where N is a power of two. We denote the quotient ring $\mathsf{R}/q\mathsf{R}$ by R_q.

We denote the set of integers modulo M by $\mathbb{Z}_M$ and its multiplicative group by $\mathbb{Z}_M^\star$. We let ζ_{2N} be a primitive $2N$-th root of unity and we denote $\mathsf{K} = \mathbb{Q}(\zeta_{2N})$ the $2N$-th cyclotomic field, which is a Galois extension of $\mathbb{Q}$. For $\sigma_d \in \mathrm{Gal}(\mathsf{K}/\mathbb{Q})$, we have $\sigma_d(\zeta_{2N}) = \zeta_{2N}^d$, for $d \in \mathbb{Z}_{2N}^\star$. Note that the trace function $\mathrm{Tr}_{\mathsf{K}/\mathbb{Q}}$ sends an element $a \in \mathsf{K}$ to $\mathrm{Tr}_{\mathsf{K}/\mathbb{Q}}(a) = \sum_{d\in\mathbb{Z}_{2N}^\star} \sigma_d(a)$. We will recall later how to homomorphically evaluate the trace function.

For any $c \in \mathsf{R}$, let $\boldsymbol{\phi}(c) \in \mathbb{Z}^N$ be the coefficient vector of c and $\boldsymbol{\Phi}(c) \in \mathbb{Z}^{N\times N}$ be the anti-circulant matrix of c, i.e. $\boldsymbol{\Phi}(c) = \left(\boldsymbol{\phi}(c \cdot X^0), \cdots, \boldsymbol{\phi}(c \cdot X^{N-1})\right)^T$. Notice that $\boldsymbol{\phi}(c + c') = \boldsymbol{\phi}(c) + \boldsymbol{\phi}(c')$ and $\boldsymbol{\phi}(c \cdot c') = \boldsymbol{\Phi}(c) \cdot \boldsymbol{\phi}(c')$.

For a real number x, $\lceil x \rceil$, $\lfloor x \rfloor$, and $\lfloor x \rceil$ denote the rounding up, rounding down, and the nearest integer to x, respectively. We denote by $[N]$ the set of integers $\{0, \cdots, N-1\}$ and by $[a, b]$, the set of integers $\{a, a+1, \cdots, b\}$. For an integer with bit decomposition a, let $a[i]$ be the i-th bit of a. For a polynomital $a(X)$ or a vector $\mathbf{a}$, we abuse notation and write a_i to denote its i-th coefficient.

We denote by χ_{err} the error distribution over R where each coefficient is sampled from a discrete Gaussian distribution of small variance and by χ_{key} the uniform distribution over polynomials in R with coefficients in $\{-1, 0, 1\}$.

2.2 Ring-LWE encryption

Message encoding and RLWE Encryption. We present a symmetric-key version of the RLWE encryption scheme known as FV [FV12]. We use it as the base of our solutions. Specifically, the method presented in Section 3 uses FV as a black-box, adding only an encoding layer before encryption, while the method presented in Section 4 proposes new homomorphic operations that generate ciphertexts whose format no longer satisfies the FV format. It is worth noticing that the choice of the FV scheme is somehow arbitrary and our techniques also work with the BGV scheme [BGV12] or essentially any other RLWE-based homomorphic encryption scheme that supports additions, multiplications, and automorphisms evaluation.

- $\mathsf{ParamGen}(1^\lambda)$: given the security parameter, choose N as a power of two and some integers q and t. Define $\Delta := \lfloor q/t \rfloor$. Let $\mathsf{R} := \mathbb{Z}[X]\langle X^N + 1\rangle$, $\mathsf{R}_q := \mathsf{R}/q\mathsf{R}$ and $\mathsf{R}_t := \mathsf{R}/t\mathsf{R}$. The message and the ciphertext spaces are R_t and R_q^2, respectively. Define two distributions χ_{err} and χ_{key} over R. Output $\mathsf{params} := (N, q, \Delta, \chi_{\mathsf{err}}, \chi_{\mathsf{key}})$.
- $\mathsf{KeyGen}(\mathsf{params})$: Sample $s \leftarrow \chi_{\mathsf{key}}$ and let $\mathtt{rlk}$ be the relinearization key with respect to s. Output $\mathsf{sk} := s$ and $\mathtt{rlk}$.
- $\mathsf{KeyGenAut}(\mathsf{params}, \mathsf{sk}, k)$: Interpret sk as $s \in \mathsf{R}$. Given an odd $k \in [N]$, output a key-switching key swt_k from $s(X^k)$ to s.
- $\mathsf{Enc}_{\mathsf{sk}}(m)$: Consider $m \in \mathsf{R}_t$. Sample a uniformly at random from R_q, and $e \leftarrow \chi_{\mathsf{err}}$. Compute $b := a \cdot \mathsf{sk} + e + \Delta \cdot m \in \mathsf{R}_q$. Output $\mathbf{c} := (a, b)$.
- $\mathsf{Dec}_{\mathsf{sk}}(\mathbf{c})$: Let $(a, b) := \mathbf{c}$. Compute $b^\star := b - a \cdot \mathsf{sk}$ over R. Output $\lfloor t \cdot b^\star / q \rceil \bmod t$.

Thus, for a ciphertext $(a, b) \leftarrow \mathsf{Enc}_{\mathsf{sk}}(m)$, we have $b = a \cdot \mathsf{sk} + e + \Delta m$. Decryption is guaranteed to be correct if the noise satisfies $\|\phi(e)\|_\infty < q/t$.

To simplify the exposition, the homomorphic operations will be presented in a high-level way. Consider that $\mathbf{c}_i$ is an encryption of m_i.

- $\mathsf{Add}(\mathbf{c}_0, \mathbf{c}_1)$: Output $\mathbf{c} := \mathbf{c}_0 + \mathbf{c}_1$, which is an encryption of $m_0 + m_1 \in \mathsf{R}_t$.
- $\mathsf{Mult}(\mathbf{c}_0, \mathbf{c}_1, \mathtt{rlk})$: Compute the tensor product $\mathbf{c}' := \mathbf{c}_0 \otimes \mathbf{c}_1 \in \mathsf{R}_q^4$, which encrypts $m_{mult} := m_0 \cdot m_1 \in \mathsf{R}_t$ under sk^2, then use $\mathtt{rlk}$ to transform $\mathbf{c}'$ into $\mathbf{c}_{mult} \in \mathsf{R}_q^2$, which encrypts m_{mult} under sk again. Output $\mathbf{c}_{mult}$.
- $\mathsf{Aut}(\mathbf{c}_i, k, \mathsf{swt}_k)$: Let $(a, b) := \mathbf{c}_i$. Interpret both a and b as polynomials in $\mathbb{Z}[X]$. Substitute X by X^k and reduce modulo $\langle X^N + 1, q\rangle$, obtaining $\mathbf{c}' := (a(X^k), b(X^k)) \in \mathsf{R}_q^2$, which encrypts $m(X^k)$ under $\mathsf{sk}(X^k)$. Use swt_k to transform $\mathbf{c}'$ into a ciphertext $\mathbf{c}_k$ under sk. Output $\mathbf{c}_k$.

