TETRIS: Composing FHE Techniques for Private Functional Exploration Over Large Datasets

Malika Izabachène
Independent
Paris, France
malika.izabachene@gmail.com

Jean-Philippe Bossuat
Independent
Lausanne, Switzerland
jeanphilippe.bossuat@gmail.com

Abstract

To derive valuable insights from statistics, machine learning applications frequently analyze substantial amounts of data. In this work, we address the problem of designing efficient secure techniques to probe large datasets which allow a scientist to conduct large-scale medical studies over specific attributes of patients' records, while maintaining the privacy of his model. We introduce a set of composable homomorphic operations and show how to combine private functions evaluation with private thresholds via approximate fully homomorphic encryption. This allows us to design a new system named TETRIS, which solves the real-world use case of private functional exploration of large databases, where the statistical criteria remain private to the server owning the patients' records. Our experiments show that TETRIS achieves practical performance over a large dataset of patients even for the evaluation of elaborate statements composed of linear and nonlinear functions. It is possible to extract private insights from a database of hundreds of thousands of patient records within only a few minutes on a single thread, with an amortized time per database entry smaller than 2ms.

1 Introduction

Large-scale models are used for several tasks such as DNA research, for understanding and generating textual contents, or producing code and many of them are trained on data extracted from millions of gigabytes text words. While these large models are significant across multiple applications, their deployment sets serious privacy challenges as both the user's requests and the model service provider are exposed to sensitive information leak. The user usually wants to protect the privacy of its request to the large model service provider as they may contain sensitive information about the data. And the large model owner usually would like to protect the model intellectual property of his algorithms. In this paper, we consider the scenario where a scientist would like to conduct a medical study over a large dataset of sensitive patients' data owned by a server. In this setting, the statistics run by the scientist are built from an intelligent system model that do contain intellectual property the scientist would like to protect. Our solution makes it possible for the scientist to probe a medical server in order to learn about public health while disclosing as little information as possible to the server. In particular, the response to the scientist request is kept encrypted on the server side and remains unknown to him. Moreover, the scientist does not learn more information on the datasets that what can be inferred from the response to his request. We explicit the function model owned by the scientist as a combination of specific attributes which defines the type of properties the scientist is probing. Such property could search how many selected patients' profiles do have a blood sugar higher than some threshold. In that case, the selection criteria could consider some additional risk factors such as functional neurological disorders or/and blood pressure for example. While the application we focus on raises a medical use case, our framework could still have other application such as credit scoring where a scientist would like to run a private system model to classify individuals' profiles according to confidential attributes.

FHE computation challenges for large models. One solution to the privacy risk of large models deployments is using *fully homomorphic encryption* (FHE), which enables the evaluation of arbitrary functions over encrypted data. Using FHE as a privacy technology in the context of large datasets poses significant performance challenges, as the homomorphic operations become more costly as the number of data grows. Some FHE schemes support SIMD arithmetic which allows to pack data and perform the evaluation on many slots in parallel. However, many privacy-preserving applications involve different types of operations, some of which behave more or less nicely depending on the message encoding format. For example, in the medical use case study we consider in this work, the scientist must perform both nonlinear functions, such as combination of threshold functions, and linear functions such as scoring functions.

The most known FHE schemes encode the message as a scalar or as a polynomial and rely on the Learning with Errors (LWE) or ring LWE (RLWE) assumptions respectively to provide IND-CPA security, meaning that the ciphertext remains indistinguishable to a random element in the ciphertext space. These FHE schemes perform the computation differently depending on the ciphertexts format and the optimization techniques available. LWE-based schemes support very efficient bit-wise computations which allows to manipulate and combine encrypted ciphertexts with high flexibility while RLWE-based schemes generally support SIMD computation and allows to efficiently process a large amount of data at once but leaves less margin to manipulate each ciphertext independently. The authors of [jLLH+21] proposed a method to evaluate both a polynomial and a non-polynomial function but at the cost of introducing approximation errors in the calculation, which is not fully suitable for use cases that do not tolerate computation failures.

From the functional standpoint, the first category of schemes has the advantage of supporting arbitrary circuit evaluations in a more efficient way. The FHEW-like bootstrapping based schemes [DM15, GINX16, CGGI16], to cite a few, falls in this category and has shown

Proceedings on Privacy Enhancing Technologies 2025(2), 23–38
© 2025 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2025-0047

to be able to evaluate discretized arbitrary functions at a slight additional cost compared to the gate bootstrapping, which allows to homomorphically reduce the noise induced by homomorphic computations in a FHE ciphertext. This generalized bootstrapping is called *functional bootstrapping* or *programmable bootstrapping* in the literature. One of the major limitations of the functional bootstrapping technique is that if one wants to keep it efficient, the precision of the message encoding functions that one would like to evaluate needs to be restricted to relatively small sizes. For example, functions with domains larger than 8-to-10 bits of precision become difficult to evaluate in a reasonable amount of time. The second category of schemes does not provide generic methods to evaluate non-polynomial functions and alternatively needs to find efficient tailor-made polynomials to well approximate the function to be evaluated as shown in [LLNK20, LLK$^+$22, CKP22, QZZ$^+$23].

Switching between the encodings in practice. Conversions between FHE ciphertexts format has raised interest over the last years [CGGI17, BGGJ20, CDKS21, jLHH$^+$21, BCK$^+$23] as it allows to switch between LWE ciphertexts and RLWE ciphertexts back and forth, which leaves the possibility to take advantage of both format properties. One of the main procedures used to perform these conversion is homomorphic *packing* (resp. *repacking*) which allows to pack several LWE (resp. RLWE) ciphertexts into one RLWE ciphertext. Packing techniques can provide very low amortized cost for regular homomorphic operations such as additions, multiplications, and bootstrapping operations, compared to repeated invocations on individual ciphertexts, especially when the number of data points to be processed is huge. These are well-suited for schemes whose modulus-to-noise ratio is super-polynomial. Packing has also been proposed in the context of bootstrapping for polynomial modulus-to-noise ratio schemes using different techniques but as these schemes provide extremely efficient bootstrapping on individual ciphertexts, the amortized cost outperforms the regular bootstrapping cost only asymptotically. The authors of [BGGJ20] proposed a framework that allows to support arbitrary functions evaluations and SIMD arithmetic by switching from one format ciphertext representation to another. Their work reports timings of $7s$ to switch from N LWE ciphertexts to one RLWE ciphertext for $N = 2^{12}$ and requires large public switching keys. The work from [CDKS21] reports close to $1s$ to pack a relatively small amount of data but both methods using a different packing method, but [BGGJ20] and [CDKS21] become impractical as the number of input ciphertexts grows.

The packing method from [jLHH$^+$21] takes advantage of the SIMD encoding and reports improved throughput compared to the previous state-of-the-art packing procedures but their method has remained relatively inefficient in term of amortized cost because it requires to evaluate a homomorphic modular reduction. A nice improvement was recently proposed in [BCK$^+$23] for approximate homomorphic encryption. Their experiments report a latency of 10.2s to pack 2^{15} LWE ciphertexts in one RLWE ciphertext compared to 52s to pack 2^{12} LWE ciphertexts using the method from [jLHH$^+$21]. However their method makes use of the LWE assumption over modules (Module LWE), which is not readily available in FHE libraries. Also, once the packing is performed, it remains to combine SIMD arithmetic techniques and arbitrary functions evaluation to enable further homomorphic evaluations over hybrid circuits. At the current state, there is still no fully satisfactory practical solutions that handle different types of operations including arbitrary functions and SIMD arithmetic over large domain without affecting the precision.

Techniques for private functions evaluation over large datasets. Other cryptographic techniques for privacy-preserving computations have been previously proposed. Multi-party computation (MPC) protocols have been proposed to solve the privacy issue in machine learning use cases ([SPT$^+$21, SKS$^+$22, GMS$^+$23, MWA$^+$23, LXY24] to mention a few of them) such as neural network evaluation, privacy-preserving models training, privacy-preserving record linkage (for large database). Most of them focus either on protecting the privacy of the datasets or/and the privacy of the function. The authors of [SPT$^+$21] proposed a solution for neural network training that instantiates an MPC framework using FHE to protect the privacy of the training data, the model and the evaluation. Their experiment processes 60K samples that are distributed among 10 parties. MPC techniques can also be used for two-party computation use cases, but we are not aware of an MPC-based work that solves the challenges addressed by our use case i.e. protecting the model functions while a possibly complex homomorphic computation is processed on a huge amount of data. Another advantage of FHE over traditional MPC techniques is that when ordering and combining homomorphic operations properly, the communication between both parties requires less interactions. In addition, our HE-based solution can be extended to support a partitioned database with only a linear cost in the setup phase (key-generation), which is not the case for traditional MPC-based solutions. The above privacy preserving techniques aim to protect the privacy of the database and/or function during the computation but cannot and are not designed to protect against attacks such as model inversion or individual singling out. In such cases, techniques such as differential privacy can be added on top of the FHE or MPC techniques to mitigate these attacks. However, their use has some limitations as they inevitably incur a trade-off between utility and privacy.

Our contributions.

- In this work, we design a new efficient solution named TETRIS allowing a scientist to probe a dataset containing a huge amount of data owned by a server. Our framework makes it possible for the scientist to privately request attribute functions over the data while maintaining the privacy of his attribute functions. On the other hand, the scientist does not learn more information on the patients' data than what can be deduced from the response from the server.
- We propose an adaptation of the large domain homomorphic evaluations of arbitrary functions technique from [IIMP22, BI24] in the case where the function is encrypted and the large dataset remains in the plaintext domain. In other words, the database owner processes the computation for the scientist who owns the function, and the large database doesn't need to be sent at any point. At the end, the database owner does not learn any information about the function that it evaluates while the scientist learns nothing except what

could be inferred from the result and the computation circuit if the latter is known to him. Adapting the techniques from [IIMP22, BI24] allows us to overcome the domain size limitation constrained by using the functional bootstrapping. Our method enables to support the evaluation of arbitrary functions over large domains in an efficient way while keeping the functions private. In our framework, the arbitrary functions are instantiated to enable the homomorphic evaluation of scoring functions parametrized by the attributes. Our methodology for the evaluation of arbitrary function is introduced in Section 4.3 and its specialization to homomorphic scoring function evaluation is presented in Section 5.4.

- In the homomorphic thresholds evaluation phase, we make use of several homomorphic building blocks, including ring repacking, ring merging and ring switching, that we carefully combine to be able to merge and switch to a larger ring dimension. By doing so, we manage scheme switching via bootstrapping, and the evaluation of two consecutive nonlinear functions. In our framework, these nonlinear functions are specialized as a local and global threshold applied on linear combinations of encrypted inputs. Based on the approach from [LLNK20] to evaluate the sign and the step functions over encrypted real inputs, we show how to evaluate threshold functions over encrypted discrete values with approximate homomorphic encryption. All these homomorphic operations are processed by thoroughly choosing appropriate dimensions and moduli to not affect precision, efficiency and security.
- We implement a real-world use case that showcases how the above techniques can be combined together, along with approximate homomorphic encryption, to provide an efficient private database exploration framework, answering the following question: are there at least x elements in the database that match at least y-out-of-z private criteria? We show that such a question can be answered within a few minutes on a single thread for a database of hundreds of thousands of entries with an amortized cost of only $2ms$.
- We tune the parameters involved in the noise growth of the underlying FHE operations, the precision, the correctness and the security of the homomorphic operations to propose an efficient implementation of each building block. We provide an open-source implementation for all aspects of this work using the Lattigo library [EL23].

