Faster homomorphic comparison operations for BGV and BFV

Ilia Iliashenko, Vincent Zucca

Faster homomorphic comparison operations for BGV and BFV

Ilia Iliashenko[1] and Vincent Zucca[2,3]

[1] imec-COSIC, Dept. Electrical Engineering, KU Leuven, Belgium
ilia@esat.kuleuven.be
[2] DALI, Universit de Perpignan Via Domitia, France,
vincent.zucca@univ-perp.fr
[3] LIRMM, Univ Montpellier, Montpellier, France,
vincent.zucca@lirmm.fr

Abstract. Fully homomorphic encryption (FHE) allows to compute any function on encrypted values. However, in practice, there is no universal FHE scheme that is efficient in all possible use cases. In this work, we show that FHE schemes suitable for arithmetic circuits (e.g. BGV or BFV) have a similar performance as FHE schemes for non-arithmetic circuits (TFHE) in basic comparison tasks such as less-than, maximum and minimum operations. Our implementation of the less-than function in the HElib library is up to 3 times faster than the prior work based on BGV/BFV. It allows to compare a pair of 64-bit integers in 11 milliseconds, sort 64 32-bit integers in 19 seconds and find the minimum of 64 32-bit integers in 9.5 seconds on an average laptop without multi-threading.

1 Introduction

Fully Homomorphic Encryption (FHE) can perform any kind of computations directly on encrypted data. It is therefore a natural candidate for privacy-preserving outsourced storage and computation techniques. Since Gentry's breakthrough in 2009 [25], FHE has received a worldwide attention which has resulted in numerous improvements. As a result, FHE can now be used in practice in many practical scenarios, e.g. genome analysis [30], energy forecasting [7], image recognition [9] and secure messaging [4]. In addition, FHE is currently going through a standardization process [1].

In practice, homomorphic encryption (HE) schemes can be classified into three main categories:

– The schemes encrypting their input bit-wise meaning that each bit of the input is encrypted into a different ciphertext. From there, the operations are carried over each bit separately. Examples of such schemes include FHEW [22] and TFHE [18]. These schemes are believed to be the most efficient in practice with relation to the *total running time*.

– The second category corresponds to word-wise encryption schemes that allow to pack multiple data values into one ciphertext and perform computations on these values in a Single Instruction Multiple Data (SIMD) fashion [36]. In particular, encrypted values are packed in different slots such that the operations carried over a single ciphertext are automatically carried over each slot independently. Schemes with these features include BGV [11] and BFV [10,24]. Although homomorphic operations in these schemes are less efficient than for bit-wise encryption schemes, their running time per SIMD slot can be better than of the binary-friendly schemes above. We refer to this performance metric as the *amortized running time*.
– The CKKS scheme [14], which allows to perform computations over approximated numbers, forms the third category. It is similar to the second category in the sense that one can pack several numbers and compute on them in a SIMD manner. The CKKS scheme does not have the algebraic constraints that lower the packing capacity of BGV and BFV. Hence, it is usually possible to pack more elements in CKKS ciphertext, thus resulting with the best amortized cost. Unlike previous schemes, CKKS encodes complex, and thus real, numbers natively. However, homomorphic computations are not exact, which means that decrypted results are only valid up to a certain precision.

Each category of schemes is more efficient for a certain application. Thus, when comparing the efficiency of different homomorphic schemes, one must take into account the given use case.

It is commonly admitted that schemes of the first category are the most efficient ones for generic applications. Since they operate at the bit level, they can compute every logical gate very efficiently. The total running time being in this case the sum of the times needed to evaluate each gate of the circuit. As a result, to optimize the computations for a given application, the only possibility is to reduce the length of the critical computational path and parallelize the related circuit as much as possible. However, as this becomes more and more difficult as the size of the circuit grows, it is possible to optimize only some parts of the circuit by identifying some patterns [5]. Another advantage of these schemes is that they have very fast so-called 'bootstrapping' algorithms that 'refresh' ciphertexts for further computation. This is very convenient in practice as one can set a standard set of encryption parameters without knowing what function should be computed.