Roughly speaking, the relinearization key is an encryption of s^2 under s. It is used during the homomorphic multiplication so that the output ciphertext is encrypted under the same key as the input ciphertexts. The key-switching keys are used in the automorphisms for the same reason and it is essentially an encryption of $s(X^k) \bmod X^N + 1$ under s. For more details, we refer to Section 1.2 of [Zuc18].

3 Full domain approach

In this section, we present our first approach to instantiate the general strategy shown in Figure 1. Firstly, we present the univariate case, i.e., we consider a function $f : [D] \to [I]$ where D and I are integers smaller than N and show how to obtain $\mathsf{Enc}_{\mathsf{sk}}(X^{f(z)})$. As discussed previously, it is easy to generalize this strategy for multivariate functions of the form $g(x_1, ..., x_n) = \sum \alpha_i f_i(x_i)$. Any input and output of f is encoded by a monomial $X^z \in \mathsf{R}_t$. The main idea of this approach is to obtain encryptions of the binary decomposition of $f(z)$, then combine then into an encryption of $X^{f(z)}$.

Let's start with a warm up example.

Example with $f(x) = \lfloor \frac{x}{3} \rfloor$ and $N = 8$. Let $\mathsf{R} = \mathbb{Z}[X]/\langle X^8 + 1 \rangle$, i.e. $N = 8$. For a polynomial $a \in \mathsf{R}$, we denote by $\mathtt{coeff}_0(a)$ the constant term of a. We consider the following look-up table for $f(z)$ where $z \in [0, 7]$:

z	0	1	2	3	4	5	6	7
$f(z) = \lfloor \frac{z}{3} \rfloor$	0	0	0	1	1	1	2	2

The idea is first to define test-vector polynomials that encode the bits of all possible outputs of f. Since, $f : [N] \to [3]$, just 2 bits are enough to describe the output of $f(z)$, and thus, we just need two test-vector polynomials $T_0(X)$ and $T_1(X)$.

1. We begin by defining two polynomials $T_0(X), T_1(X)$, such that:
   - for $z \in [0, 7]$, $\mathtt{coeff}_0(T_0(X) \cdot X^z) = f(z)[0]$;
   - for $z \in [0, 7]$, $\mathtt{coeff}_0(T_1(X) \cdot X^z) = f(z)[1]$.
2. Hence, the next step consists of homomorphically extracting the constant terms $f(z)[i]$ of these two polynomials $T_i(X) \cdot X^z$ by applying the trace function. This step will be detailed later in the paper.
3. Since $f(z)[i](X^{2^i} - 1) + 1 = X^{f(z)[i] \cdot 2^i}$, we then homomorphically combine the encryption of the bits $f(z)[i]$ in order to obtain $X^{\sum 2^i \cdot f(z)[i]} = X^{f(z)}$. Thus, we compute the products of $f(z)[i](X^{2^i} - 1) + 1$ for all i in order to obtain $X^{f(z)}$.
4. In order to obtain the number of elements z such that $f(z) = i$, we sum all the obtained monomials together. The coefficient a_i of X^i is the number of elements that are mapped to $f(z)$.

Now, going back to our example, let's define the two test-vector polynomials such that the constant term of $T_0(X) \cdot X^z$ (resp. $T_1(X) \cdot X^z$) is equal to the first (resp. second) bit of $f(z)$. We thus have:

$$T_0(X) = X^{-3} + X^{-4} + X^{-5},$$
$$T_1(X) = X^{-6} + X^{-7}.$$

It is clear that for any $z \in \{0, 1, 2\}$, we have

$$\mathtt{coeff}_0(T_0(X)X^z) = 0 = f(z)[0],$$
$$\mathtt{coeff}_0(T_1(X)X^z) = 0 = f(z)[1].$$

Extracting these constant terms and then combining them we obtain

$$(f(z)[0](X-1)+1)(f(z)[1](X^2-1)+1) = 1 = X^{f(z)}.$$

For any $z \in \{3,4,5\}$, we have

$$\begin{aligned}\texttt{coeff}_0(T_0(X)X^z) &= 1 = f(z)[0],\\ \texttt{coeff}_0(T_1(X)X^z) &= 0 = f(z)[1].\end{aligned}$$

Extracting these constant terms and then combining them we obtain

$$(f(z)[0](X-1)+1)(f(z)[1](X^2-1)+1) = X = X^{f(z)}.$$

For any $z \in \{6,7\}$, we have

$$\begin{aligned}\texttt{coeff}_0(T_0(X)X^z) &= 0 = f(z)[0],\\ \texttt{coeff}_0(T_1(X)X^z) &= 1 = f(z)[1].\end{aligned}$$

Extracting these constant terms and then combining them we obtain

$$(f(z)[0](X-1)+1)(f(z)[1](X^2-1)+1) = X^2 = X^{f(z)}.$$

Then, we sum all the combinations obtained in order to perform the counting step, such that the coefficient of the monomials X^i, for $0 \leq i \leq 2$, gives the number of elements which are mapped to i.

3.1 Computing output bits.

The crucial idea of this solution is to compute all the possible bit outputs of $f(z)$ inspired by a look-up table encoding in the TFHE functional or multi-value bootstrapping. Namely, we design the so-called 'test vector' polynomials $T_i(X) \in \mathsf{R}_q$ for $i \in [0, \lceil \log_2 I \rceil - 1]$ such that the constant term of $T_i(X) \cdot X^z$ is equal to the ith bit of $f(z)$.

Denoting the i-th bit of $f(z)$ by $f(z)[i]$, such polynomials $T_i(X)$ are defined as follows

$$T_i(X) = \sum_{j=0}^{N-1} f(j)[i] \cdot X^{-j}. \tag{1}$$

Lemma 1. *Let $T_i(X)$ be defined as in Equation 1. Then, the constant term of $T_i(X) \cdot X^z$ is equal to $f(z)[i]$.*

Proof. Note that $X^{-k} = -X^{N-k}$ in R as we work modulo $X^N + 1$, thus, $T_i(X) \in \mathsf{R}$. Furthermore, $T_i(X)$ satisfies the following equality

$$T_i(X) \cdot X^z = f(z)[i] + \underbrace{\sum_{j \neq z} f(j)[i] \cdot X^{z-j}}_{g(X)}$$

Now, we want to show that the constant term of $g(X)$ is zero. But that is easy to see, since g_0 is defined by $z - j$ such that $z - j \equiv 0 \mod N$, and because both z and j belong to $\{0, ..., N-1\}$, it is clear that $-N < z - j < N$ and the only value congruent to 0 modulo N in this interval is the zero itself. Therefore, for the constant term of $g(X)$ to be different from zero, we would need $z = j$ to be included in the sum defining $g(X)$.