Application to private database exploration. Assume a scientist would like to privately explore a database of size $p \times h$ to get a binary answer on whether this database contains a minimal set of items meeting private criteria of his choosing. At a high level, this can be done by (i) evaluating a private scoring function on each of the h attributes of each p entry of the database, (ii) summing the encrypted scores to produce a single score for each p entries of the database, (iii) evaluating a private threshold on the summed scores that returns 1 if the score is larger than a private threshold, else 0 and (iv) summing all binary values from the previous step to produce an encrypted count of how many entries match the private criteria and (v) evaluating a second private global threshold that returns 1 if enough entries meet the private criteria, else 0.

The private criteria are sent to the database owner as RLWE ciphertexts of degree 2^{12} encrypting the polynomial representation of the scoring functions, using the evaluation of arbitrary function technique we will review in Section 4.3. The database owner then homomorphically uses the repacking algorithm to repack the results in RLWE ciphertexts (steps (i) and (ii)). This enables to densely pack up to 2^{12} scores in a single RLWE ciphertext. Since an RLWE ciphertext of degree 2^{12} only allow for a modulus of up to 108 bits to be secure, they are merged into RLWE ciphertexts of degree 2^{16} to enable bootstrapping to a larger modulus (e.g. 1540 bits) enabling to continue performing further homomorphic evaluations. This merging step also reduces the number of ciphertexts by a factor of $2^{16}/2^{12} = 16$, with each ciphertext encrypting up to 2^{16} scores. Finally, the database owner bootstraps and homomorphically encodes theses large degree RLWE ciphertexts to enable encrypted SIMD arithmetic and leverage batching when evaluating the threshold functions on the encrypted scores (steps (iii), (iv) and (v)).

Relations to existing works on Private Function Evaluation (e.g., via universal circuits). Private Function Evaluation (PFE) protocols enables two parties P_A and P_B owning respective private inputs x_A and x_B to learn $f(x_A, x_B)$ where only one of the two parties knows the function f. Several papers ([Val76a, KS08, KS16, MS13, LYZ$^+$21, AGKS20] for example) have focused on constructing PFE for arbitrary polynomial-size functions. A well-known approach consists of relying on secure evaluations of universal circuits (UC), which takes a circuit C of size n from a sequence of circuits $\{\mathsf{C}_n\}_{n\in\mathbb{N}}$ and an input x and outputs $\mathsf{C}_n(C, x)$ which corresponds to the evaluation of C on x. In that case, the PFE protocol inherits from the properties of the underlying secure function evaluation protocol used to evaluate C_n on private inputs C and x. Valiant [Val76a] proposed a size-optimal UC-based construction of size $\Theta(n \log n)$ that can simulate any Boolean circuit of size (less than or equal to) n. Due to this optimal bound, as the circuit size increases, the complexity of UC-based PFE protocols becomes large as well. Katz and Malka [KM11] proposed a passively secure PFE protocol with linear complexity. An implementation of the PFE protocol from [KM11] proposed by Holz et al. in [HKRS20] shows that HE-based PFE outperforms UC-based PFE in communication for all circuit-size and in computation starting from circuit with a few thousand gates. We make a brief comparison of the performance of our protocol with the one of [HKRS20] in Section 6.6.

Organization. Section 2 provides the necessary technical backgrounds to understand the homomorphic building blocks composing our use case. Section 3 introduces the system model for TETRIS and presents in details the homomorphic ring packing, ring merging and ring switching along with the HE-based PFE we use to handle computations over large datasets. Section 5 shows how to combine all the homomorphic ring operations to build TETRIS, our solution for private functional exploration of large datasets. Section 6 details the performance of our construction. Finally, Section 7 concludes the paper.

2 Backgrounds

2.1 Notations

We denote single elements (polynomials or numbers) in italics, e.g., a, and vectors of such elements in bold, e.g., $\boldsymbol{a}$. The i-th position of the vector $\boldsymbol{a}$ or the degree-i coefficient of the polynomial a will be denoted $a[i]$. The interval from the i-th to to j-th coefficients will be denoted 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 polynomial convolution. 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 by $\leftarrow \chi_d^n$ the act of sampling a vector (or polynomial) of size n from the distribution d (we omit n when $n=1$). Unless otherwise stated, logarithms are in base 2.

2.2 Plaintexts and Ciphertexts Domain

The N-th cyclotomic polynomial ring modulo Q will be denoted $\mathcal{R}_Q^N[X] = \mathbb{Z}_Q[X]/(X^N+1)$ and defines the messages space and ciphertexts space. Elements of $\mathcal{R}_Q^N[X]$ 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, i.e. lie in $[-\lfloor Q/2\rfloor, \lfloor Q/2\rfloor)$. To keep notations concise, the variables N, Q and/or X will be omitted when denoting $\mathcal{R}$ when these do not need to be defined.

The integer 5 is a generator of order $N/2$ modulo $2N$, and along with the generator -1, it spans $\mathbb{Z}_{2N}^*$. Therefore the set of automorphisms $\phi_i : X \mapsto X^i$ for $i \in (5^{j\in\mathbb{Z}_{N/2}}, -1)$ forms a group over $\mathrm{Gal}(K/\mathbb{Q})$ under composition, i.e. $\phi_i(a)+\phi_i(b) = \phi_i(a+b)$ and $\phi_i(a)\cdot\phi_i(b) = \phi_i(a\cdot b)$ for $a,b \in \mathrm{R}$.

The ring $\mathcal{R}^N[X]$ contains multiple subrings with interesting properties, one of them being $\mathcal{R}^N[Y]$ for $Y = X^2$. Since $\mathcal{R}^N[Y] \cong \mathcal{R}^{N/2}[X]$ we have $\mathcal{R}^N[X] \cong \mathcal{R}^{N/2}[X] + X * \mathcal{R}^{N/2}[X] \mod (X^2+1)$. This relation can be generalized to $Y = X^{N/n}$ for n a power-of-two smaller than N giving us the following isomorphism rule $\forall n|N$:

$$\mathcal{R}^N \cong \overbrace{\mathcal{R}^{N/n} \otimes \cdots \otimes \mathcal{R}^{N/n}}^{n}$$

The ring $\mathcal{R}^N[X]$ allows to encode polynomials in the ring (a.k.a in the *coefficients*) or in the canonical embedding (a.k.a in the *slots*). And messages can be homomorphically switched between these two encodings. And messages can be homomorphically switched between these two encodings and these follow different plaintext arithmetic. We detail in Appendix 8.1 the encoding in both plaintext domains.

2.3 FHE Ciphertexts

Let s be polynomial in $\mathcal{R}^N[X]$ with coefficients sampled from a uniform distribution over $\{-1,0,1\}$ and with exactly h non-zero coefficients. An RLWE ciphertext of a polynomial message m under secret key s is generated as $(c_0,c_1) = (-as+e+m, a)$ where a's coefficients are uniform over $U(\mathbb{Z}_Q)$ and e's coefficients are sampled from a discrete Gaussian distribution with standard deviation σ. An RLWE ciphertext generated as such follows a distribution generated as $\mathcal{D}_{\mathsf{RLWE}}^{N,Q,h,\sigma}$. Note that a fresh RLWE encryption of zero defines a polynomial in s of degree 1 that can be written as $\mathcal{P}_Q(s) = c_0 + c_1 * s = e$ with $||e||_\infty$ small. After a few homomorphic operations, an RLWE encryption of zero defines a polynomial of degree k that will be denoted $\mathcal{P}_Q^k(s)$, if $k \neq 1$. And an (resp. the set of) RLWE encryption of m of under secret key s will be denoted $\mathsf{RLWE}_s(m)$ or $\mathsf{RLWE}(m)$ when the secret key doesn't need to be specified.

2.4 Homomorphic Operations

2.4.1 Keyswitching. keyswitching is at the core of the homomorphic properties of RLWE-based encryption and used as a building block in a number of homomorphic operations most notably, homomorphic multiplication, morphism and ring splitting or ring merging. It makes use of an additional modulus P which allows to control the noise growth.

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

2.4.2 Advanced Homomorphic Operations for Scheme Switching. we now describe how the keyswitching operation SwitchKey is used to perform some of the key homomorphic operations and give further details in Appendix 8.2 on how these operations are processed using keyswitching.

- Relinearization: the multiplication between two RLWE ciphertexts $\mathcal{P}(s)$ returns a new RLWE ciphertext $\mathcal{P}^2(s)$, and the degree will further increase if we perform additional multiplication with non-plaintext operands.
- Automorphism: an automorphism ϕ applied on an 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 $\mathsf{SwitchKey}(\mathcal{P}^{[1]}(\phi(s)), \mathsf{swk}_{\phi(s)\to s}) + \mathcal{P}^{[0]}(\phi(s))$;
- Ring Splitting: it is possible to split an RLWE ciphertext $\mathcal{P}(s) + m$ with $m, s \in \mathcal{R}^N$ into two ciphertexts of half the dimension $(\mathcal{P}(s') + m_0, \mathcal{P}(s') + m_1)$ with $m_0, m_1, s' \in \mathcal{R}^{N/2}$ such that $m(X) = m_0(Y) + X * m_1(Y)$ for $Y = X^2$;
- Ring Merging: it is possible to merge two RLWE ciphertexts $d_0 = \mathcal{P}(s') + m_0$ and $d_1 = \mathcal{P}(s') + m_1$ with $m_0, m_1, s' \in \mathcal{R}^{N/2}$ into an RLWE ciphertext of twice the dimension $\mathcal{P}(s) + m$ with $m, s \in \mathcal{R}^N$ such that $m(X) = m_0(Y) + X * m_1(Y)$ for $Y = X^2$.

2.5 Approximate Homomorphic Encryption

The CKKS scheme by Cheon et al. [CKKS17] is an RLWE homomorphic-encryption scheme that enables SIMD arithmetic over $\mathbb{C}^n$. Since its introduction, this scheme has grown in popularity, as it is currently the most efficient scheme for performing encrypted fixed-point arithmetic over complex numbers.

The setup of the scheme is done by instantiating a secure RLWE distribution $\mathcal{D}^{\mathsf{RLWE}}_{N,Q,h,\sigma}$, and a fresh encryption of m is of the form $\mathcal{P}_Q(s) + m$. This scheme is said to be approximate because the error is mixed with the message such that when we evaluate the decryption of $\mathsf{RLWE}_s(m)$ we obtain $m+e$ which is an approximation of m.

The scheme makes use of the canonical embedding to enable SIMD operations over $\mathbb{C}^n$ in the encrypted domain. We provide a full description in Appendix 8.1 on how the encoding works in the canonical embedding. The basic operations (addition, multiplication, rotation, conjugation) are sufficient to evaluate linear transformations and polynomial functions over $\mathbb{C}^n$. Such linear transformations include the ability to homomorphically switch between the *slots* and *coefficients* encoding.

The encoding procedure uses fixed-point arithmetic with a scaling factor Δ to emulate $\mathbb{R}$ on $\mathbb{Z}$. The consequence is that the multiplication of two messages scaled by Δ_0 and Δ_1 respectively returns a new message scaled by $\Delta_2 = \Delta_0\Delta_1$. It is easy to see that this scaling factor can have an exponential growth if not controlled, thus a *rescaling* operation is required after each multiplication to ensure a linear growth. This rescaling operation is straight-forward: truncate the lower bits of the ciphertext by evaluating $\lfloor 1/\Delta \rceil$ on the coefficients. However, given a ciphertext $\mathcal{P}_Q(s) + m$, the rescaling procedure will return a new ciphertext in the ring $\mathcal{P}_{Q/\Delta}(s) + \lfloor m/\Delta \rceil$ (we assume that Δ is a factor of Q). After i rescaling operations, we will get a ciphertext with a modulus $\Delta < Q/\Delta^i < \Delta^2$ and no further *rescaling* operation can be carried out, therefore no more multiplication. Such ciphertext is said to be *exhausted* and then there are two options available: decrypt or homomorphically re-encrypt the ciphertext to a modulus $Q' > \Delta^2$ such that at least one additional rescaling (i.e. multiplication) can be evaluated before it becomes exhausted again. This homomorphic re-encryption is called *bootstrapping* and it enables the evaluation of circuits of arbitrary depth under encryption.