Schemes of the second category operate naturally on p-ary arithmetic circuits, i.e. they are very efficient to evaluate polynomial functions over $\mathbb{F}_p$, for a prime p. However, these schemes become much less efficient when considering other kinds of computations, e.g. comparison operations, step functions. To alleviate this problem, one can use tools from number theory to evaluate specific functions with relatively efficient p-ary circuits. Nonetheless, in general this techniques are too weak to outperform schemes of the first category. Moreover, bootstrapping algorithms of these schemes are quite heavy and usually avoided in practice.

CKKS, similarly to second category schemes, is very efficient when operating on arithmetic circuits. However, unlike other schemes which perform modular

arithmetic, it allows to perform computations on complex (and thus real) numbers. Although this is an important advantage for many use cases, CKKS lacks simplification tools for evaluation of certain functions due to number-theoretic phenomena as for the second category. However, since CKKS usually supports huge packing capacity, it usually presents the best amortized cost. The bootstrapping algorithm of CKKS is fundamentally different from the above schemes as it refreshes ciphertexts only partially and introduces additional loss of output precision. Therefore, the CKKS bootstrapping is usually avoided in practice.

Although FHE now offers a relatively efficient alternative for secure computation, some functions remain difficult to evaluate efficiently regardless the considered scheme. Step functions, which are required in many practical applications, form a good example of such functions because of their discontinuous nature. The difficulty to evaluate discontinuous functions comes from the hardness to evaluate a quite basic and relatively simple function: the comparison function. Although comparison is an elementary operation required in many applications including the famous *Millionaires problem* of Yao [39] or advance machine learning tasks of the iDASH competition[†], it remains difficult to evaluate homomorphically.

By now, schemes of the first category look much more suitable for such non-arithmetic tasks, but they are hopelessly inefficient for evaluating arithmetic functions. Hence, one should resort to heavy conversion algorithms [8] to leverage the properties of different schemes.

1.1 Contributions

In this work, we describe the structure of the circuits corresponding to comparison functions for the BGV and BFV schemes. For theses schemes, there exists two approaches: either compare two numbers x and y directly by evaluating a bivariate polynomial in x and y, or study the sign of the difference $z = x - y$ by evaluating a univariate polynomial in z.

By exploiting the structure of these two polynomials, we show that it is possible to evaluate them more efficiently than what was proposed in the state of the art. In particular, we prove that these polynomials have multiple zero coefficients that can be ignored during polynomial evaluation.

The benefit of our approach results in significant performance enhancement for both methods. On the one hand, our bivariate circuit can compare two 64-bit integers with an amortized cost of 21ms, which is a gain of 40% with relation to the best previously reported results of Tan et al. [37] (See Table 1). On the other hand, our univariate circuit shows even better results with an amortized cost of 11ms for 64-bit numbers – which is, to the best of our knowledge, more than 3 times faster than previously reported results for this kind of scheme [37]. Note that we can compare two 20-bit numbers with an amortized cost of 3ms, which is better by a factor 1.9 than what can be achieved with CKKS-based algorithms and is comparable to TFHE-based implementations (see Table 5).

[†] http://www.humangenomeprivacy.org/2020/index.html

We also apply our comparison methods to speed up popular computational tasks such as sorting and computing minimum/maximum of an array with N elements. For example, for $N = 64$, we obtain an amortized cost of 6.5 seconds to sort 8-bit integers and 19.2 seconds for 32-bit integers, which is faster than the prior work by a factor 9 and 2.5 respectively (see Table 3). For $N = 64$, we can find the minimum of 8-bit integers with an amortized running time of 404 ms and of 32-bit integers with an amortized time of 9.57 seconds (see Table 4).

1.2 Related Art

Comparison is a common function required in many applications; as a consequence, its homomorphic evaluation has been the object of several works. Since inputs are encrypted, a comparison algorithm cannot terminate whenever it finds the first difference between most significant bits. As a result, homomorphic comparison has a complexity corresponding to the worst-case complexity in the plain domain. The practical efficiency of homomorphic comparison depends on the type of HE schemes considered.