Algorithm 1: Trace operation

Input: $a(X) \in \mathsf{R}_t$

Output: $r(X) = Na_0$ where a_0 is the constant term of $a(X)$

$r_0(X) \leftarrow a(X)$

for $\ell \leftarrow 1$ **to** $\log_2 N$ **do**

  $r_\ell(X) \leftarrow r_{\ell-1}(X) + \tau_{i_\ell}(r_{\ell-1}(X))$

Return $r_{\log_2 N}(X)$.

Hence, given $\mathsf{Enc}_{\mathsf{sk}}(T_i(X) \cdot X^z)$, we can homomorphically compute $\mathsf{Enc}_{\mathsf{sk}}(T_i(X) \cdot X^z)$ for $0 \leq i \leq \lceil \log_2 I \rceil - 1$, which gives us encryptions of polynomials that have the desired bits in the coefficient factors. This requires $\lceil \log_2 I \rceil$ plaintext-ciphertext multiplications.

3.2 Extraction of bits.

In Lemma 1, we see that after multiplication by the test-vector polynomial, we obtain a $\mathtt{Enc}_{\mathsf{sk}}(a(X))$ for some polynomial $a(X)$ whose constant term is the i-th bit of $f(z)$, denoted by $f(z)[i]$. But we are interested only in that bit, thus, we want to "remove" all the other coefficients of $a(X)$ and obtain $\mathtt{Enc}_{\mathsf{sk}}(f(z)[i])$. This can be done by computing the trace function $\mathrm{Tr} : \mathsf{R}_t \to \mathbb{Z}$ that maps $a(X) = \sum_{i=0}^{N-1} a_i X^i \in \mathsf{R}_t$ to Na_0. Since t and N are co-prime, the multiple of N can be removed by computing $N^{-1} \cdot \mathrm{Tr}(a(X)) = a_0 \mod t$.

To describe the trace function, we need the automorphisms of R_t, which are defined as

$$\tau_i : a(X) \mapsto a(X^i)$$

where $i \in \mathbb{Z}_{2N}^\star$. It is a standard fact that the set $\{\tau_i\}_{i \in \mathbb{Z}_N^\star}$ forms a group under the composition of functions.

To perform an automorphism τ_i homomorphically, one needs to compute τ_i on both components of an input RLWE ciphertext, i.e., $(a, b) \mapsto (\tau_i(a), \tau_i(b))$, then perform key-switching from $\tau_i(\mathsf{sk})$ to sk using the key-switching key swt_i, as described in subsection 2.2. More details can be found in [GHS12].

Our goal now is to show that the trace operation defined in Algorithm 1 indeed extracts the constant term of the message. For $0 \leq \ell \leq \log_2 N - 1$, define J_ℓ as the set of integers $k2^\ell$ for odd $k \in [1, N/2^\ell - 1]$. Also, let $J_{\log_2 N} := \{0\}$ and $J_{-1} := \emptyset$ Notice that $J_\ell \cap J_{\ell'} = \emptyset$ if $\ell \neq \ell'$ and that $[1, N-1] = \cup_{\ell=0}^{\log_2 N - 1} J_\ell$ with disjoint union. Furthermore, the product of any $j \in J_\ell$ and $i_\ell := N/2^\ell + 1$ is equal to $i_\ell \cdot j = j + kN = j + N \mod 2N$ since k is odd. In other words, $\tau_{i_\ell}(X^j) = X^{j+N} = -X^j$ for any $j \in J_\ell$. And for $j \in J_{\log_2 N}$, $j \cdot i_\ell = j \mod 2N$, hence $X^{j \cdot i_\ell} = X^j$.

Lemma 2. *Algorithm 1 outputs* $Na_0 \bmod t$.

Proof. Let $r_\ell(X) \in \mathsf{R}_t$ be the polynomial obtained at the ℓ-th iteration for $\ell \in [0, \log_2 N]$. We have $r_0(X) = a(X)$ and $I = [0, N-1]$. We prove by induction that at the ℓ-th iteration:

$$r_\ell(X) = 2^\ell \cdot \sum_{j \in I \setminus \cup_{k<\ell} J_k} a[j] X^j \mod t$$

Indeed, for the base case $\ell = 0$, $r_0(X) = 2^\ell \sum_{j \in I \setminus J_{-1}} a[j]X^j$, i.e $r_0(X) = a(X)$. By induction, the following holds modulo t:

$$\begin{aligned} r_{\ell-1}(X) = 2^{\ell-1} \cdot \sum_{j \in I \setminus \cup_{k<\ell} J_k} a[j]X^j \\ + 2^{\ell-1} \cdot \sum_{j \in J_{\ell-1}} a[j]X^j \end{aligned} \tag{2}$$

Applying τ_{i_ℓ} gives:

$$\begin{aligned} \tau_{i_\ell}\left(r_{\ell-1}(X)\right) = 2^{\ell-1} \cdot \sum_{j \in I \setminus \cup_{k<\ell} J_k} a[j]X^j \\ - 2^{\ell-1} \cdot \sum_{j \in J_{\ell-1}} a[j]X^j \mod t \end{aligned} \tag{3}$$

Summing (2) and (3) gives :

$$r_\ell(X) = 2^\ell \cdot \sum_{j \in I \setminus \cup_{k<\ell} J_k} a[j]X^j \mod t$$

At the end of Algorithm 1, we obtain $r_{\log_2 N} = 2^{\log_2 N} \cdot \sum_{j \in I \setminus \cup_{k<\log_2 N} J_k} a[j]X^j \mod t$, i.e $r_{\log_2 N} = N \cdot \sum_{j \in J_{\log_2 N}} a[j]X^j = Na_0 \mod t$.

Hence, performing the trace function homomorphically on $\mathsf{Enc}_{\mathsf{sk}}(T_i(X) \cdot X^z)$ and multiplying by $N^{-1} \mod t$, we obtain $\mathsf{Enc}_{\mathsf{sk}}(f(z)[i])$ for all $i \in [0, \lceil \log_2 I \rceil - 1]$. Since the trace function needs $\log_2 N$ automorphisms to be performed, the running time complexity of this step is equal to $(\log_2 N)(\lceil \log_2 I \rceil)$ automorphisms. Multiplications by $N^{-1} \mod t$ are very fast and thus ignored in this analysis.

3.3 Combining output bits.

It remains to combine the encrypted bits $f(z)[i]$ via tree-based homomorphic multiplication. The following lemma shows how to perform the computation:

Lemma 3. *Let $b_i = \mathsf{Enc}_{\mathsf{sk}}(f(z)[i])$ for $i \in [0, \lceil \log_2 I \rceil - 1]$. The following product is a valid encryption of $X^{f(z)}$:*

$$\prod_{i=0}^{\lceil \log_2 I \rceil - 1} b_i \cdot (X^{2^i} - 1) + 1.$$

Proof. Notice that $b_i \cdot (X^{2^i} - 1) + 1$ is a valid encryption of $X^{f(z)[i] \cdot 2^i}$ as $f(z)[i] \in \{0, 1\}$. As a result, the above product is a valid encryption of $X^{\sum_{i=0}^{\lceil \log_2 I \rceil - 1} f(z)[i] \cdot 2^i} = X^{f(z)}$ under key sk.