2.6 Bootstrapping for Approximate Homomorphic Encryption

For an RLWE distribution $\mathcal{D}^{N,Q,h,\sigma}_{\mathsf{RLWE}}$, $X = Y^{N/n}$, a polynomial message $m \in \mathcal{R}^n$ and Δ a scaling factor, the bootstrapping procedure of the CKKS scheme [CHK+18] takes as input an *exhausted* ciphertext $\mathcal{P}_q(s) + m(Y)$ with $\Delta < q < \Delta^2$ and returns a new ciphertext $\mathcal{P}_{Q'}(s) + m(Y)$ for $\Delta^2 < Q' < Q$, enabling further homomorphic evaluations. The procedure consists of the following four steps: ModRaise, CoeffsToSlots, EvalMod, and SlotsToCoeffs which are recalled in Appendix 8.3.

3 System Model

3.1 Use case Context

Our setting is the following: a scientist would like to conduct a horizontal study requiring patients' datasets by applying a very specific combination of attributes, which the scientist would like to keep private. We consider a two-party protocol between the database owner holding the patients' data and the scientist, who holds a private function of the form defined at the end of this subsection. To be funded, the scientist would like to assess if there are enough patients meeting the selected research criterion. During a first phase called *homomorphic private function evaluation*, the database owner applies the scientist's encrypted criteria over the patient's data. This phase computes an encrypted result which is the evaluation of the encrypted attribute functions over the patient's dataset. The outcome of this homomorphic computation provides a preliminary validation over the patients' datasets by computing a score per patient with respect to some metrics such as the age, the blood pressure or the blood sugar level. Then in a second phase called *homomorphic thresholds evaluation*, the database owner evaluates two consecutive threshold functions, but privately parameterized by the scientist, over an encrypted batch of patients' data built from the output of the first phase. Overall, the computation boils down to answering the following question:

Given a database of p patients and a set of h criteria, are there at least $p' \leq p$ patients meeting at least $h' \leq h$ criteria?

We will show how this question can be answered efficiently for a database of hundreds of thousands of patients with dozens of attributes while still ensuring privacy with respect to the database owner and the scientist.

3.2 Threat Model

Both the database owner and scientist are assumed to be *honest-but-curious*, meaning that both follow the protocol honestly but still try to infer as much as possible from the each other's private inputs (the database and the private functions respectively). To model this setting, we consider the three following security properties:

1) Privacy with respect to the database owner: this property captures the fact that the database owner does not learn the response output to the question.
2) Privacy with respect to the scientist: this property captures the fact that the scientist does learn more information on the patients' datasets than what can be inferred from the database owner output.
3) Bounded extractable information: since the scientist has full control over the question asked to the database we need a way to quantify the amount of information extracted from the database by the scientist per request after decryption. Assuming the database owner evaluates the function honestly, this notion aims to quantify the amount of information that is sent back to the scientist, even in the case where the latter sends a malicious function. In our case, the extractable information is defined as the domain size of the decrypted response i.e. number of bits sent back by the database owner. Hence in our framework, the extractable information is bounded by 1 bit per query.

Potential leak regarding the selected attributes: in order to achieve a concretely-efficient solution for multivariate functions, the scientist uploads an attributes-selection plaintext matrix which is used to select and/or concatenate sets of attributes together. In the case where this matrix contains exactly the attributes that will be used, the database owner learns which attributes are being grouped, but not the combining function. However, the scientist can include dummy attributes and build the private function to not take these dummy attributes into account. In that case, the database owner learns the sets of attributes that are used but not necessarily which of them exactly. In particular, if the scientist selects all the attributes, there is no leakage regarding the attributes used for univariate functions. Quantifying how much information is leaked will highly depend on the attributes-selection matrix being sent by the scientist. As the database owner will not learn the final output at all, property (1) is expressed as an all-or-nothing privacy property. This means that in our framework, we consider the database owner can learn which attributes are being grouped or selected, but not the combining function and its output.

Potential leak by using a malicious function: as discussed above, the scientist might attempt to send a malicious function in order to learn more information on the patients dataset than what the database owner would allow. In our use case, the database owner processes the patients' data in two steps. In the first phase (homomorphic private function evaluation), the patients' data are processed in the cleartext domain by applying the encrypted function sent by the scientist and the output is encrypted. Considering the form of the statement tested by the scientist (as expressed at the end of subsection 3.1), the scientist explicitly asks for a threshold computation on a private (scoring) function. Regardless of the parameters of the thresholds evaluated by the database owner, the amount of information is bounded to 1 bit per request (the output domain size of the threshold function). If the scientist acts dishonestly by sending a malicious encrypted function in the first phase, we argue that the database owner still has full control on how much information is returned to the scientist for each request.

Dealing with multiple malicious requests: In our framework, we can not avoid attacks from a scientist who would use high dimensional requests to extract information that it should not, for example by using multiple requests to try to infer if a specific individual is part of the database. Such attack becomes unavoidable as soon as any amount of information about the content of the database is returned to the scientist. In that case, the amount of information extracted by the scientist relates to the number of requests that would be needed to achieve the desired outcome. However, limiting the throughput of information that is returned by request to the scientist, or bounding the number of request that a scientist can make, in combination with differential privacy techniques can help mitigate this kind of attacks.

We now give further details on the protocol interactions between the scientist and the patients database owner.

3.3 TETRIS High-Level Overview

The protocol is divided in two phases, a *homomorphic private function evaluation phase* and a *homomorphic thresholds evaluation phase* which work as follow:

(1) *Homomorphic private function evaluation phase.* The inputs to the homomorphic private function evaluation phase are:
- Scientist: an attributes-selection plaintext matrix M of size $h \times m$ and a list of m encrypted scoring functions over the features represented as a vectors of RLWE encryptions.
- Database owner: a database P represented as a $p \times h$ matrix, where p is the number of patients and h is the number of features. We will denote by $P[i]$ the i-th entry of the database (i.e. the matrix row) by $P[i][j]$ and its j-th feature.

Throughout this phase, the goal of the attributes-selection plaintext matrix is to enable features selection, but it can also be used to produce a linear combination of features. The scientist starts by sending the encrypted functions together with the attributes-selection plaintext matrix to the database owner. At the end of the homomorphic private function evaluation phase, the database owner obtains the encrypted evaluation of the functions applied to the database.

(2) *Homomorphic thresholds evaluation phase.* The database owner homomorphically combines the patients' scoring results and homomorphically evaluates a first threshold (local threshold), which corresponds to answering the question *does* $P[i]$ *meets* $h' \leq h$ *criteria* (for all i)? The database owner then aggregates the results and homomorphically evaluates a second threshold (global threshold), which corresponds to answering the question *are there at least* $p' \leq p$ *patients fulfilling the first question?*

4 Building blocks

In this section, we introduce the different building blocks that are required by our construction.

4.1 Ring Repacking

Several (re)packing algorithms were proposed in the literature [CGGI17, CDKS21, jLHH+21, BGGJ20, KDE+21, BCK+23], some of them are proposed to pack a list of n encryptions of scalar messages x_i (LWE encryptions) to an RLWE encryption of $\sum_{i=0}^{n-1} x_i \cdot X^i$, and the procedure is called *ring packing* in that case. Ring repacking refers to the operation that maps a list of RLWE encryptions of some polynomial messages to an RLWE encryption whose message coefficients are taken from the input list of messages coefficients. In our work, we will rely on the iterative version of the *ring repacking* algorithm proposed in [CDKS21, KDE+21], that we recall in Algorithm 1. This iterative version was recently used in [BI24] in the context of homomorphic evaluation of (public) arbitrary functions over encrypted inputs. The goal of the *ring repacking* algorithm is to repack n RLWE ciphertexts encrypting each a polynomial with x_i in its constant coefficient for $i \in [0, n-1]$ into an RLWE ciphertext encrypting $\sum_{i=0}^{n-1} x_i \cdot X^i$.

For sake of simplicity, we take the number of input ciphertexts to be equal to N i.e the degree of the polynomial ring, but the algorithm can be easily generalized to any number of inputs less than N just by taking zero values for the remaining coefficients and if $n > N$ we can split the n ciphertexts into batches of up to N ciphertexts and evaluate a repacking per batch. Regarding its complexity, Algorithm 1 requires N homomorphic automorphisms evaluations and $\log N$ distinct switching keys.

Algorithm 1: RING REPACKING of N RLWE encryptions into one RLWE

```
Inputs: $N$ RLWE ciphertexts $c_1 = (a_0, b_0), \dots, c_n = (a_{N-1}, b_{N-1})$
where $(a_i, b_i)$ is an encryption under secret key $s$ of a
polynomial $m_i(X)$ whose constant coefficient is $m_i[0]$; the
keyswitching keys $\mathsf{swk}_{\phi_g(s)\to s}$ for $g = 5^{2^{i-1}}$ and
$i \in [0, \log N - 1]$ for homomorphically evaluating the
automorphisms $\phi_g$.
Output: an RLWE encryption of $m'(X) = \sum_{i=0}^{N-1} m_i[0] \cdot X^i$.

for $i \leftarrow 0 < N$ do
    $c_i \leftarrow N^{-1} c_i$ // pre-multiplication by $1/N \mod Q$
for $i \leftarrow 0 < \log N$ do
    $t = N/2^{i+1}$ // number of remaining steps
    if $i = 0$ then
        $g \leftarrow 2N - 1$
    else
        $g \leftarrow 5^{2^{i-1}} \mod 2N$
    for $j \leftarrow 0 < t$ do
        $c_j \leftarrow c_j + c_{j+t} \cdot X^t + \phi_g(c_j - c_{j+t} \cdot X^t)$
return $c_0$
```

LEMMA 1. *Let $c_1 = (a_1, b_1), \dots, c_n = (a_N, b_N)$ be encryption of $m_1(X), \cdots, m_N(X)$. Algorithm 1 takes as inputs the c_i and returns an encryption of $\mu(X) = \sum_i \mu_i \cdot X^i$ such that $\mu_i = m_i[0]$.*

We refer to [CDKS21, proof of Algorithm 2] for the proof of Lemma 1. Regarding the noise growth induced by Algorithm 1, [CDKS21] does not provide an analysis, however its behavior is predictable and can easily be explained: we can notice that at each of the $\log N$ steps of the algorithm, pairs of ciphertexts have their even coefficients that double and their odds coefficients that vanish before being added together. Additionally, one of the ciphertexts goes through a keyswitching operation. Therefore, given e_{ct}, the noise output of the previous step and e_{ks} the noise of the keyswitching operation, the new noise after each iteration becomes $2e_{\mathrm{ct}} + e_{\mathrm{ks}}$. Over $\log N$ steps, this gives us an estimated noise growth of $\approx N(e_{\mathrm{ct}} + e_{\mathrm{ks}})$. Now since the initial ciphertext coefficients, and thus the noise, have been pre-scaled by N^{-1}, the final noise is actually $\approx e_{\mathrm{ct}} + Ne_{\mathrm{ks}}$. The noise growth is therefore independent of the initial noise and depends solely on the additive noise of the keyswitching operation, which can be parameterized through the gadget decomposition to achieve the desired bounds.