For bit-wise HE schemes (FHEW, TFHE), Chillotti et al. [19,20] showed that one could compare two n-bit integers by evaluating a deterministic weighted automata made of $5n$ `CMux` gates. Using the TFHE scheme, evaluating a `CMux` gate takes around 34 microseconds on a commodity laptop, meaning that one can homomorphically compare two n-bit numbers in around $170n$ microseconds. Note that these estimations correspond to the fastest (leveled) version of TFHE which avoids bootstrapping. If one wants to use the bootstrapped version then the best method requires to evaluate $7n$ `Mux` gates, where each gate takes around 26 millisecond to be evaluated, which makes a total of $182n$ millisecond.

Schemes from the second category (BGV and BFV) can use SIMD techniques to batch several plaintexts into a single ciphertext [36]. Therefore, a natural idea would be to pack the input bits into a single ciphertext. Cheon et al. [17,13] studied comparison functions in this context using the bivariate polynomial interpolation. Some of the algorithmic tools they have used – e.g. computation of running sums and products – are optimal in the homomorphic setting, i.e. regarding the multiplicative depth and the number of multiplications, and have laid the ground for future works in this direction.

Some works have tried to exploit other features of these schemes by encoding integers modulo an odd prime p instead of bits. In [33], Narumanchi et al. compare integer-wise comparison algorithms based on the univariate interpolation with bit-wise counterparts. The SIMD packing was ignored in this study. They concluded that bit-wise methods are more efficient because they have a smaller multiplicative depth. In particular, n-bit numbers can be compared with a circuit of depth $\mathcal{O}(\log n)$ instead of $\mathcal{O}(n)$ in the case of integer-wise algorithms. This comes from the fact that integer-wise comparison circuits require to evaluate a Lagrange interpolation polynomial of degree $p - 1 \geq 2^n$.

In [29], Kim et al. noticed that SIMD packing techniques reduce the multiplicative complexity of homomorphic comparison circuits. In addition, they took advantage of the nature of the finite field $\mathbb{F}_{p^d}$, which corresponds to the plaintext

space of a SIMD slot. Namely, any power x^{p^i} can be evaluated with the homomorphic Frobenius automorphism $x \mapsto x^p$, which does not consume any homomorphic multiplicative depth level. This allowed to reduce the depth of the equality circuit $\texttt{EQ}(x, y) = 1 - (x - y)^{p^d - 1}$ from $\lceil d \log_2(p) \rceil$ to $\lceil \log_2(d) \rceil + \lceil \log_2(p - 1) \rceil$.

Tan et al. [37] proposed a method to perform digit-wise comparison using SIMD and the bivariate polynomial interpolation. Their idea consists in decomposing input integers into digits of size p^r encoded into a subfield of $\mathbb{F}_{p^d}$, with $r|d$, in order to reduce the degree of the Lagrange interpolation polynomial of a comparison function. To compare input integers, one should extract digits, compare them and combine the results of digit comparison using the lexicographical order. Note that their evaluation of the lexicographical order intensively uses the efficient equality circuit of Kim et al. [29]. Overall, they have used their method to compare integers up to 64-bit while reporting, to the best of our knowledge, the current best timings for performing homomorphic comparison with BGV scheme.

Finally, in [35], Shaul et al. used the univariate approach to evaluate comparison functions in the context of top-k selection with integer-wise circuits. However, they did not use the decomposition method of Tan et al. [37], thus obtaining relatively poor performance of comparison.

Note that all these works did not exploit the structure of comparison interpolation polynomials neither in the bivariate nor in the univariate case. Kaji et al. [28] described basic properties of the polynomial expressions of max, argmax and other non-arithmetic functions over non-binary prime fields $\mathbb{F}_p$. However, their results do not allow to evaluate these functions very efficiently, as an example their homomorphic circuit to evaluate max has a quadratic complexity in p.

The situation for the CKKS scheme is quite different since its plaintext space natively supports complex/real numbers. Therefore, circuit optimizations related to data encoded into finite fields are not applicable for CKKS. Nonetheless, the approximated nature of computations in CKKS makes it suitable to use iterative methods from real analysis to compute close approximations of non-arithmetic functions. Bajard et al. [6] used Newton iteration to evaluate the sign function, while independently Cheon et al. [16,15] generalized this approach and studied its efficiency in more details. Using the methods of [15], one can compare 20-bit numbers with an amortized cost comparable, although slower, to TFHE. However, to obtain these timings one has to use quite large encryption parameters (ring dimension 2^{17} and ciphertext modulus up to 2200 bits). Since the running time of homomorphic operations increases quasi-linearly with the ring dimension, while the number of slots only increases linearly, increasing the dimension would affect significantly the timings. Therefore, it would be interesting to know whether these methods can be used in practice to compare larger inputs – e.g. 64-bit integers – without degrading the performance.