This step needs $\lceil \log_2 I \rceil$ plaintext-ciphertext multiplications by powers of X, which boil down to negacyclic shifts of polynomial coefficients and thus they are extremely fast and noise free. In addition, it requires $\lceil \log_2 I \rceil - 1$ ciphertext-ciphertext multiplications that result in a circuit of depth $\lceil \log_2 \lceil \log_2 I \rceil \rceil$.

Complexity of the full domain approach. The method we present in this section computes $X^{f(z)}$ with $\lceil \log_2 I \rceil$ plaintext-ciphertext multiplications, $(\log_2 N)(\lceil \log_2 I \rceil)$ automorphisms and $\lceil \log_2 I \rceil - 1$ ciphertext-ciphertext multiplications. Its multiplicative depth in ciphertext-ciphertext multiplications is $\lceil \log_2 \lceil \log_2 I \rceil \rceil$. For any known RLWE-based leveled FHE scheme [FV12,BGV12], the functional requirement is that $\log q \in \mathcal{O}(L(\log t + \log N))$ where L is the multiplicative depth of computation. This implies that in this method one ciphertext needs $\mathcal{O}(NL(\log t + \log N))$ bits of memory where $L \in \mathcal{O}(\log \log I)$ and the public key material including various key-switching keys amounts to $\mathcal{O}(NL^2(\log t + \log N)^2)$. The running time complexity of ciphertext-ciphertext multiplication or time complexity of an automorphism in such schemes is bounded by $\mathcal{O}(N \cdot \log N \cdot \text{polylog}(q))$ basic operations. Hence, this approach requires running time $\mathcal{O}(N \cdot (\log N)^2 \cdot \log I \cdot \text{polylog}(L(\log t + \log N)))$.

4 Split domain approach

In this section, we propose another method to homomorphically compute encryptions of $X^{g(x_1,\cdots,x_n)}$, where $g(x_1, ..., x_n) = \sum_{i=0}^{n} \alpha_i f_i(x_i)$ for known integers α_i's and functions f_i's, as discussed in Section 1.2. We first show how to evaluate univariate functions in the exponent, then we proceed showing how to combine them to obtain $g(x_1, \cdots, x_n)$. This method is specially suitable for the cases where $|\mathsf{dom}(f)| > |\mathsf{img}(f)|$, as it allows us to choose the parameter N of the RLWE scheme equal to $|\mathsf{img}(f)|$.

4.1 Computing univariate functions

For two positive integers D and I, where $D \geq I$, consider a function $f : [D] \to [I]$. Let N be a power of two larger than or equal to I and define $k = \lceil D/N \rceil$. The idea here is to split the domain of f to reduce the degree N that we need to use when we instantiate the RLWE problem. Then, basically, the client sends k ciphertexts instead of one and the server repeats the same computation k times, using different ciphertexts as inputs. Notice that the size of a ciphertext is linear in the degree N, hence, using $N \approx D$ and sending one single ciphertext or using $N \approx D/k$ and sending k ciphertexts represents the same communication complexity for the client. However, the computation executed by the server is super linear in N, therefore, it takes less time to execute it k times with smaller N than one single time with a large N.

Thus, for $1 \leq i \leq k$, let $P_i := [(i-1)N, iN]$. Notice that $|P_i| = N$ for all i, $|P_i \cap P_j| = 0$ if $i \neq j$, and $\mathsf{dom}(f) \subset [kN] = P_1 \cup P_2 \cup \cdots \cup P_k$. Thus, we can use the sets P_i as a partition of $[D]$.

Then, for each message $m \in \mathsf{dom}(f)$, the client produces k ciphertexts $\mathbf{c}_i$'s as follows: if $m \in P_i$, then we can write m as $m = (i-1) \cdot N + j$ for some $0 \leq j < N$ and encrypt X^j, i.e., define $\mathbf{c}_i := \mathsf{Enc}_{\mathsf{sk}}(X^j)$. If $m \notin P_i$, let $\mathbf{c}_i := \mathsf{Enc}_{\mathsf{sk}}(0)$, i.e., an encryption of zero. Thus, for every message, we have k ciphertexts.

Now, the server defines k vectors $\mathbf{u}_i$'s as

$$\mathbf{u}_i := (X^{f((i-1)N)}, X^{f((i-1)N+1)}, \cdots, X^{f(iN-1)}) \in \mathbb{Z}[X]^N$$

and it computes

$$(\mathbf{a}', b') := \sum_{i=1}^{k} \mathbf{u}_i \cdot \boldsymbol{\phi}(\mathbf{c}_i) \in \mathbb{Z}_q[X]^{N+1}$$

$$= \sum_{i=1}^{k} \left(\mathbf{u}_i \cdot \boldsymbol{\Phi}(a_i), \mathbf{u}_i \cdot \boldsymbol{\phi}(b_i)\right) \tag{4}$$

where $(a_i, b_i) = \mathbf{c}_i$, $\boldsymbol{\phi}$ denotes the coefficient vector, and $\boldsymbol{\Phi}$ the anti-circulant matrix, as explained in Section 2.1. Notice that no reduction modulo $X^N + 1$ is done, since all monomials in $\mathbf{u}_i$ have degree smaller than N and they are multiplied only by integers. Now we want to prove that, although not being a normal RLWE ciphertext, $(\mathbf{a}', b')$ is a valid encryption of $X^{f(m)}$.

Lemma 4. *Let (a, b) be an RLWE encryption of zero under the secret key $s \in \mathsf{R}$ with noise e. Then $(\mathbf{a}'_i, b'_i) := (\mathbf{u}_i \cdot \boldsymbol{\Phi}(a),\ \mathbf{u}_i \cdot \boldsymbol{\phi}(b))$ is of the form $(\mathbf{a}'_i, b'_i = \mathbf{a}'_i \cdot \boldsymbol{\phi}(s) + e'_i)$. Moreover, $\|e'_i\|_\infty \leq N \|e\|_\infty$.*

Proof. We know that $b = a \cdot s + e \in \mathsf{R}_q$ for some noise term e. Thus, $\phi(b) = \boldsymbol{\Phi}(a) \cdot \boldsymbol{\phi}(s) + \boldsymbol{\phi}(e) \in \mathbb{Z}^N$. Therefore, by defining $\mathbf{a}'_i := \mathbf{u}_i \cdot \boldsymbol{\Phi}(a)$ and $b'_i := \mathbf{u}_i \cdot \boldsymbol{\phi}(b)$, we have $b'_i := \mathbf{a}'_i \cdot \boldsymbol{\phi}(s) + e'_i$, where $e'_i := \mathbf{u}_i \cdot \boldsymbol{\phi}(e) = \sum_{j=0}^{N-1} X^{f((i-1)N+j)} \cdot e_i$. Moreover, it is clear that

$$\left\|e'_i\right\|_\infty \leq \sum_{j=0}^{N-1} \left\| X^{f((i-1)N+j)} \cdot e_i \right\|_\infty \leq \sum_{j=0}^{N-1} |e_i| \leq N \|e\|_\infty .$$