4.2 Ring Merging and Schemes Switching

Ring merging is employed in *scheme-switching* which makes use of the CKKS bootstrapping which requires a large ring degree. We first explain the principle behind *scheme-switching* and then where ring merging comes into play. To enable an efficient *ring repacking* procedure, the functions that will be evaluated are encrypted in a ring of small degree n and with a small modulus Q_0: $\mathcal{R}^n_{Q_0}$. So the output of the ring repacking is in the coefficients domain and in the ring $\mathcal{R}^n_{Q_0}$. To enable SIMD arithmetic and further homomorphic operations, we need to homomorphically switch the values from the coefficients' domain to the slots domain and we need to increase the size of the modulus. For the sake of completeness, the switching operations in the plaintext domains are detailed in Appendix 8.1. This can be done using the CKKS bootstrapping, however to be able to evaluate the bootstrapping circuit, with a large enough residual homomorphic capacity after its evaluation, we must increase the modulus from Q_0 to Q_ℓ, and as a consequence the ring degree from n to N to maintain the same security. To switch from a ring degree of n to N, we rely on *ring merging*.

The goal of the *ring merging* procedure is to pack a batch of N/n ciphertexts in $\mathcal{R}^n_Q$, each encrypting $\sum_{j=0}^{n-1} x_{i,j} X^j$ for $i \in [0, N/n]$ in a single ciphertext in $\mathcal{R}^N_Q$ encrypting $\sum_{i=0}^{N/n-1} \left(\sum_{j=0}^{n-1} x_{i,j} \cdot X^{N/n}\right) \cdot X^i$. In other words, we can merge a list of N/n ciphertexts of degree n into a single RLWE ciphertext of degree N. Concrete values for *scheme switching* operations will be given in Section 6.

4.3 Large-Domain Private Function

The goal of this building block is to evaluate an encrypted arbitrary function $f : \mathbb{R}_{[a,b]} \to \mathbb{R}_{[c,d]}$ on one or several plaintext values in $[a, b]$. There are several methods in the literature that show how to evaluate arbitrary functions. However, as discussed in Section 1 most of them are constrained in the number of inputs to keep the evaluation method sufficiently efficient in practice. Our method is based on an adaptation of a solution proposed by [IIMP22, BI24], which outperforms known homomorphic LUT evaluation techniques for domains as large as 2^{14} and higher while providing sufficient precision. While this method is not composable (i.e. the method cannot be recursively invoked), it still sufficient for our purpose) and it supports large domain size in an efficient manner; by way of illustration, [BI24] reports a two-to-three orders of magnitude improvements compared to previous evaluation methods. In their case, the inputs of the large domain function are encrypted and the function to be evaluated is encoded in the plaintext domain. We consider the dual scenario where the functions to be evaluated are encrypted and the inputs are in the plaintext domain. We review below the interactions between the scientist and the database owner, where the latter evaluates $f : \mathsf{Dom}_f \mapsto \mathsf{Img}_f$ (defined in the encrypted domain) on a set of points in $I \subseteq \mathsf{Dom}_f$ (in the plaintext domain):

(1) Each point $i \in I := \{i_0, i_1, \cdots, i_{I-1}\}$, where I is held by the database owner in our case, is encoded in the exponent as X^i;

(2) The scientist defines a polynomial representation of the function f as $\mathbf{u}_f := f(0) - f(N-1) \cdot X - f(N-2) \cdot X^2 - \cdots - f(1) \cdot X^{N-1}$ that is encrypted as an RLWE ciphertext, denoted $\mathbf{U}_f \in \langle \mathcal{P}_Q(s) + \mathbf{u}_f \rangle$.
For each element encoded as X^i by the database owner, it holds that:
$$X^i \cdot \mathbf{u}_f = X^i \cdot \mathbf{U}_f \in \langle \mathcal{P}_Q(s) + (f(i) \cdot X^0 + \star) \rangle$$
which encrypts a polynomial whose constant coefficient is equal to $f(i)$. Using the ring-repacking procedure, the database owner maps the RLWE encryptions of $f(i_0) \cdot X^0 + \star$, $f(i_1) \cdot X^0 + \star, \cdots$ and $f(i_{I-1}) \cdot X^0 + \star$ to an RLWE encryption $\mathbf{U}_f$ of $f(i_0) + f(i_1) \cdot X + f(i_2) \cdot X^2 + \cdots + f(i_{I-1}) \cdot X^{I-1}$;

(3) The scientist receives and decrypts $\mathbf{U}_f$ to obtain $f(i_0)+f(i_1)\cdot X+f(i_2)\cdot X^2+\cdots+f(i_{I-1})\cdot X^{I-1}$. He parses the result and obtains the consecutive evaluations on the target points in the decrypted polynomial coefficients.

Complexity. Using this methodology, the database owner homomorphically evaluates $2|I|$ plaintext-ciphertext products and $|I|$ automorphisms. At the second step, the database owner can alternatively send $|I|$ RLWE encryptions of $f(i)\cdot X^0+\star$ for each $i\in I$ and the scientist would decrypt each of them independently to retrieve $f(i)$ for each $i\in I$. However, this method has two drawbacks. First, it will not permit further computations on the encryption of $f(i_0)+f(i_1)\cdot X+f(i_2)\cdot X^2+\cdots+f(i_{I-1})\cdot X^{I-1}$ without using repacking. Second, it would incur an additional $|I|$ factor in the communication complexity from the database owner to the scientist side. Instead, using ring repacking enables the database owner to send back only one RLWE ciphertext, this is why we implement the first method.

Security properties. In our use case, the points are processed in clear by the database owner who applies the encrypted function sent by the scientist, during the homomorphic private function evaluation phase. As discussed in Subsection 3.2, while the database owner knows which of the attributes are used, it does not learn the function itself (ie. how the attributes are grouped) during this phase since both the function and its output are encrypted. Then, in the homomorphic thresholds evaluation phase, the database owner applies two thresholds privately parametrized by the scientist over an encrypted selected dataset. The privacy property with respect to the scientist is guaranteed if the underlying RLWE encryption scheme is semantically secure which holds under the RLWE assumption. We provide concrete parameters for our use case in Section 6. In the homomorphic thresholds evaluation phase, the database owner controls the information that is computed over the encrypted selected patients' dataset. Assuming the RLWE encryption scheme is correct, the scientist does not learn more than what the evaluation of the function over a selected patient's dataset provides; this means that after decryption the scientist does not learn more that 1 bit of information per request. As defined by the bounded extractable information property, the amount of information is quantified per request as we cannot avoid attacks where the scientist tries to infer more than 1 bit by combining evaluation results of many malicious functions requests. Hence, the privacy property with respect to the database owner and the bounded extractable information per request property are both guaranteed.

5 TETRIS Specifications

We first give the concrete setting for the private database exploration use case and detail how the core functions are defined and implemented in the encrypted domain. We then present the circuit's structure in TETRIS and its different steps.

5.1 Dataset, Attributes Selection & Criteria

5.1.1 Dataset. We generate a synthetic dataset of $p=2^{19}$ entries and $h=16$ features. Features are drawn from a Gaussian distribution of standard deviation $\sigma=1$ and mean $\mu=1$ and are bounded to the interval $[0,2)$.

5.1.2 Attributes Selection & Criteria. The scientist is assumed to provide an attributes-selection plaintext M matrix $h\times m$. The primary goal of this matrix is to enable features selection, but it can also be used to produce a linear combination of attributes before evaluating a scoring function. In our practical case, we use 16 scoring functions of the form:

$$f:\mathbb{R}_{[a,b)}\rightarrow\mathbb{Z}_{10},$$

where we take $a=0$ and $b=2$.

5.2 Scheme Switching Operations

5.2.1 Ring Repacking. Each evaluation of the private function $\mathsf{RLWE}(\boldsymbol{u}_{f_j})$, for $j\in[0,m-1]$ on a value x returns an RLWE ciphertext encrypting a polynomial whose constant coefficient is the value $f_j(x)$. We recall that the goal of this building block is to repack n RLWE ciphertexts, for $i\in[0,n-1]$ encrypting $\mathsf{RLWE}(\boldsymbol{u}_{f_i})$ in their constant coefficient into a single RLWE ciphertext encrypting $\sum_{i=0}^{n-1}f_j(x_i)\cdot X^i$, producing a densely packed ciphertext in the ring $\mathcal{R}^n_{Q_0}$ for some possibly distinct $j\in[0,m-1]$. In our case, the f_j's are the final scoring functions for each row. If $p<n$ (we recall that p is the number of rows in the database), we can just take zero values for the remaining coefficients. This homomorphic operation is done by using the repacking technique described in Algorithm 1.

$$\begin{tikzcd}(\mathsf{RLWE}(\mathsf{score}_{P[0]}),\ldots,\mathsf{RLWE}(\mathsf{score}_{P[n-1]}))\in\mathcal{R}^n_{Q_0} \arrow[d, "\text{Repack}"] \\ \mathsf{RLWE}\left(\sum_{i=0}^{n-1}\mathsf{score}_{P[i]}X^i\right)\in\mathcal{R}^n_{Q_0}\end{tikzcd}$$

5.2.2 Ring Merging. The output of the *ring packing* is in the coefficients domain and in the ring $\mathcal{R}^n_{Q_0}$, with a small modulus Q_0 to enable more efficient homomorphic operations. However, in order to support additional homomorphic operations and SIMD encoded arithmetic, we make use of the CKKS bootstrapping, which requires to increase the modulus and ring degree to maintain the same level of security, as explained in Section 4.2. Given a batch of N/n ciphertexts, each encrypting $\sum_{j=0}^{n-1}f(x_{i,j})$ for $0\le i<N/n$, the *ring merging* operation merges them together into a single ciphertext encrypting $\sum_{i=0}^{N/n-1}\left(\sum_{j=0}^{n-1}f(x_{i,j})\cdot X^{N/n}\right)\cdot X^i\in\mathcal{R}^N_{Q_0}$. If a batch contains less than N/n ciphertexts, it is padded with noiseless zero ciphertexts.

$$\begin{tikzcd}\left(\mathsf{RLWE}\left(\sum_{i=0}^{n-1}\mathsf{score}_{P[i]}\cdot X^i\right),\ldots,\mathsf{RLWE}\left(\sum_{i=0}^{n-1}\mathsf{score}_{P[i+n]}\cdot X^i\right)\right)\in(\mathcal{R}^n_{Q_0})^{N/n} \arrow[d, "\text{Merge}"] \\ \mathsf{RLWE}\left(\sum_{i=0}^{N/n-1}\left(\sum_{j=0}^{n-1}\mathsf{score}_{P[j+in]}\cdot X^{N/n}\right)\cdot X^i\right)\in\mathcal{R}^N_{Q_0}\end{tikzcd}$$

5.3 Bootstrapping

Given a ciphertext in $\mathcal{R}^N_{Q_0}$, the primary goal of the CKKS bootstrapping is to produce a new ciphertext in $\mathcal{R}^N_{Q_\ell}$, with $Q_\ell \gg Q_0$ encrypting the same message. Appendix 8.3 provides a full description of the CKKS bootstrapping. We review here the main steps. The CKKS bootstrapping is divided into four steps:

(1) `ModRaise`: raise the modulus from Q_0 to Q_L.
(2) `CoeffsToSlots`: homomorphically encode the underlying message.
(3) `EvalMod`: homomorphically evaluate a modular reduction by Q_0.
(4) `SlotsToCoeffs`: homomorphically decode the underlying message.