2 Background

2.1 Notations

Vectors will be written in column form and denoted by boldface lower-case letters. The set of integers $\{\ell, \ldots, k\}$ is denoted by $[\ell, k]$. For a non-negative integer a, let $\mathtt{wt}(a)$ be the Hamming weight of its binary expansion. We denote the set of residue classes modulo p by $\mathbb{Z}_p$ and the class representatives of $\mathbb{Z}_p$ are taken from the half-open interval $[-p/2, p/2)$.

2.2 Comparison of integers with finite fields operations

Let $\mathcal{S}$ be a totally ordered set with a binary relation $<$. For any $x, y \in \mathcal{S}$, we can define the less-than and the equality functions as follows.

$$\mathtt{LT}_{\mathcal{S}}(x, y) = \begin{cases} 1, & \text{if } x < y; \\ 0, & \text{if } x \geq y, \end{cases}$$

$$\mathtt{EQ}_{\mathcal{S}}(x, y) = \begin{cases} 1, & \text{if } x = y; \\ 0, & \text{if } x \neq y. \end{cases}$$

Functions over finite fields The map defined by $\chi : x \mapsto x^{p^d-1}$ from $\mathbb{F}_{p^d}$ to the binary set $\{0, 1\}$ is called the *principal character*. According to Euler's theorem, it returns 1 if x is non-zero and 0 otherwise. Using the principal character, every function from $\mathbb{F}_{p^d}^l$ to $\mathbb{F}_{p^d}$ can be interpolated by a unique polynomial according to the following well-know lemma.

Lemma 1. *Every function $f : \mathbb{F}_{p^d}^l \to \mathbb{F}_{p^d}$ is a polynomial function represented by a unique polynomial $P_f(X_1, \ldots, X_l)$ of degree at most $p^d - 1$ in each variable. In particular,*

$$P_f(X_1, \ldots, X_l) = \sum_{\mathbf{a} \in \mathbb{F}_{p^d}^l} f(\mathbf{a}) \prod_{i=1}^{l} \left(1 - \chi(X_i - a_i)\right) .$$

where a_i is the ith coordinate of vector $\mathbf{a}$.

Comparison of integers Let $\mathbb{F}_{p^d} = \mathbb{F}_p[X] / \langle f(X) \rangle$ for some irreducible monic polynomial $f(X)$ of degree d. Let $d' \leq d$ and $\mathcal{S} \subseteq [0, p^{d'} - 1]$, we can map $\mathcal{S}$ into $\mathbb{F}_{p^d}$ by using the decomposition of integers in base p:

$$\begin{array}{rccc} \iota_p : & \mathcal{S} & \to & \mathbb{F}_{p^d} , \\ & \sum_{i=0}^{d'-1} a_i p^i & \mapsto & \sum_{i=0}^{d'-1} a_i X^{i-1} . \end{array}$$

Note that the map ι_p is injective and gives a one-to-one correspondence between $\mathcal{S}$ and $\mathbb{F}_{p^d}$ when $d' = d$ and $\mathcal{S} = [0, p^d - 1]$. Therefore, we identify integers belonging to $\mathcal{S}$ with their image by ι_p and thus omit ι_p when the situation is clear from the context.

Let $a, b \in \mathcal{S}$ be two integers to be compared and $\sum_{i=0}^{d'-1} a_i X^i$ and $\sum_{i=0}^{d'-1} b_i X^i$ with $a_i, b_i \in \mathbb{F}_p$ are their respective encodings into $\mathbb{F}_{p^d}$. The order of the set $\mathcal{S}$ induces a polynomial function in $\mathbb{F}_{p^d}$, which can be interpolated by Lemma 1. However, since $\mathtt{LT}_{\mathcal{S}}(0, y) = 1$ for any non-zero $y \in \mathbb{F}_{p^d}$ and zero otherwise, we obtain $\mathtt{LT}_{\mathcal{S}}(0, y) = \chi(y)$. Hence, the total degree of the interpolation polynomial is at least $p^d - 1$, which might be prohibitive in practice.