Lemma 5. *Let (a, b) be an RLWE encryption of X^j under the secret key $s \in \mathsf{R}$. Then $(\mathbf{a}'_i, b'_i) := (\mathbf{u}_i \cdot \boldsymbol{\Phi}(a),\ \mathbf{u}_i \cdot \boldsymbol{\phi}(b))$ is a valid encryption of $X^{f((i-1)N+j)})$ under key $\boldsymbol{\phi}(s)$, i.e. $(\mathbf{a}'_i, b'_i)$ is of the form $(\mathbf{a}'_i, b'_i = \mathbf{a}'_i \cdot \boldsymbol{\phi}(s) + e'_i + \Delta X^{f((i-1)N+j)})$. Moreover, $\|e'_i\|_\infty \leq N \|e\|_\infty$.*

Proof. We have $b = a \cdot s + e \in \mathsf{R}_q$, then, similarly to Lemma 4, we have $b'_i := \mathbf{a}'_i \cdot \boldsymbol{\phi}(s) + e'_i + \Delta \mathbf{u}_i \cdot \boldsymbol{\phi}(X^j)$. Because $\boldsymbol{\phi}(X^j)$ has a single one at the j-th position and zeros elsewhere, it is clear that $\mathbf{u}_i \cdot \boldsymbol{\phi}(X^j)$ is equal to the j-th entry of $\mathbf{u}_i$, which is $X^{f((i-1)N+j)}$ by definition.

Corollary 1. *Consider the pair $(\mathbf{a}', b')$ defined in Equation 4. It holds that $b' = \mathbf{a}' \cdot \boldsymbol{\phi}(s) + e' + \Delta X^{f(m)}$. Moreover, if the noise of each ciphertext $\mathbf{c}_i$ is bounded by some value B, then $\|e'\|_\infty \leq kNB$.*

Proof. It follows from lemmas 4 and 5 that $\mathbf{a}' = \sum_{i=1}^{k} \mathbf{a}'_i$ and $\mathbf{b}' = \mathbf{a}' \cdot \boldsymbol{\phi}(s) + \sum_{i=1}^{k} e'_i + \Delta X^{f((i-1)N+j)}$ for some j. But from the encryption procedure run by the client, we have $m = (i-1)N + j$.

Also, we have $\|e'_i\|_\infty \leq NB$, thus, the bound on the noise term e' follows.

Notice that since the client knows s, they can decrypt $(\mathbf{a}', b')$ by subtracting $\mathbf{a}' \cdot \boldsymbol{\phi}(s)$ from b' then multiplying by t/q and rounding, as it is done for a normal RLWE ciphertext. Moreover, for $i \in [N]$, given encrypted messages m_i, we can obtain $(\mathbf{a}'_i, b'_i)$ encrypting $X^{f(m_i)}$ and add all of them to get $(\mathbf{a}', b')$ that encrypts a polynomial w whose each coefficient w_j is equal to the number of messages that evaluate to j. Thus, if we just want to count considering a univariate function f, then the procedure presented so far is enough.
Notice that to compute $\mathbf{u}_i \cdot \boldsymbol{\Phi}(a_i) \bmod q$ we do not need polynomial multiplications, thus, it can be computed with $\mathcal{O}(N^2)$ additions on $\mathbb{Z}_q$ (e.g., by setting an accumulator $w \in \mathbb{Z}[X]$ for each column of $\boldsymbol{\Phi}(a_i)$ and simply adding an entry $a_{j,\ell}$ from $\boldsymbol{\Phi}(a_i)$ to the coefficient of w defined by the degree of the ℓ-entry of $\mathbf{u}_i$). Therefore, the cost of computing $(\mathbf{a}', b')$ in Equation 4 is $\mathcal{O}(kN^2)$ additions on $\mathbb{Z}_q$. If $D > I$, we can set $N \approx I$ and $kN \approx D$, and obtain the final cost as $\mathcal{O}(|\mathsf{dom}(f)| \cdot |\mathsf{img}(f)|)$.

Algorithm 2: Format-fixing key switching

Input: $(\mathbf{a}', b') \in \mathbb{Z}[X]^{N+1}$ with $b' = \mathbf{a}' \cdot \boldsymbol{\phi}(s) + e' + \Delta X^{f(m)}$ and $\mathsf{ffk} := (\bar{\mathbf{a}}, \bar{\mathbf{b}})$.
Output: An RLWE ciphertext $\mathsf{Enc}_{\bar{s}}(X^{f(m)}) \in \bar{\mathsf{R}}_q^2$

$b'' \leftarrow P \cdot b' - \mathbf{a}' \cdot \bar{\mathbf{b}} \in \bar{\mathsf{R}}$
$b \leftarrow \lfloor b''/P \rceil$ ▷ Division on $\mathbb{Q}[X]$
$a \leftarrow -\lfloor \mathbf{a}' \cdot \bar{\mathbf{a}}/P \rceil$ ▷ Division on $\mathbb{Q}[X]$
Return (a, b).

4.2 Computing multivariate functions

In this section, we show how to transform encryptions of $X^{f_i(m_i)}$ obtained in Equation 4 into regular RLWE ciphertexts. Once we are able to do so, we can then use automorphisms to obtain encryptions of $X^{\alpha_i f_i(m_i)}$ and homomorphic multiplications to get encryptions of $X^{g(m_1,...,m_n)}$, where $g(m_1, ..., m_n) := \sum_{i=1}^{n} \alpha_i f_i(m_i)$. Notice that the image of g can be different from the image of the f_i's, so when we perform this transformation, we want to switch to a new ring $\bar{\mathsf{R}} := \mathbb{Z}[X]/\langle X^{\bar{N}+1}\rangle$, defined with respect to $\bar{N}$ instead of N, where $\bar{N} \geq |\mathsf{img}(g)|$.

Thus, we consider a second secret key $\bar{s} \in \bar{\mathsf{R}}$ and define the *format-fixing* key from $s \in \mathsf{R}$ to $\bar{s} \in \bar{\mathsf{R}}$ as

$$\mathsf{ffk} = [\bar{\mathbf{a}},\ \bar{\mathbf{b}} := \bar{\mathbf{a}} \cdot \bar{s} + \bar{\mathbf{e}} + P\boldsymbol{\phi}(s)] \in \bar{\mathsf{R}}_{Pq}^{N\times 2} \tag{5}$$

for some integer $P = \Theta(q)$. Notice that ffk is essentially a set of N RLWE ciphertexts, each one encrypting one coefficient of the secret key s under the key $\bar{s}$. We use an extended ciphertext modulus Pq and multiply $\boldsymbol{\phi}(s)$ by P so that we can perform the key-switching similarly to [GHS12]. It is also possible to use a gadget matrix with powers of some base B and decompose the ciphertext in base B, as it is done in the original formulation of the key-switching presented in [BV11], however, this multiplies both the running time and the space by a $O(\log q)$ factor.