The output ciphertext is at level $Q_\ell = Q_{L-k}$, where k is the depth of the bootstrapping circuit. The default bootstrapping aims to keep the message in the same encoding domain, making it independent of the underlying message encoding. However, in addition to raising the modulus from Q_0 to Q_ℓ, one of our goals is to switch the messages from the coefficients domain to the slots domain. We observe that this can be obtained for free by skipping step 4 of the bootstrapping circuit. We denote such a bootstrapping circuit as Half-BTS.

$$\begin{aligned}&\mathsf{RLWE}\left(\sum_{i=0}^{N/n-1}\left(\sum_{j=0}^{n-1}\mathsf{score}_{P[j+in]}\cdot X^{N/n}\right)\cdot X^i\right)\in\mathcal{R}^N_{Q_0}\\ &\qquad\Big\downarrow \mathsf{Half\text{-}BTS}\\ &\mathsf{RLWE}\left(\mathsf{Ecd}\left(\sum_{i=0}^{N/n-1}\left(\sum_{j=0}^{n-1}\mathsf{score}_{P[j+in]}\cdot X^{N/n}\right)\cdot X^i\right)\right)\in\mathcal{R}^N_{Q_\ell}\end{aligned}$$

Note that, although not illustrated here for sake of readability, Half-BTS actually returns two ciphertexts, each encrypting a vector of $N/2$ encoded scores. The reason is that during the `CoeffsToSlots` step, the N scores are encoded as an $N/2$ complex vector, with half of the the scores in the real component and the other half in the imaginary component. And since the step function is defined on the reals, we need to extract this imaginary part into a real vector.

5.4 Core Functions Implementation

5.4.1 Encrypted Scoring Functions. The private scoring function is evaluated for the h attributes of each of the p entries of the matrix P. Each of the attribute scoring function $f_j : \mathbb{R}_{[a,b]} \to \mathbb{Z}_p$ is encoded as a polynomial $\boldsymbol{u}_{f_j} \in \mathcal{R}^n$. We define a discretization factor of $\frac{1}{N}$ and the function f_j is encoded as:

$$\boldsymbol{u}_{f_j} = f(0) - \sum_{i=1}^{N-1} f_j\left(g^{-1}\left(\tfrac{N-i}{N}\right)\right)\cdot X^i,$$

where $g : \mathbb{R}_{[a,b]} \to \mathbb{R}_{[0,1]}$ and $g(x) = \frac{1}{2}\left(\frac{2x-b-a}{b-a}+1\right)$.

Then for $x \in [a, b]$, we have

$$X^{\lfloor Ng(x)\rceil}\cdot \mathsf{RLWE}(\boldsymbol{u}_{f_j}) \approx \mathsf{RLWE}(f_j(x)\cdot X^0 + \star)$$

with an error bounded by $|f_j(x) - f_j(x \pm \frac{b-a}{N})|$. We stress that if $|x_i - x_j| \geq (b-a)/N, \forall x_i, x_j$ then this error does appear since all possible input points can exhaustively be covered. If $|x_i - x_j| \leq (b-a)/N$ (assuming that N cannot be further increased), then the error depends on the smoothness of the function and in such cases it can be mitigated by ensuring that f is an *$\mathcal{L}$-Lipschitz* function over the considered interval[1].

In our use case, the functions are encrypted by the scientist and sent to the database owner for evaluation over the features, each function representing a private selection criterion. The results of each scoring function are aggregated together to form a final score for each of the p entries of the database. This final score for row i is denoted $\mathsf{score}_{P[i]}$ and defined as:

$$\mathsf{RLWE}(\mathsf{score}_{P[i]}) = \sum_{j=0}^{15}\mathsf{RLWE}(\boldsymbol{u}_{f_j})\cdot X^{P[i][j]},$$

5.4.2 Private Threshold Functions. Let the threshold function parameterized by t be:

$$\mathtt{thresh}_t(x) : \begin{cases}1 & \text{if } x > t\\ 0 & \text{otherwise.}\end{cases}$$

Note that $\mathtt{thresh}_t(x)$ is equivalent to $\mathtt{step}(x-t)$, where

$$\mathtt{step}(x) : \begin{cases}1 & \text{if } x > 0\\ 0.5 & \text{if } x = 0\\ 0 & \text{otherwise.}\end{cases}$$

Assuming that $|x_i - x_j| \geq \epsilon$ for some ϵ, and that $|x - t| = k\epsilon$ for some integer k, we can instead evaluate $\mathtt{step}(x - t + \epsilon/2)$, which will prevent the case $x - t = 0$.

For the approach we use, evaluating the threshold functions requires to scale $x - t$ to the interval $[-1, 1]$. For the first local threshold, this is done by normalizing with the factor $\frac{1}{\sum_{j=0}^{15}\max F_j}$ and for the second global threshold by using a normalization factor of $\frac{1}{p}$.

Furthermore, we can make $\mathtt{thresh}_t(x)$ private by providing $\mathsf{RLWE}(t)$ to the evaluator, computing $\mathtt{step}(\mathsf{RLWE}(x) - \mathsf{RLWE}(t))$:

$$\begin{aligned}&\mathsf{RLWE}\left(\mathsf{Ecd}\left(\sum_{i=0}^{N/n-1}\left(\sum_{j=0}^{n-1}\mathsf{score}_{P[j+in]}\cdot X^{N/n}\right)\cdot X^i\right)\right)\in\mathcal{R}^N_{Q_\ell}\\ &\qquad\Big\downarrow \mathtt{thresh}_t\\ &\mathsf{RLWE}\left(\mathsf{Ecd}\left(\sum_{i=0}^{N/n-1}\left(\sum_{j=0}^{n-1}\mathtt{thresh}_t(\mathsf{score}_{P[j+in]})\cdot X^{N/n}\right)\cdot X^i\right)\right)\in\mathcal{R}^N_{Q_\ell}\end{aligned}$$

Evaluating $\mathtt{thresh}_t$ with approximate homomorphic encryption is a challenging task. We chose the approach of [LLNK20], in which they approximate the sign and step functions in the interval $[-1, -2^{-\alpha}] \cup [2^{-\alpha}, 1]$ as a composition of low-degree minimax approximations. Since their approach requires the input to be in

[1] f is an *$\mathcal{L}$-Lipschitz* function if $\forall x_i, y_i$ we have $|f(x_i) - f(x_j)| \leq \mathcal{L}|x_i - x_j|$.

the interval $[-1, -2^{-\alpha}] \cup [2^{-\alpha}, 1]$, the input values need to be normalized to this interval. Because we are evaluating thresholds over discrete values, we can ensure that the threshold always returns the correct value by setting $2^{-\alpha} < 1/p$, where $1/p$ is the normalization factor.

5.5 Putting Altogether

5.5.1 Illustration of the Exploration database procedure. We detail in Figure 1 and Figure 2 the different steps of the homomorphic private function evaluation phase and the homomorphic thresholds evaluation phase processed by the database owner.

5.5.2 Circuit Structure. The full pipeline of the evaluations circuits is summarized in Algorithm 2. Before the protocol begins, the scientist holds the attributes-selection matrix M and the scoring functions and the database owner holds the patients' database.

We review the evaluation process below:

(i) The scientist generates:
- the FHE secret and the necessary evaluation keys, including the keyswitching key and the bootstrapping key;
- the m encrypted scoring functions $\left[\mathsf{RLWE}(\boldsymbol{u}_{f_0}), \ldots, \mathsf{RLWE}(\boldsymbol{u}_{f_{m-1}})\right]$;
- the local encrypted threshold parameter $\mathsf{RLWE}(t_0)$ and normalization factor $\mathsf{RLWE}\left(\frac{1}{\sum_{j=0}^{m-1} \max(f_j)}\right)$;
- the global encrypted threshold parameter $\mathsf{RLWE}(t_1)$ and normalization factor $\mathsf{RLWE}\left(\frac{1}{\sum_{j=0}^{m-1} \max(f_j)}\right)$.

(ii) The scientist sends evaluation keys together with the encrypted values to the database owner;
(iii) the database owner applies the attributes-selection matrix to P to obtain a new patients' database P' and evaluates: $\mathsf{RLWE}(\mathsf{score}_{P'[i]}) = \sum_{j=0}^{m-1} \mathsf{RLWE}(\boldsymbol{u}_{f_j}) \cdot X^{P'[i][j]}$ for $i \in [0, p-1]$. It sets $\mathsf{ct}_i := \mathsf{RLWE}(\mathsf{score}_{P'[i]})$;
(iv) the database owner applies the homomorphic ring packing to the p low degree RLWE ciphertexts;
(v) the database owner applies the homomorphic ring merging operation to switch to a large degree RLWE ciphertexts;
(vi) the database owner applies the scheme switching operation via the partial bootstrapping `Half-BTS` operation to obtain CKKS ciphertexts;
(vii) the database owner evaluates the local threshold homomorphically and aggregates all the values;
(viii) the database owner evaluates the final threshold homomorphically and sends back the result to the scientist.

Algorithm 2: Private Database Exploration

```
1:  Inputs:
    A database $P$ given as a $p \times h$ matrix
    - $M$ an $h \times m$ attributes-selection matrix
    - $[\mathsf{RLWE}(\boldsymbol{u}_{f_0}), \ldots, \mathsf{RLWE}(\boldsymbol{u}_{f_{m-1}})] \in \left((\mathcal{R}^n_{Q_0})^2\right)^m$ a vector of
    encrypted scoring function
    - $\mathsf{RLWE}\left(\frac{1}{\sum \max(f_i)}\right) \in (\mathcal{R}^N_{Q_\ell})^2$
    - $\mathsf{RLWE}(t_0), \mathsf{RLWE}(t_1) \in (\mathcal{R}^N_{Q_\ell})^2$, two private threshold values
    sent encrypted by the scientist to the database owner.

2:  Output: $\mathsf{RLWE}(b)$ where $b = 1$ if at least $t_1$ entries of $P$
    produce individual an score of at least $t_0$, else $b = 0$.

3:  $P' \leftarrow P \times M$ // Attributes selection
4:  $\mathsf{ctsN12} \leftarrow [\emptyset]$
5:  for $i \leftarrow 0$ to $p - 1$ do
6:      $\mathsf{ct}_i \leftarrow \sum_{j=0}^{m-1} \mathsf{RLWE}(\boldsymbol{u}_{f_j}) \cdot X^{P'[i][j]} \in (\mathcal{R}^n_{Q_0})^2$
7:  $\mathsf{ctsN12} \leftarrow \mathtt{Repack}(\mathsf{ct}_0, \cdots, \mathsf{ct}_{p-1})$

8:  $\mathsf{ctsN16} \leftarrow \mathtt{Merge}(\mathsf{ctsN12})$
    // Ring merging & ring switching
        // $\mathcal{R}^n_{Q_0} \rightarrow \mathcal{R}^N_{Q_0}$

9:  $\mathsf{ctScore} \leftarrow \emptyset$
10: for $i \leftarrow 0$ to $\lceil (p(n/N) \rceil$ do
11:     $(\mathsf{ctL}, \mathsf{ctR}) \leftarrow \mathtt{Half\text{-}BTS}(\mathsf{ctsN16}[i])$ // $\mathcal{R}^N_{Q_0} \rightarrow \mathcal{R}^N_{Q_L}$
12:     $\mathsf{ctScore} \leftarrow$
    $\mathsf{ctScore} + \mathtt{step}\left((\mathsf{ctL} - \mathsf{RLWE}(t_0)) \cdot \mathsf{RLWE}\left(\frac{1}{\sum \max(f_i)}\right)\right)$
13:     $\mathsf{ctScore} \leftarrow$
    $\mathsf{ctScore} + \mathtt{step}\left((\mathsf{ctR} - \mathsf{RLWE}(t_0)) \cdot \mathsf{RLWE}\left(\frac{1}{\sum \max(f_i)}\right)\right)$
14:
15: return $\mathtt{step}\left((\mathtt{InnerSum}(\mathsf{ctScore}) - \mathsf{RLWE}(t_1)) \cdot p^{-1}\right)$
```

5.6 Extending Algorithm 2 to Partitioned Databases

In the case of a horizontally or vertically partitioned database, Algorithm 2 either needs trivial or no modification since we are dealing with counts and only one of the database owner is required to download the bootstrapping keys (7.4GB), while all others only need to download the attributes-selection plaintext matrix, the HE-based PFE and the repacking keys (a few MB). In both cases, all database owners can proceed without modification with the PFE evaluation and repacking until line 7 of Algorithm 2 and then send their repacked ciphertexts to the database owner holding the bootstrapping keys. Note that repacked ciphertexts after the PFE evaluation (line 7 of Algorithm 2) have a small expansion ratio and can store up 2^{12} scores each, for a ciphertext size of 65KB, meaning that transmitting the 2^{19} scores at this stage would only require $2^{19}/2^{12} \cdot 65536 \approx 8$MB to transmit p scores under this form.