Tan et al. [37] proposed an alternative approach where $\mathcal{S} = [0, p-1]$. If input integers a and b belong to $\mathcal{S}$, the result of $\mathtt{LT}_{\mathcal{S}}$(a,b) is computed with its interpolation polynomial over $\mathbb{F}_p$. If a, b are larger, e.g. $a, b \in [0, p^{d'-1}]$, they are encoded into $\mathbb{F}_{p^d}$ as above, but their comparison is performed via the lexicographic order defined on $\mathbb{F}_p^{d'}$. This method is based on the extraction of coefficients $a_i, b_i \in \mathbb{F}_p$ thanks to the following result from the theory of finite fields (see [31, Theorem 2.24] for the proof).

Lemma 2. *The linear transformations from $\mathbb{F}_{p^d}$ to $\mathbb{F}_p$ are exactly the mappings $L_\alpha(x) = \mathrm{Tr}_{\mathbb{F}_{p^d}/\mathbb{F}_p}(\alpha x)$ for some $\alpha \in \mathbb{F}_{p^d}$. Furthermore, $L_\alpha \neq L_\beta$ if $\alpha \neq \beta$.*

This lemma implies that for any $i \in [0, d' - 1]$ there exist $\alpha_i \in \mathbb{F}_{p^d}$ such that $L_{\alpha_i}(a) = a_i$ for any a from the vector subspace $\mathbb{F}_p^{d'}$ of $\mathbb{F}_{p^d}$. Such α_i's can be computed by solving the following system of equations over $\mathbb{F}_{p^d}$

$$\mathbf{X}\mathbf{A} = \mathbf{I}_d$$

where

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x & x^p & \dots & x^{p^{d-1}} \\ \vdots & \vdots & \ddots & \vdots \\ x^{d-1} & x^{(d-1)p} & \dots & x^{(d-1)p^{d-1}} \end{pmatrix},$$

$$\mathbf{A} = \begin{pmatrix} \alpha_0 & \alpha_1 & \dots & \alpha_{d-1} \\ \alpha_0^p & \alpha_1^p & \dots & \alpha_{d-1}^p \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{p^{d-1}} & \alpha_1^{p^{d-1}} & \dots & \alpha_{d-1}^{p^{d-1}} \end{pmatrix}$$

and $\mathbf{I}_d \in \mathbb{F}_{p^d}^{d \times d}$ is the identity matrix. Hence, the ith column of $\mathbf{X}^{-1}$ contain the powers $\alpha_i, \alpha_i^p, \dots, \alpha_i^{p^{d-1}}$, which define the linear map L_{α_i}.

Given the input encodings $\sum_{i=0}^{d'-1} a_i X^i$ and $\sum_{i=0}^{d'-1} b_i X^i$, we can extract and then compare their vectors of coefficients $\boldsymbol{a} = (a_0, a_1, \dots, a_{d'-1})$ and $\boldsymbol{b} = (b_0, b_1, \dots, b_{d'-1}) \in \mathbb{F}_p^{d'}$ using the lexicographical order $<$ on $\mathbb{F}_p^{d'}$ defined by

$$\boldsymbol{a} < \boldsymbol{b} \Leftrightarrow \exists i \in [0, d' - 1] \text{ such that } a_i < b_i \text{ and } a_j = b_j \quad \forall j > i\,.$$

The corresponding less-than function is equal to

$$\mathtt{LT}_{\mathcal{S}^{d'}}(\boldsymbol{a},\boldsymbol{b}) = \sum_{i=0}^{d'-1} \mathtt{LT}_{\mathcal{S}}(a_i,b_i) \prod_{j=i+1}^{d'-1} \mathtt{EQ}_{\mathcal{S}}(a_j,b_j)\,,$$

whereas the equality function is defined by

$$\mathtt{EQ}_{\mathcal{S}^{d'}}(\boldsymbol{a},\boldsymbol{b}) = \prod_{i=0}^{d'-1} \mathtt{EQ}_{\mathcal{S}}(a_i,b_i)\,,$$

Notice that the above construction is generic for any set $\mathcal{S}$ embedded into $\mathbb{F}_p$. For example, if $\mathcal{S} = [0, s-1]$ for some $s < p$, then one can encode input integers via decomposition in base s and compare them using $\mathtt{LT}_{\mathcal{S}^{d'}}$.