Our key-switching procedure is described in Algorithm 2. Lemma 6 proves its correctness, i.e., that it outputs a valid RLWE encryption of $X^{f(m)}$ and that it does not increase the noise too much.

Lemma 6. *Let $(a, b) \in \bar{\mathsf{R}}_q^2$ be a pair output by Algorithm 2. Let e' be the noise term of the input ciphertext $(\mathbf{a}', b')$ and $\bar{\mathbf{e}}$ be the noise term of ffk. Then, $b = a \cdot \bar{s} + e + \Delta X^{f(m)}$ and $\|e\| = \Theta(\|e'\| + N\|\bar{\mathbf{e}}\| + N\|\bar{s}\|)$.*

Proof. Since $Pb' = P\mathbf{a}' \cdot \boldsymbol{\phi}(s) + Pe' + P\Delta X^{f(m)}$ and $\mathbf{a}' \cdot \bar{\mathbf{b}} = \mathbf{a}' \cdot \bar{\mathbf{a}} \cdot \bar{s} + \mathbf{a}' \cdot \bar{\mathbf{e}} + P\mathbf{a}' \cdot \boldsymbol{\phi}(s)$, it holds that $b'' = -\mathbf{a}' \cdot \bar{\mathbf{a}} \cdot \bar{s} + Pe' - \mathbf{a}' \cdot \bar{\mathbf{e}} + P\Delta X^{f(m)}$. Thus, for some polynomial ϵ such that $\|\epsilon\| \leq 1/2$, we have

$$\begin{aligned} b := \lfloor b''/P \rceil &= b''/P + \epsilon \\ &= -(\mathbf{a}' \cdot \bar{\mathbf{a}}/P) \cdot \bar{s} + e' - (\mathbf{a}'/P) \cdot \bar{\mathbf{e}} + \epsilon + \Delta X^{f(m)}. \end{aligned}$$

By writing $\mathbf{a}' \cdot \bar{\mathbf{a}}/P = \lfloor \mathbf{a}' \cdot \bar{\mathbf{a}}/P \rceil - \epsilon'$ for some $\epsilon' \in \bar{\mathsf{R}}$ such that $\|\epsilon'\| \leq 1/2$, we have

$$b = a \cdot \bar{s} + \underbrace{\epsilon' \cdot \bar{s} + e' - (\mathbf{a}'/P) \cdot \bar{\mathbf{e}} + \epsilon}_{e} + \Delta X^{f(m)}.$$

Therefore, (a, b) is a valid RLWE encryption of $X^{f(m)}$. Moreover, since $\|\epsilon' \cdot \bar{s}\| \leq (N/2) \cdot \|\bar{s}\|$ and $\|\mathbf{a}' \cdot \bar{\mathbf{e}}/P\| \leq N \cdot \Theta(1) \cdot \|\bar{\mathbf{e}}\|$, we have

$$\begin{aligned} \|e\| &\leq \left\|\epsilon' \cdot \bar{s}\right\| + \left\|e'\right\| + \left\|\mathbf{a}'/P \cdot \bar{\mathbf{e}}\right\| + \|\epsilon\| \\ &= \Theta(N\|\bar{s}\| + \left\|e'\right\| + N\|\bar{\mathbf{e}}\|). \end{aligned}$$

Complexity of format-fixing procedure. The time complexity of Algorithm 2 is dominated by the first line, in which we compute an inner product between two N-dimensional vectors, thus, we need $\mathcal{O}(N)$ products on $\bar{\mathsf{R}}_{Pq}$, and each one costs $\mathcal{O}(\bar{N}\log\bar{N})$ multiplications on $\mathbb{Z}_{Pq}$. Therefore, the total cost is $\mathcal{O}(N\bar{N}\log\bar{N}\log q\log(\log q))$ basic operations.

Putting it all together. We now show the cost of computing an encryption of $X^{g(m_1,\ldots,m_n)}$ where $g(m_1,\ldots,m_n) := \sum_{i=1}^{n}\alpha_i f_i(m_i)$. The basic idea is to produce RLWE encryptions of $X^{f_i(m_i)}$, as described above, then applying the automorphism to obtain $X^{\alpha_i\cdot f_i(m_i)}$, then multiply all the ciphertexts. We recall that when we apply the automorphism $X\mapsto X^{\alpha_i}$ to a ciphertext encrypting some message m under a key $\bar{s}$, we obtain an encryption of $m(X^{\alpha_i})$ under key $\bar{s}(X^{\alpha_i})$, thus, a key-switching is required to obtain again a ciphertext under the original key $\bar{s}$. Moreover, the key-switching is much more expensive than the automorphism itself. Thus, to remove this key-switching, we propose the following:

- Let N_i be the dimension of the ring used to encrypt m_i (the N_i's are not necessarily different).
- Let s_i be the key under which $X^{f_i(m_i)}$ is encrypted (the keys are not necessarily all different).
- For each α_i, compute β_i such that $\alpha_i\cdot\beta_i = 1 \pmod{2\bar{N}}$.
- Using Equation 5, generate a format-fixing key ffk_i from s_i to $\bar{s}(X^{\beta_i})\in\bar{\mathsf{R}}$.

Thus, when we apply ffk_i, we obtain an RLWE encryption of $X^{f_i(m_i)}$ under the key $\bar{s}(X^{\beta_i})$. Then, applying the automorphism produces an encryption of $X^{\alpha_i\cdot f_i(m_i)}$ under $\bar{s}(X^{\alpha_i\cdot\beta_i}) = \bar{s}(X)$, and no key-switching is needed.

To add all the exponents, n homomorphic multiplications are needed. Each one costs $\mathcal{O}(\log(q))$ multiplications on $\bar{\mathsf{R}}_q$, and each of those products require $\mathcal{O}(\bar{N}\log\bar{N})$ products on $\mathbb{Z}_q$, thus, we need $\mathcal{O}(n\bar{N}\log\bar{N}\log q)$ products on $\mathbb{Z}_q$ to combine the encryptions of $X^{\alpha_i f_i(m_i)}$.

Therefore, computing $\mathtt{Enc}_{\bar{s}}(X^{g(m_1,\ldots,m_n)})$ costs

$$\sum_{i=1}^{n}\left(\mathcal{O}(|\mathsf{dom}(f_i)|\cdot|\mathsf{img}(f_i)|)+\mathcal{O}(N_i\cdot\bar{N}\cdot\log\bar{N})\right) + \mathcal{O}(n\bar{N}\log\bar{N}\log q)$$

operations on $\mathbb{Z}_q$ or $\mathbb{Z}_{Pq}$ (they are assumed to cost the same, since $P=\Theta(q)$).