5.6.1 Horizontally Partitioned Database. Once the database owner holding the bootstrapping keys has received all ciphertexts from the others, it can proceed from line 8 of Algorithm 2 until the end and without additional modification. If the respective value p^{-1} of each database owner cannot be shared, it can be homomorphically computed by multiplying the encryptions of their respective p^{-1}.

5.6.2 Vertically Partitioned Database. Once the database owner holding the bootstrapping keys has received all ciphertexts from other database owners, it must aggregate them before being able to proceed starting from line 8 of Algorithm 2 until the end and without additional modification. In this scenario the value p is the same for all database owners and thus known by all.

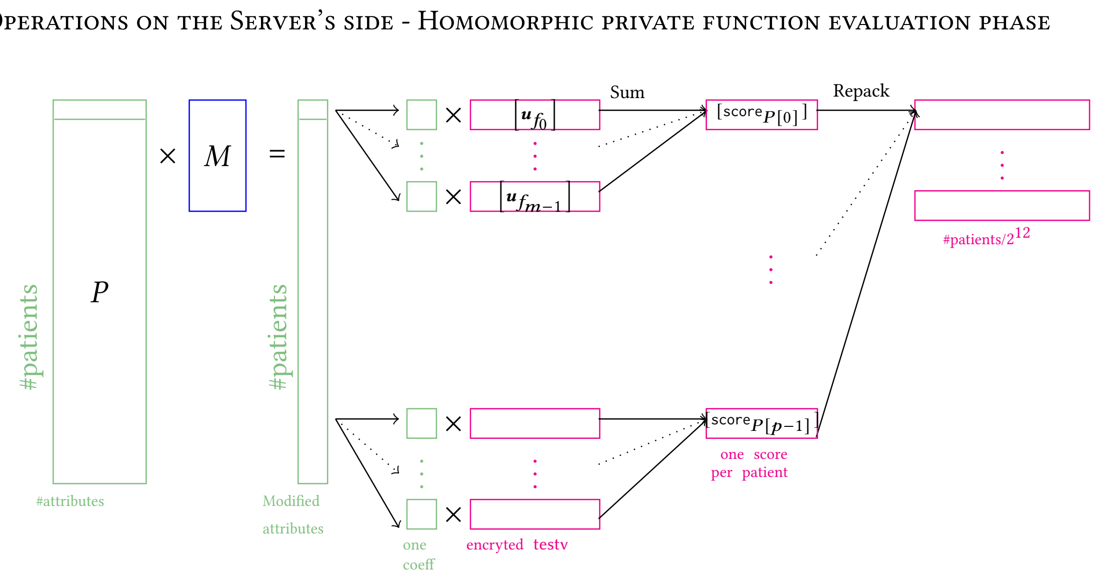

Figure 1: Illustration of the homomorphic private function evaluation (over selected attributes) phase in TETRIS. At the beginning, the scientist holds the attributes-selection matrix M (in blue) and the scoring functions definitions f_j, $j \in [0, m-1]$, and the database owner holds the patients database P (colored in green) represented as a matrix of size $p \times h$. The scientist first sends the attributes-selection matrix M defined in the plaintext domain and the m attribute scoring functions f_j sent encrypted as a polynomial vector $\mathsf{RLWE}(\boldsymbol{u}_{f_j})$ (encrypted testv colored in pink). For the sake of readability, each encrypted polynomial $\mathsf{RLWE}(\boldsymbol{u}_{f_j})$ is denoted $[\boldsymbol{u}_{f_j}]$ in the figure. The results of each scoring function are aggregated together to form an encrypted score for each patient, denoted as $[\mathsf{score}_{P[j]}]$, for $j \in [0, p-1]$ (colored in pink). These intermediate scoring functions are then packed into an encrypted score over a batch of #patients$/2^{12}$ (colored in pink).

6 Implementation & Performance

6.1 Cryptographic Parameters

The private database exploration circuit uses three sets of cryptographic parameters:

- **Set I:** The parameters used for the PFE and for the Ring-Packing and Merging.
- **Set II:** The parameters used for the private thresholds evaluation.
- **Set III:** The parameters used for the bootstrapping circuit, which contain as a subset the parameters of **Set II**.

The details of these parameters can be found in Table 1.

6.2 Other Parameters

Besides the cryptographic parameters, we need to define parameters for the bootstrapping and private threshold functions.

The parametrization of the bootstrapping circuit, uses the default parameters [BTPH22] of the Lattigo library [EL23]: a ring degree of $N = 2^{16}$, a circuit-depth of $k = 15$ (for a total modulus consumption of 821 bits), a ternary secret with Hamming-weight $h = 192$, an ephemeral secret with Hamming-weight $\tilde{h} = 32$, which provide a failure probability of $2^{-138.7}$ and a precision of 27.25 bits for a message $\mathbb{C}^{32768}$ with the real and imaginary part uniformly distributed in $[-1, 1]$. The parametrization of the private threshold functions can be found in Table 2.

6.3 Scientist's Data

The scientist's data sent to the database owner comprises the evaluation keys as well as the encrypted test vectors and encrypted threshold values. The size for each of these objects can be found in Table 3.

6.4 Performance

We implemented Algorithm 2 using the Lattigo library [EL23]. The code can be found in the following repository https://github.com/izama/private-database-exploration. All benchmarks were conducted on the following hardware: Go 1.21, Windows 11, i9-12900K, 32GB DDR4, single threaded. The code is available in the supplementary materials. Table 4 reports the timings for a database of 2^{19} entries with 16 attributes after the evaluation of the attributes-selection matrix. Three out of the 6 values reported are independent of the number of entries in the database: (i) the scientist generation which consist in the evaluation-keys generation, (ii) the database owner generation which consists in the instantiation of the bootstrapping evaluator (generation of the plaintext matrices for the `CoeffsToSlots` and `SlotsToCoeffs` steps) and (iii) the second private threshold which is evaluated on the final count. The three other timings amortize to 1.85ms per entry or 32555 entries per minute, single threaded.

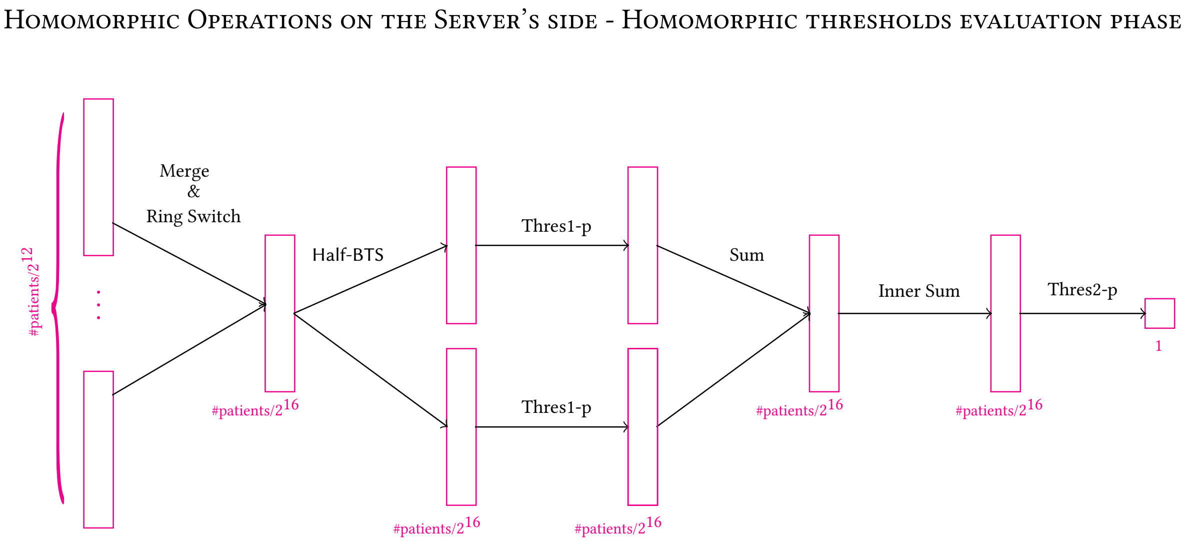

Figure 2: Illustration of the homomorphic thresholds evaluation phase in TETRIS. This phase is processed on the database owner side. At the beginning, he holds a set of #patients/2^{12} ciphertexts which are merged into a ciphertext in a small ring of dimension $N = 2^{12}$ and switched to a larger ring of 2^{16} before entering into the CKKS bootstrapping. The output is two ciphertexts, each encrypting a vector of $N/2 = 2^{15}$ encoded scores for the #patients/2^{16} patients. Then the database owner evaluates a first local threshold, denoted as thres1-p **in the figure and aggregates all the encrypted results via homomorphic summation. The database owner then applies the second private global threshold, denoted as** thres1-p **in the figure and outputs the encrypted result.**

Set	λ	$\log(N)$	$\log(QP)$	$\log(Q)$	$\log(P)$	base2	h
I	128	12	109	55	54	30	2N/3
II	256	16	577	$55 + 8 \cdot 45$	$3 \cdot 54$	-	192
III	128	16	1541	$55 + 8 \cdot 45 + 3 \cdot 39 + 8 \cdot 60 + 4 \cdot 56$	$5 \cdot 61$	-	192

Table 1: Cryptographic Parameters Sets. base2 refers to an additional base 2 decomposition during the key-switching, which is required since P cannot be set to be larger without reducing the security parameters.

	α	β	degrees
Threshold1-p	8	12	[15, 15, 15]
Threshold2-p	16	20	[15, 15, 15, 15, 15]

Table 2: Parameters of the threshold functions: α parametrizes the sensitivity of the threshold, i.e. the interval $[-1, -2^{-\alpha}] \cup [2^{-\alpha}, 1]$ in which the function returns a correct value and $\beta = -\log_2(|b - \lfloor b \rceil|)$, the precision of the output bit b

.