Comparison of large integers When the size of input integers exceeds $|\mathcal{S}|^d$, we can decompose integers in base $|\mathcal{S}|^{d'}$ and then compare their vectors of digits using the lexicographical order $<$ on $\left(\mathbb{F}_p^{d'}\right)^l$, for some $d' \leq d$. In fact, we compute two lexicographical orders on top of each other.

Let $a, b \in [0, |\mathcal{S}|^{ld'} - 1]$ be input integers. We represent an integer $a = \sum_{i=0}^{l} a_i |\mathcal{S}|^{id'}$ by the vector $\mathbf{a} = (\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{l-1}) \in \left(\mathbb{F}_p^{d'}\right)^l$ of its digits of length l. The comparison of two integers a and b is thus equivalent to the comparison of their vector of digits $\mathbf{a} = (\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{l-1})$ and $\mathbf{b} = (\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{l-1})$ using the lexicographical order $<$ on $\left(\mathbb{F}_p^{d'}\right)^l$ defined as follows

$$\mathbf{a} < \mathbf{b} \Leftrightarrow \exists i \in [0, l-1] \text{ such that } \boldsymbol{a}_i < \boldsymbol{b}_i \text{ and } \boldsymbol{a}_j = \boldsymbol{b}_j \;\; \forall j > i\,.$$

As done in [37], we can employ $\mathtt{EQ}_{\mathcal{S}^{d'}}$ and $\mathtt{LT}_{\mathcal{S}^{d'}}$ to compute the corresponding less-than function $\mathtt{LT}(\mathbf{a}, \mathbf{b})$ as follows

$$\mathtt{LT}(\mathbf{a},\mathbf{b}) = \sum_{i=0}^{l-1} \mathtt{LT}_{\mathcal{S}^{d'}}(\boldsymbol{a}_i,\boldsymbol{b}_i) \prod_{j=i+1}^{l-1} \mathtt{EQ}_{\mathcal{S}^{d'}}(\boldsymbol{a}_j,\boldsymbol{b}_j). \tag{1}$$

2.3 Homomorphic Encryption

We are interested in homomorphic encryption schemes that support SIMD operations on their plaintexts. This section aims at giving the necessary background regarding these schemes.

Cyclotomic fields and Chinese Remainder Theorem Let m be a positive integer and $n = \varphi(m)$ where φ is Euler's totient function. Let $\mathcal{K} = \mathbb{Q}(\zeta_m)$ be the cyclotomic number field constructed by adjoining a primitive m-th root of unity

$\zeta_m \in \mathbb{C}$ to the field of rational numbers. The ring of integers of $\mathcal{K}$, denoted by $\mathcal{R}$, is isomorphic to $\mathbb{Z}[X]/\langle \Phi_m(X) \rangle$ where $\Phi_m(X)$ is the m-th cyclotomic polynomial. Let $p > 1$ be a prime number coprime to m, then $\Phi_m(X)$ splits modulo p into ℓ irreducible factors of same degree d, i.e. $\Phi_m(X) = F_1(X) \cdots F_\ell(X) \bmod p$. The degree d is actually the order of p modulo m, and $\ell = n/d$. The Chinese Remainder Theorem (CRT) states that in this case the following ring isomorphism holds:

$$\begin{aligned}\mathcal{R}_p &= \mathbb{Z}_p[X]/\langle \Phi_m(X) \rangle \\ &\cong \mathbb{Z}_p[X]/\langle F_1(X) \rangle \times \ldots \times \mathbb{Z}_p[X]/\langle F_\ell(X) \rangle\end{aligned}$$

For each $i \in [1, \ell]$ the quotient ring $\mathbb{Z}_p[X]/\langle F_i(X) \rangle$ is isomorphic to the finite field $\mathbb{F}_{p^d}$. Hence, the above isomorphism can be rewritten as $\mathcal{R}_p \cong \mathbb{F}_{p^d}^\ell$. We call every copy of $\mathbb{F}_{p^d}$ in this direct product a *slot*. Therefore, every element of $\mathcal{R}_p$ contains ℓ slots, which implies that an array of ℓ independent $\mathbb{F}_{p^d}$-elements can be encoded as a unique element of $\mathcal{R}_p$. The slot isomorphic to $\mathbb{Z}_p[X]/\langle F_i(X) \rangle$ is referred to as the ith slot.