5 Application to privacy-preserving heatmap

In this section, we show how to use our proposed methods to homomorphic evaluate heatmaps and we present practical results obtained with our proof-of-concept C++ implementation, which is publicly available.[†]

5.1 Model for homomorphic heatmap computation

In a heatmap, there is a "map", i.e., a rectangle of dimensions $x_{\max}\times y_{\max}$, and several points (x_i, y_i) within this map. Then, we divide the map in cells of dimension $b\times h$ and our goal is to

[†] `https://github.com/KULeuven-COSIC/Homomorphic-Heatmap`

	$\log N$	sk	σ	$\log q$	λ
Full domain	12	ternary	3.2	109	128
	13	ternary	3.2	130	229
	14	ternary	3.2	145	454
	15	ternary	3.2	157	939
Split domain	9	uniform	3000	32	128
	12	ternary	3.2	64	232

Table 1: Parameters used to evaluate the homomorphic heatmap applications with both our methods. The security level is at least 128 bits and $\lambda > 128$ means that we can still increase $\log q$ to support more noise (with some penalty in the performance).

count how many points lie inside each cell. In our framework, this corresponds to counting how many points evaluate to the same integer in the image of the following function:

$$g(x, y) = \left\lfloor \frac{x}{b} \right\rfloor \cdot \left\lfloor \frac{y_{\max}}{h} + 1 \right\rfloor + \left\lfloor \frac{y}{h} \right\rfloor.$$

Using the notation defined above, we have $g(x, y) = f_1(x) \cdot \alpha_1 + f_2(y)$, where $f_1(x) := \left\lfloor \frac{x}{b} \right\rfloor$, $f_2(y) := \left\lfloor \frac{y}{h} \right\rfloor$, and $\alpha_1 := \left\lfloor \frac{y_{\max}}{h} + 1 \right\rfloor$. Thus, we can compute it in "two levels", by first computing $X^{f_1(x)}$ and $X^{f_2(y)}$, then combining them.

We assume that a client encrypted a database of points (x_i, y_i) and sent it to a server, which stores it in encrypted form. Then, at any moment, the client can send to the server a query $(x_{\max}, y_{\max}, b, h)$ defining a heatmap instance. The server computes the heatmap homomorphically and sends to the client one ciphertext c encrypting a polynomial $\sum_{i=0}^{N-1} a_i X^i$ where each a_i represents the number of points inside the i-th cell. Then, the client can decrypt c using their own secret key sk. We assume the server to be honest-but-curious; in particular, the homomorphic computation steps are public and we do not address circuit privacy in this work.

5.2 Implementation results

In this section, we provide practical results of the homomorphic evaluation of the heatmap with our both methods. To instantiate the FHE schemes, we used the parameters presented in Table 1, which provide at least 128 bits of security, according to the sieve BKZ reduction cost model of the online LWE estimator [APS15]. For the full domain strategy, we always used ternary secret keys, that is, coefficients in $\{-1, 0, 1\}$, and parameter $\sigma = 3.2$ for the discrete Gaussian, then for each $N \in \{2^{12}, 2^{13}, 2^{14}, 2^{15}\}$, we selected $\log q$ that allows us to run our application and also gives us $\lambda \geq 128$. For the split domain strategy, we have one parameter set with $N = 2^9$, which is used to compute the first level of functions, that is, encryptions of $X^{\lfloor x_i/b \rfloor}$ and $X^{\lfloor y_i/h \rfloor}$, and another parameter set with $\bar{N} = 2^{12}$, for the final result $X^{g(x_i, y_i)}$.

Each heatmap instance was defined using $x_{\max} = y_{\max}$ and $b = h$, i.e., a square map divided in grid of squares cells of side b, then we sampled 50 random points (x_i, y_i), encrypted each of them into two ciphertexts and executed both the full and the split domain strategy. In Table 2, we show the running times per point (evaluating a homomorphic heatmap with n encrypted points takes essentially n times the displayed time). All the experiments were run on a single core of an Intel(R)

$x_{\max}$ and $y_{\max}$	b and h	Full domain	Split domain
2^{10}	2^6	0.174 s	0.507 s
2^{11}	2^7	0.174 s	0.507 s
2^{12}	2^8	0.174 s	0.512 s
2^{13}	2^9	0.394 s	0.568 s
2^{14}	2^9	1.0617 s	0.635 s
2^{15}	2^9	2.832 s	0.820 s

Table 2: Time needed for our both methods to process each point (x_i, y_i) of several different heatmap instances.

$x_{\max}$	$X^{f(x)}$	ffk_x	$X^{g(y)}$	ffk_y	Mult.
2^{10}	3	232	3	232	17
2^{11}	7	228	7	227	17
2^{12}	13	226	13	225	17
2^{13}	23	233	25	231	17
2^{14}	58	255	62	255	17
2^{15}	147	243	147	242	17

Table 3: Time spent in each main subrotine of the split-domain strategy, considering the running times presented in Table 2. The time unit is millisecond. The second column refers to the homomorphic computation of $X^{f(x)}$ given $\mathtt{Enc}(X^x)$. The third column corresponds to the format-fixing key key switching applied to $\mathtt{Enc}(X^{f(x)})$. Third and fourth column are the same as the second and third one, but for $\mathtt{Enc}(X^y)$. The final column shows the time of the homomorphic multiplication used to compute an encryption of $X^{f(x)+g(y)}$.

Xeon(R) CPU E5-2630 v2 @ 2.60GHz. In all cases, the number of cells, therefore, the size of the image of $g(x, y)$ is smaller than or equal to 2^{12}, thus, for $2^{10} \leq x_{\max} \leq 2^{12}$, we could use $N = 2^{12}$ for the full domain strategy. However, since $x_{\max}$ and $y_{\max}$ define the domain of f and N has to be larger than or equal to $\max(|\mathsf{dom}(f)|, |\mathsf{img}(f)|)$, we have to increase N and choose it as $N = x_{\max}$ for $x_{\max} \geq 2^{13}$. This explains the slowdown in the running times presented in Table 2. On the other hand, for the split domain, we always used $N = 2^9$ and $\bar{N} = 2^{12}$. As the domain grows, only the parameter $k \approx |\mathsf{dom}(f)|/N$ grows, but it has little impact in the running time. As a result, the split domain method is slower for small domain, but becomes faster as the size of the domain grows.

In tables 3 and 4, we show how the running times of Table 2 are divided in the main steps of each method. We can see that the format-fixing key switchings, which take a non-standard ciphertext and outputs a normal RLWE ciphertext, dominates the running time of the split-domain method. As for the full-domain strategy, the procedures taking most of the execution time are the traces to extract the bits b_i of the evaluated functions and the multiplications to group those bits and move the values to the exponent of X.

$x_{\max}$	$T_i(X) \cdot X^x$	Trace	$X^{f(x)}$	Final mult.
2^{10}	3	60	19	5
2^{11}	3	60	19	5
2^{12}	3	60	19	5
2^{13}	6	140	42	11
2^{14}	17	383	111	23
2^{15}	45	1000	291	47

Table 4: Time spent in each main subrotine of the full-domain strategy, considering the running times presented in Table 2. The time unit is millisecond. The second column displays the time needed to multiply $\texttt{Enc}(X^x)$ by all test polynomials $T_i(X)$ corresponding to the i-th bit of $f(x)$. The third column corresponds to trace operation needed to extract the constant term. The fourth column shows the time needed to multiply the encryptions of the bits of $f(x)$ to obtain $\texttt{Enc}(X^{f(x)})$. Notice that the same operations are computed again to obtain $\texttt{Enc}(X^{g(y)})$ from $\texttt{Enc}(X^y)$ and they cost essentially the same. The final column shows the time of the homomorphic multiplication used to compute $\texttt{Enc}(X^{f(x)+g(y)})$.