Set	Size [MB]
Ring Packing Keys	3.25
Ring Merging Keys	15.0
Bootstrapping Keys	7395
RLWE(f_i)	1.04
RLWE(t_i)	4.5 + 2.25
Total	≈ 7421

Table 3: Scientist's Data

6.5 Comparison with Sending the Database Encrypted.

Using an HE-based PFE greatly outperforms the alternative, which is to send the database encrypted to the scientist. Although this would remove the need for the scientist to transmit the 7.4GB bootstrapping keys to the server, encrypting a database of p patients and h features has an expansion ratio that is $O(|\mathrm{Dom}_f| \cdot |\log(Q)|)$ (our HE-based PFE requires one-hot encoding of the inputs). In our use case we have $|\mathrm{Dom}_f| = 2^{12}$, $p = 2^{19}$, $h = 16$ and $\log(Q) = 55$ (8 bytes). As such transmitting the same database in an encrypted form to the scientist would require a communication in the order of $8 \cdot (2^{12} \cdot 2^{19} \cdot 16) = 64$GB. Finally, it is noteworthy to point out that this 7.4GB of keys is independent of the circuit and database size and that sending the database in encrypted form to the scientist would leak its size, which remains hidden when using the PFE approach.

Operation	Total	Amortized
Scientist Generation	24.4s	46μs
Database Owner Generation	10.9s	20μs
Private Function & Packing	466.5s	888μs
Half-BTS	73.4s	140μs
Private Threshold 1	384.1s	732μs
Private Threshold 2	58.7s	111μ
Total	1018.0	1.941ms

Table 4: Timings for a database of 2^{19} entries and 16 attributes, single threaded.

6.6 Comparison with PFE

Recalling Section 1, PFE originates from UC-based constructions, first introduced by Valiant [Val76b]. A recent work of Holz et al. [HKRS20] compares the efficiency of different UC-based approaches PFE with Homomorphic-based approaches PFE when using Elliptic curve ElGamal [ElG85], the Brakerski/Fan-Vercauteren (BFV) scheme [FV12] and Damgård-Jurik-Nielsen [DJN10]. Similar to Holz et al., we split our protocol in different phases with a first phase where the party owning the private data can learn either the size of the function or some part of the function such that the global homomorphic evaluation becomes faster. In our framework during the homomorphic private function evaluation phase, the database owner evaluates a private function. In our case, the private function is a scoring function but the method we propose can be used to evaluate an arbitrary function. Making a direct and fair comparison with their work is difficult because of the hardware difference (32 Core @ 2.8Ghz vs. ours which is single thread @ 4.9Ghz) and because they express the complexity of their circuit in term of the number of boolean gates, but we can make a rough estimate. Our HE-based PFE can be expressed as the evaluation of a polynomial of $d = 2^{12}$ coefficients over GF(2^{12}), which requires as many multiplications and additions over GF(2^{12}). One addition over GF(2^{12}) requires 12 XOR gates while a multiplication by X requires 4 XOR gates (modular reduction by a primitive polynomial of Hamming weight 5). As such the multiplication of an unknown field element by a fixed field element a requires $\approx \mathsf{HW}(a) \cdot (12 + 4)$ XOR gates. Therefore, the circuit size would be in the order of 10^5 to 10^6 gates. For 10^6 gates, Holz et al. [HKRS20, Table 1] report a setup of between 300MB with EC ElGamal (smallest) to 2.3GB UC per function (largest), which would amount to 4.8GB for EC ElGamal and 37GB for UC, for 16 different functions. Discarding the setup, since with our approach it is negligible (in the order of a few ms to generate RLWE ciphertexts) and becomes negligible for all approaches when the PFE is evaluated over many points, [HKRS20, Table 2] reports an online time of 1.15 sec per evaluation with BFV (best) to 19 sec with UC (worst) per function evaluation. In our work we only need 0.125MB per function as well as 3.25MB for the repacking keys and have an amortize time of $888/16 = 55\mu$s per function evaluation (which is $\approx 20'000\times$ faster).

7 Conclusion

In this paper, we present TETRIS, a practical solution that allows a scientist to explore a large dataset of patients' records, while maintaining both the privacy of the patients and exploration criteria. Our solution is also communication efficient in the sense that the database does not need to be sent at any point of the computation. We adapt efficient HE-based techniques for arbitrary functions over large domains with the CKKS bootstrapping to support customized and efficient homomorphic circuit compositions. Our experimental results show that a database with 2^{19} entries and 16 features has an amortized total homomorphic processing time of around 1.9ms per entry. Our protocol can be easily extended to support multiple databases, either horizontally or vertically partitioned.

Acknowledgments

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[AGKS20] Masaud Y. Alhassan, Daniel Günther, Ágnes Kiss, and Thomas Schneider. Efficient and scalable universal circuits. *Journal of Cryptology*, 33(3):1216–1271, July 2020.

[BCK+23] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, Jai Hyun Park, and Damien Stehlé. HERMES: Efficient ring packing using MLWE ciphertexts and application to transciphering. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 37–69. Springer, Cham, August 2023.

[BGGJ20] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. CHIMERA: combining ring-lwe-based fully homomorphic encryption schemes. *J. Math. Cryptol.*, 14(1):316–338, 2020.

[BI24] Jean-Philippe Bossuat and Malika Izabachène. Large domain homomorphic evaluation in levelled mode. https://fhe.org/conferences/conference-2024/resources, 2024.

[BTPH22] Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security*, pages 521–541, Cham, 2022. Springer International Publishing.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 34–54. Springer, 2019.

[CDKS21] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 21International Conference on Applied Cryptography and Network Security, Part I*, volume 12726 of *LNCS*, pages 460–479. Springer, Cham, June 2021.

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

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

[CHH18] Jung Hee Cheon, Kyoohyung Han, and Minki Hhan. Faster homomorphic discrete fourier transforms and improved FHE bootstrapping. *IACR Cryptol. ePrint Arch.*, 2018:1073, 2018.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 360–384. Springer, Cham, April / May 2018.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[CKP22] Jung Hee Cheon, Wootae Kim, and Jai Hyun Park. Efficient homomorphic evaluation on large intervals. *IEEE Trans. Inf. Forensics Secur.*, 17:2553–2568, 2022.

[DJN10] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.

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

[EL23] Tune Insight SA EPFL-LDS. Lattigo v5. Online: https://github.com/tuneinsight/lattigo, November 2023.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[GINX16] Nicolas Gama, Malika Izabachène, Phong Q. Nguyen, and Xiang Xie. Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 528–558. Springer, Berlin, Heidelberg, May 2016.

[GMS$^+$23] Till Gehlhar, Felix Marx, Thomas Schneider, Ajith Suresh, Tobias Wehrle, and Hossein Yalame. Safefl: Mpc-friendly framework for private and robust federated learning. In *2023 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, May 25, 2023*, pages 69–76. IEEE, 2023.

[HKRS20] Marco Holz, Ágnes Kiss, Deevashwer Rathee, and Thomas Schneider. Linear-complexity private function evaluation is practical. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 401–420. Springer, Cham, September 2020.

[HS15] Shai Halevi and Victor Shoup. Bootstrapping for HElib. In *Annual International conference on the theory and applications of cryptographic techniques*, pages 641–670. Springer, 2015.

[IIMP22] Ilia Iliashenko, Malika Izabachène, Axel Mertens, and Hilder V. L. Pereira. Homomorphically counting elements with the same property. *PoPETs*, 2022(4):670–683, October 2022.

[jLHH$^+$21] Wen jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy*, pages 1057–1073. IEEE Computer Society Press, May 2021.

[KDE$^+$21] Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for RLWE-based homomorphic encryption. Cryptology ePrint Archive, Report 2021/691, 2021.

[KM11] Jonathan Katz and Lior Malka. Constant-round private function evaluation with linear complexity. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 556–571. Springer, Berlin, Heidelberg, December 2011.

[KS08] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In Gene Tsudik, editor, *FC 2008*, volume 5143 of *LNCS*, pages 83–97. Springer, Berlin, Heidelberg, January 2008.

[KS16] Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 699–728. Springer, Berlin, Heidelberg, May 2016.

[LLK$^+$22] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 551–580. Springer, Cham, May / June 2022.

[LLNK20] Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. Minimax approximation of sign function by composite polynomial for homomorphic comparison. Cryptology ePrint Archive, Paper 2020/834, 2020. https://eprint.iacr.org/2020/834.

[LXY24] Fengrun Liu, Xiang Xie, and Yu Yu. Scalable multi-party computation protocols for machine learning in the honest-majority setting. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

[LYZ$^+$21] Hanlin Liu, Yu Yu, Shuoyao Zhao, Jiang Zhang, Wenling Liu, and Zhenkai Hu. Pushing the limits of valiant's universal circuits: Simpler, tighter and more compact. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 365–394, Virtual Event, August 2021. Springer, Cham.

[MS13] Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 557–574. Springer, Berlin, Heidelberg, May 2013.

[MWA$^+$23] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *2023 IEEE Symposium on Security and Privacy*, pages 477–496. IEEE Computer Society Press, May 2023.

[QZZ$^+$23] Jiaming Qian, Ping Zhang, Haoyong Zhu, Muhua Liu, Jiechang Wang, and Xuerui Ma. Lhdnn: Maintaining high precision and low latency inference of deep neural networks on encrypted data. *Applied Sciences*, 13(8), 2023.

[SKS$^+$22] Sebastian Stammler, Tobias Kussel, Phillipp Schoppmann, Florian Stampe, Galina Tremper, Stefan Katzenbeisser, Kay Hamacher, and Martin Lablans. Mainzelliste secureepilinker (mainsel): privacy-preserving record linkage using secure multi-party computation. *Bioinformatics*, 38(6):1657–1668, 2022.

[SPT$^+$21] Sinem Sav, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. POSEIDON: Privacy-preserving federated neural network learning. In *NDSS 2021*. The Internet Society, February 2021.

[Val76a] Leslie G. Valiant. Universal circuits (preliminary report). In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*, pages 196–203, https://doi.org/10.1145/800113.803649, 1976. ACM.

[Val76b] Leslie G. Valiant. Universal circuits (preliminary report). In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, page 196–203, New York, NY, USA, 1976. Association for Computing Machinery.

Appendix

8 Additional preliminaries

8.1 Plaintext Domains

Throughout this work we will distinguish between two plaintext domains: in the ring (a.k.a in the *coefficients*) and in the canonical embedding (a.k.a in the *slots*).

8.1.1 In the ring. Messages are said to be in the *coefficients* if their encoding provides an arithmetic that follows the regular arithmetic of $\mathcal{R}^N$ (see Section 2.2). Encoding a message $m \in \mathbb{R}^n$ for $n \leq N$ on $\mathcal{R}^N$ (and its inverse operation) is done with the following steps:

$$\mathbb{R}^n \underset{(1)}{\longleftrightarrow} \mathbb{R}[Y] \overset{\lfloor \Delta \cdot \rceil}{\underset{1/\Delta}{\longleftrightarrow}} \mathbb{Z}[Y] \underset{(2)}{\longleftrightarrow} \mathcal{R}^N[X],$$

for $Y = X^{N/n}$.

- (1): the mapping $\mathbb{R}^n \rightarrow \mathbb{R}[Y]$ is defined as interpreting a vector in $\mathbb{R}^n$ as a polynomial in Y of n real coefficients.
- (2): the mapping $\mathbb{Z}[Y] \rightarrow \mathcal{R}^N[X]$ is defined as taking an integer polynomial in Y and embedding it in $\mathcal{R}^N[X]$ by applying the change of variable $Y \rightarrow X^{N/n}$.

The scaling factor Δ is used to map $\mathbb{R}$ to $\mathbb{Z}$ via fixed-point arithmetic. Naturally, it will grow from Δ to Δ^2 after a multiplication and its magnitude is managed by applying $\lfloor 1/\Delta \rceil$ on the coefficients and the modulus of $\mathcal{R}^N[X]$. In the following, we will denote $\lll_k$ for the cyclic rotation by k positions to the left of a vector or polynomial. As $\mathbb{Z}^*_{2N}$ is generated by -1 and 5 (see Section 2.2), we will denote $\phi_{(k_0,k_1)}$ the automorphism which maps X^i to $X^{i\cdot(-1)^{k_0}(5)^{k_1}}$.