Additions and multiplications of $\mathcal{R}_p$-elements results in the corresponding coefficient-wise operations of their respective slots. In other words, each ring operation on $\mathcal{R}_p$ is applied to every slot in parallel, which resembles the Single-Instruction Multiple-Data (SIMD) instructions used in parallel computing. Therefore, the above encoding method from $\mathbb{F}_{p^d}^\ell$ to $\mathcal{R}_p$ is often called the *SIMD packing*.

The HE schemes that support SIMD packing and *exact* computations over encrypted data include `BGV` [11] and `FV` [24]. These schemes have a common framework described below.

Basic setup Let λ be the security level of an HE scheme. Let L be the maximal multiplicative depth of homomorphic circuits we want to evaluate. Let d be the order of the plaintext modulus p modulo the order m of $\mathcal{R}$. Assume that the plaintext space $\mathcal{R}_p$ has ℓ SIMD slots, i.e. $\mathcal{R}_p \cong \mathbb{F}_{p^d}^\ell$. The basic part of any HE schemes consists of key generation, encryption and decryption algorithms.

$\texttt{KeyGen}(1^\lambda, 1^L) \to (\texttt{sk}, \texttt{pk})$. Given λ and L, this function outputs the secret key `sk` and the public key `pk`.

$\texttt{Encrypt}(\texttt{pt} \in \mathcal{R}_p, \texttt{pk}) \to \texttt{ct}$. The encryption algorithm takes a plaintext `pt` and the public key `pk` and outputs a ciphertext `ct`.

$\texttt{Decrypt}(\texttt{ct}, \texttt{sk}) \to \texttt{pt}$. The decryption algorithm takes a ciphertext `ct` and the secret key `sk` and returns a plaintext `pt`. For freshly encrypted ciphertexts, the decryption correctness means that $\texttt{Decrypt}(\texttt{Encrypt}(\texttt{pt}, \texttt{pk}), \texttt{sk}) = \texttt{pt}$.

Homomorphic operations The homomorphic addition (multiplication) algorithm takes two input ciphertexts $\texttt{ct}_1$ and $\texttt{ct}_2$ encrypting plaintexts $\texttt{pt}_1$ and $\texttt{pt}_2$ respectively. It outputs a ciphertext `ct` that encrypts the sum (product) of these plaintexts in the ring $\mathcal{R}_p$. It implies that homomorphic addition (multiplication) sums (multiplies) respective SIMD slots of $\texttt{pt}_1$ and $\texttt{pt}_2$. Similar operations between ciphertexts and plaintexts are defined as well.

Every homomorphic ciphertext contains a special component called *noise* that is removed during decryption. However, the decryption function can deal only with noise of small enough magnitude; otherwise, this function fails. This noise bound is defined by encryption parameters in a way that larger parameters result in a larger bound. The ciphertext noise increases after every homomorphic operation and, therefore, approaches its maximal possible bound. It implies that to reduce encryption parameters one needs to avoid homomorphic operations that significantly increase the noise. Therefore, while designing homomorphic circuits, we need to take into account not only the running time of homomorphic operations but also their effect on the noise.

The most expensive homomorphic operation with relation to both noise and running time is ciphertext-ciphertext multiplication (`Mul`). This operations takes place when two expressions containing input values are multiplied. Such multiplication is called *non-scalar*. In contrast, ciphertext-plaintext multiplication (`MulPlain`) is used when an expression with input values is multiplied by an unencrypted or publicly known value. This is a *scalar* multiplication. Since `Mul` is much more expensive than `MulPlain`, the multiplicative depth and complexity of a homomorphic circuit is usually calculated with relation to the number of `Mul`'s, or non-scalar multiplications. Thus, in the following sections we focus on the non-scalar complexity of comparison circuits.

3 Optimising the comparison circuits over $\mathbb{F}_p$

In this section, we study the structure of basic comparison circuits over $\mathbb{F}_p$ used in Section 2 to compare integers, namely $\texttt{LT}_{\mathcal{S}}$ and $\texttt{EQ}_{\mathcal{S}}$ for some $\mathcal{S} \subseteq [0, p-1]$.