References

APS15. Martin Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 2015.

BBH+20. Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. Privately connecting mobility to infectious diseases via applied cryptography. *arXiv preprint arXiv:2005.02061*, 2020.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

BIP+22. Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. Final: Faster fhe instantiated with ntru and lwe. Cryptology ePrint Archive, Report 2022/074, 2022. `https://ia.cr/2022/074`.

BV11. Zvika Brakerski and Vinod Vaikuntanathan. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 505–524, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

CGGI20. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology*, Apr 2020.

CKK15. Jung Hee Cheon, Miran Kim, and Myungsun Kim. Search-and-compute on encrypted data. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *FC 2015 Workshops*, volume 8976 of *LNCS*, pages 142–159. Springer, Heidelberg, January 2015.

DM15. Léo Ducas and Daniele Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

DPP17. Morten Dahl, Valerio Pastro, and Mathieu Poumeyrol. Private data aggregation on a budget. Cryptology ePrint Archive, Report 2017/643, 2017. `https://eprint.iacr.org/2017/643`.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://ia.cr/2012/144`.

GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic Evaluation of the AES Circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 850–867, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

HS20. Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481, 2020. `https://ia.cr/2020/1481`.

INZ21. Ilia Iliashenko, Christophe Negre, and Vincent Zucca. Integer functions suitable for homomorphic encryption over finite fields. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 1–10, 2021.

MDD16. Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics with succinct sketches. In *NDSS 2016*. The Internet Society, February 2016.

NKW15. Muhammad Naveed, Senny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *CCS 2015*, LNCS, pages 644–655. ACM, 2015.

Per21. Hilder Vitor Lima Pereira. Bootstrapping fully homomorphic encryption over the integers in less than one second. In Juan A. Garay, editor, *Public-Key Cryptography – PKC 2021*, pages 331–359, Cham, 2021. Springer International Publishing.

PRZB11. Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, page 85–100, 2011.

Zuc18. Vincent Zucca. *Towards efficient arithmetic for Ring-LWE based homomorphic encryption*. Theses, Sorbonne Université, June 2018.

A Natural BGV/FV solution

Consider the BGV or the FV scheme over the cyclotomic ring $\mathbb{Z}[X]/\langle \Phi_m(X)\rangle$. Let $t = p^r$ be the plaintext modulus, where p is prime. Let $N = \varphi(m)$ be the degree of the cyclotomic polynomial. Consider a function $f : [\![0, D[\![\to [\![0, I[\![$.

Let d be the order of p modulo m and $n := N/d$. be the number of plaintext slots. Then, in each ciphertext, we can encrypt n elements, $(x_1, \ldots, x_n) \in [\![0, D[\![^n$, and each homomorphic operation applies independently to each slots. For example, if $\mathbf{c}_1 := \mathtt{Enc}_{\mathsf{sk}}(x_1, \ldots, x_n)$ and $\mathbf{c}_2 := \mathtt{Enc}_{\mathsf{sk}}(x'_1, \ldots, x'_n)$, then $\mathbf{c} := \mathsf{Mult}(\mathbf{c}_1, \mathbf{c}_2)$ encrypts $(x_1 \cdot x'_1, \ldots, x_n \cdot x'_n)$. This is known as SIMD (single-instruction multiple-data) or batching and allows us to accelerate the homomorphic evaluation, by computing a function in parallel in all the slots with the same number of homomorphic operations needed to evaluate the function one single time. It is also possible to homomorphically rotate the slots. Namely, given an encryption of $(x_1, \ldots, x_n)$ and an integer k prime with m, we can generate an encryption of $(x_k, x_{k+1}, \ldots, x_n, x_1, \ldots x_{k-1})$. We refer to [HS20] for more details.

Notice that the plaintext modulus t has to be larger than the domain, so we have the restriction $t \geq D$.

The first step then, would be to compute f (in each slot via SIMD), obtaining $(f(x_1), \cdots, f(x_n))$. This also implies $t \geq I$.

Then, to count how many elements evaluate to each value in the image, we have to proceed as follows for each $0 \leq i < I$:

- Let $e_i : [\![0, I[\![\to \{0, 1\}$ be a function that outputs 1 if, and only if, its argument is equal to i.
- Compute e_i homomorphically, producing then an encryption of $(e_i(f(x_1)), ..., e_i(f(x_n)))$.
- Rotate and add all the slots, and multiply by the plaintext $(1, 0, ..., 0)$, which yields an encryption $(\sum_{j=1}^{n} e_i(f(x_j)), 0, 0, ..., 0)$. This step requires $\log n$ rotations.

Let's say that evaluating f is done in time T_f, each e_i in time T_e, and each rotation in time T_r, then, the total cost is

$$T_f + I \cdot (T_e + T_r \cdot \log n).$$

But notice that evaluating each e_i is highly non-trivial and is likely to require at least one bootstrapping, which is very costly. In more detail, since e_i implements a comparison with i, the best methods to evaluate e_i require at least $\lfloor \log_2 t \rfloor$ homomorphic multiplications [INZ21], so,

considering that the bootstrapping takes time T_b and each multiplication takes time T_m, we can write $T_e \geq T_m \cdot \log_2 t + T_b$. Thus, the amortized cost, or cost per each instance x_i is at least

$$\frac{T_f + I \cdot (T_m \cdot \log_2 t + T_b + T_r \cdot \log n)}{n} \tag{6}$$

For concreteness, consider that we use the HElib implementation of BGV[‡] to compute the homomorphic heatmap, as in Section 5.2. Typical parameters of BGV for 128 bits of security use N around 2^{15}, so we can fix the set of parameters precomputed by HElib, namely $N = 42799 \approx 2^{15.3}$, ciphertext modulus with 970 bits, and plaintext modulus $t = 2^r$ for any r. With this, there are $n = 2016$ slots. Moreover, on the same machine used in Section 5, we have $T_m = 0.65$ s and $T_r = 0.37$ s. To run the heatmap we can fix $t = x_{\max}$. Notice that as we increase t, the bootstrapping time also increases. Hence, even ignoring the time needed to compute f, that is, setting $T_f = 0$ in Expression (6), the following holds.

- For the first row of Table 2, we have $t = 2^{10}$, $I = 288$, and $T_b = 61$ s. Thus, the amortized time per point (x_i, y_i) is at least 10 seconds, while our full domain solution runs in 0.174 seconds, that is, at least 57 times faster.
- For the last row of Table 2, we have $t = 2^{15}$, $I = 4224$, and $T_b = 72$ s. Thus, the amortized time per point (x_i, y_i) is at least 3 minutes, while our split domain strategy runs in 0.820 seconds, i.e., at least 219 times faster.

[‡] `https://github.com/homenc/HElib/`