8.1.2 In the Canonical Embedding. Messages are said to be in the *slots* if their plaintext arithmetic, when doing operation in $\mathcal{R}$, behaves as single-instruction-multiple-data (SIMD) over $\mathbb{C}^n$.

Let n be a power-of-two integer such that $1 \leq n < N$ and $\psi = e^{i\pi/n}$ be a $2n$-th primitive root of unity. Since $(-1)^{k_0}5^{k_1}$ for $0 \leq k_0 < 2$ and $0 \leq k_1 < n/2$ spans $\mathbb{Z}^*_{2n}$, we have $\{\psi^{5^k}, \overline{\psi^{5^k}}, 0 \leq k < n/2\}$ the set of all $2n$-th primitive roots of unity, which form an orthogonal basis over $\mathbb{C}^n$.

The decoding complex discrete Fourier transform of dimension n (DFT_n) is characterized by the $n \times n$ special Fourier matrix $\mathsf{SF}_n[j][k] = \psi^{j5^k}$ (up to a bit-reversal permutation) for $0 \leq j, k < n$, with its inverse (DFT_n^{-1}), the encoding matrix, being $\mathrm{SF}_n^{-1} = \frac{1}{n}\overline{\mathrm{SF}}_n^T$ due to the orthogonality of the basis. For $Y = X^{N/n}$, encoding a vector $\boldsymbol{m} \in \mathbb{C}^n$ on $\mathcal{R}^N[X]$ (and its inverse operation) is similar to encoding in the *coefficients* but with a pre-processing step involving the evaluation of DFT_n^{-1} (or DFT_n for the decoding) on $\boldsymbol{m}$:

$$\mathbb{C}^n \xleftrightarrow[\mathsf{DFT}_n]{\mathsf{DFT}_n^{-1}} \mathbb{C}^n \xleftrightarrow[(1)]{} \mathbb{R}^{2n} \xleftrightarrow[(2)]{} \mathbb{R}[Y] \xleftrightarrow[1/\Delta]{\lfloor\Delta\cdot\rceil} \mathbb{Z}[Y] \xleftrightarrow[(3)]{} \mathcal{R}^N[X].$$

- (1): the mapping $\mathbb{C}^n \to \mathbb{R}^{2n}$ maps a complex vector of size n to a real vector of size $2n$ and is defined as $\mathsf{Re}(\boldsymbol{m})||\mathsf{Im}(\boldsymbol{m})$ (its inverse being defined as $\boldsymbol{m}[:n] + i \cdot \boldsymbol{m}[n:]$).
- (2): the mapping $\mathbb{R}^{2n} \to \mathbb{R}[Y]$ is defined as interpreting the vector $\boldsymbol{m}$ as a polynomial in Y of $2n$ real coefficients.
- (3): the mapping $\mathbb{R}[Y] \to \mathcal{R}^N[X]$ is defined as taking a polynomial in Y and embedding it in $\mathcal{R}^N[X]$ by applying the change of variable $Y \to X^{N/n}$.

We will identify messages in the *slots* with the brackets $\langle\cdot\rangle$ (else they are assumed to be in the *coefficients*) and for any two vectors $\boldsymbol{m}_0, \boldsymbol{m}_1 \in \mathbb{C}^n$, the following holds due to the convolution property of the complex DFT:

- Addition: $\langle \boldsymbol{m}_0\rangle + \langle \boldsymbol{m}_1\rangle = \langle \boldsymbol{m}_0 + \boldsymbol{m}_1\rangle$
- Multiplication: $\langle \boldsymbol{m}_0\rangle * \langle \boldsymbol{m}_1\rangle = \langle \boldsymbol{m}_0 \cdot \boldsymbol{m}_1\rangle$

Additionally, automorphisms on messages in the *slots* act as vector operations in the following way:

- Rotation: $\phi_{(0,k)}(\langle \boldsymbol{m}\rangle) = \langle \boldsymbol{m} \lll_k\rangle$
- Conjugation: $\phi_{(1,0)}(\langle \boldsymbol{m}\rangle) = \langle \overline{\boldsymbol{m}}\rangle$

The same scaling factor management as the *coefficient* encoding is required after multiplication. For the rest of this work, we will denote $\lfloor \Delta\tau_n(m)\rceil$ the process of encoding $\boldsymbol{m} \in \mathbb{C}^n$ on $\mathcal{R}^N[X]$ and $\tau^{-1}(\Delta^{-1}m)$ its inverse.

8.2 Advanced Homomorphic Operations

We detailed below how homomorphic operations are performed using key-swicthing:

- Relinearization: the multiplication between two RLWE ciphertexts $\mathcal{P}(s)$ returns a new RLWE ciphertext $\mathcal{P}^2(s)$, and the degree will further increase if we perform additional multiplication with non-plaintext operands. To ensure the compactness of the multiplication, the key-switching operation is used to homomorphically re-enrypt $\mathcal{P}^2(s)$ to $\mathcal{P}^1(s)$ by evaluating $\mathsf{SwitchKey}(\mathcal{P}^{[2]}(s^2), \mathsf{swk}_{s^2\to s}) + \mathcal{P}^{[0:1]}(s)$;

- Automorphism: an automorphism ϕ on an RLWE ciphertext $\mathcal{P}(s)$ maps it to a new RLWE ciphertext $\mathcal{P}(\phi(s))$. The key-switching operation enables to re-encrypt $\mathcal{P}(\phi(s))$ to $\mathcal{P}(s)$ by evaluating $\mathsf{SwitchKey}(\mathcal{P}^{[1]}(\phi(s)), \mathsf{swk}_{\phi(s)\to s}) + \mathcal{P}^{[0]}(\phi(s))$;

- Ring Splitting: it is possible to split an RLWE ciphertext $\mathcal{P}(s) + m$ with $m, s \in \mathcal{R}^N$ into two ciphertexts of half the dimension $(\mathcal{P}(s') + m_0, \mathcal{P}(s') + m_1)$ with $m_0, m_1, s' \in \mathcal{R}^{N/2}$ such that $m(X) = m_0(Y) + X * m_1(Y)$ for $Y = X^2$, by evaluating the following: $d = \mathcal{P}(s'(Y)) + m = \mathsf{SwitchKey}\left(\mathcal{P}^{[1]}(s), \mathsf{swk}_{s\to s'(Y)}\right) + \mathcal{P}^{[0]}(s) + m$ and returning $(d(Y \to X), (X * d)(Y \to X))$, where $Y \to X$ denotes the change of variable from Y to X. This can be generalized to $Y = X^{2^i}$;

- Ring Merging: it is possible to merge two RLWE ciphertexts $d_0 = \mathcal{P}(s') + m_0$ and $d_1 = \mathcal{P}(s') + m_1$ with $m_0, m_1, s' \in \mathcal{R}^{N/2}$ into an RLWE ciphertext of twice the dimension $\mathcal{P}(s) + m$ with $m, s \in \mathcal{R}^N$ such that $m(X) = m_0(Y) + X * m_1(Y)$ for $Y = X^2$ by evaluating the following: $\mathcal{P}(s'(Y)) + m = d_0(Y) + X * d_1(Y)$ and returning $\mathcal{P}(s) + m = \mathsf{SwitchKey}\left(\mathcal{P}^{[1]}(s'(Y)), \mathsf{swk}_{s'(Y)\to s}\right) + \mathcal{P}^{[0]}(s'(Y)) + m$. This can be generalized to $Y = X^{2^i}$.

8.3 Bootstrapping for Approximate Homomorphic Encryption

In this section, we recall the CKKS bootstrapping fours steps: ModRaise: the ciphertext $\mathcal{P}_q(s) + m(Y)$ is expressed as modulo Q. This returns a new ciphertext $\mathcal{P}_Q(s) + m'(X)$ where $m'(X) = q \cdot I(X) + m(Y)$. The unwanted integer polynomial $q \cdot I(X)$ comes from the absence of a reduction modulo q during the decryption process with $||I(X)||$ being $O(\sqrt{h})$. If $2n \neq N$, then $Y \neq X$ and $I(X)$ is not a polynomial in Y. In other words, we have multiples of q in the coefficients X that are not multiples of $N/2n$. In this case, we can map $q \cdot I(X) + m(Y)$ to $q \cdot I(Y) + m(Y)$ by evaluating a trace-like map [CHK+18] that maps $X \to Y$ making coefficients in X whose degree is not a multiple of $N/2n$ vanish. This map can be efficiently evaluated with $\log(N/2n)$ automorphisms [CHK+18] and doing so reduces the complexity of the next step. The remaining steps of the bootstrapping are aimed at removing $q \cdot I(Y)$ by homomorphically evaluating a coefficient-wise modular reduction by q on m'.

CoeffsToSlots: $m'(Y)$ can be seen as a fresh message in the *coefficients* and SIMD evaluation, it needs to be encoded in the *slots*. To do so, this step homomorphically evaluates $\tau_n(m'(Y))$ returning $\langle m'(Y)\rangle$;

EvalMod: a polynomial approximation of the modular reduction by q is homomorphically evaluated on $\langle m'(Y)\rangle$, making the polynomial $I(Y)$ vanish;

SlotsToCoeffs: the inverse of the canonical embedding, τ_n^{-1}, is evaluated on $\langle m'(Y)\rangle$ and a close approximation of original message $m(Y)$ minus the unwanted polynomial is retrieved. After this last step the ciphertext has modulus $Q' > \Delta^2 > q$ and we can evaluate further operations, until the ciphertext becomes *exhausted* again and a new bootstrapping is needed.

Note that a ciphertext at modulus $Q' < Q$ is returned because the *bootstrapping* circuit consumes part of the original modulus Q as it requires to call rescaling operations during the CoeffsToSlots, EvalMod and SlotsToCoeffs steps.

8.4 CoeffsToSlots and SlotsToCoeffs

The goal of the CoeffsToSlots and SlotsToCoeffs steps is to homomorphically evaluate τ_n on a message m. Since the encoding and decoding discrete Fourier transforms are fully defined by the $n \times n$

matrices SF_n and SF_n^{-1} respectively (see Section 8.1 which details the plaintext domain spaces), its homomorphic evaluation can be expressed in terms of plaintext matrix-vector products:

(1) $\mathsf{CoeffsToSlots}(m_{rev}) \rightarrow \langle m \rangle : m_{rev} \times \mathrm{SF}_n^{-1}$;
(2) $\mathsf{SlotsToCoeffs}(\langle m \rangle) \rightarrow m_{rev} : \langle m \rangle \times \mathrm{SF}_n$,

where the suffix $_{rev}$ denotes the bit-reversal permutation.

In their initial bootstrapping proposal, Cheon et al. [CHK+18] homomorphically compute the DFT as a single matrix-vector product in $O(\sqrt{n})$ rotations and depth 1, by using the baby-step giant-step (BSGS) approach of Halevi and Shoup [HS15]. To further reduce the complexity, two recent works from Cheon et al. [CHH18] and Chen et al. [CCS19] exploit the structure of the equivalent FFT algorithm by recursively merging its iterations, reducing the complexity to $O(\sqrt{r}\log_r(n))$ rotations at the cost of increasing the depth to $O(\log_r(n))$, for r a power-of-two radix between 2 and n.

Note that this FFT factorization does not extend to the bit-reversal permutation as it cannot be factorized, thus it is omitted in practice, and we might require to adapt the following homomorphic operations to bit-reversal inputs.