For any choice of $\mathcal{S}$, the corresponding equality function over $\mathbb{F}_p$ is equal to

$$\texttt{EQ}_{\mathcal{S}}(x, y) = 1 - (x - y)^{p-1}.$$

Unfortunately, $\texttt{LT}_{\mathcal{S}}$ is not that simple and universal and we have to rely on Lagrange interpolation (Lemma 1) to compute it. Yet, there are two different ways to evaluate it. The first method (as done in [37]) uses $\mathcal{S} = [0, p-1]$ and directly interpolates $\texttt{LT}_{\mathbb{F}_p}(x, y)$ as a bivariate polynomial over $\mathbb{F}_p$. The second approach (as done in [33] and [35]) has $\mathcal{S} = [0, (p-1)/2]$ and interpolates a univariate polynomial of $\texttt{LT}_{\mathcal{S}}(z, 0)$ with $z = x - y$. In this section, we show how to exploit the structure of these polynomials to speed-up their evaluation.

3.1 Bivariate interpolation of $\texttt{LT}_{\mathcal{S}}$.

Let $\mathcal{S} = [0, p-1]$. The less-than function can be interpolated using Lemma 1 and the following truth table.

$<$	0	1	2	$\cdots$	$p-1$
0	0	1	1	$\cdots$	1
1	0	0	1	$\cdots$	1
2	0	0	0	$\cdots$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$p-1$	0	0	0	$\cdots$	0

In particular, the interpolation polynomial of $\mathtt{LT}_{\mathcal{S}}$ over $\mathbb{F}_p$ is equal to

$$\begin{aligned} P_{\mathtt{LT}_{\mathcal{S}}}(X,Y) &= \sum_{a=0}^{p-2} \mathtt{EQ}_{\mathcal{S}}(X,a) \sum_{b=a+1}^{p-1} \mathtt{EQ}_{\mathcal{S}}(Y,b) \\ &= \sum_{a=0}^{p-2} \left(1-(X-a)^{p-1}\right) \sum_{b=a+1}^{p-1} \left(1-(Y-b)^{p-1}\right). \end{aligned}$$

Surprisingly, the total degree of $P_{\mathtt{LT}_{\mathcal{S}}}(X,Y)$ is only p and its coefficients can be described by the following theorem.

Theorem 1. *Let $p>2$ be a prime number and $\mathcal{S}=[0,p-1]$, then the interpolation polynomial of $\mathtt{LT}_{\mathcal{S}}$ over $\mathbb{F}_p$ has the following form*

$$P_{\mathtt{LT}_{\mathcal{S}}}(X,Y) = Y^{p-1} - \frac{p-1}{2}(XY)^{\frac{p-1}{2}} + \sum_{\substack{i,j>0,\\ i\neq j,\\ i+j\leq p}} a_{ij} X^i Y^j$$

where $a_{ij} = \sum_{a=0}^{p-2}\sum_{b=a+1}^{p-1} a^{p-1-i}b^{p-1-j} \in \mathbb{F}_p$. The total degree of $P_{\mathtt{LT}_{\mathcal{S}}}(X,Y)$ is p.

Proof. See Appendix A.

From the definition of $\mathtt{LT}_{\mathcal{S}}$, one can easily prove the following facts about $P_{\mathtt{LT}_{\mathcal{S}}}$:

- $P_{\mathtt{LT}_{\mathcal{S}}}(X,0)=0$, thus Y divides $P_{\mathtt{LT}_{\mathcal{S}}}(X,Y)$;
- $P_{\mathtt{LT}_{\mathcal{S}}}(X,X)=P_{\mathtt{LT}_{\mathcal{S}}}(Y,Y)=0$, thus $(X-Y)$ divides $P_{\mathtt{LT}_{\mathcal{S}}}(X,Y)$;
- $P_{\mathtt{LT}_{\mathcal{S}}}(p-1,Y)=0$ thus $X+1$ divides $P_{\mathtt{LT}_{\mathcal{S}}}(X,Y)$.

Hence, there exists a bivariate polynomial $f(X,Y)$ of total degree $p-3$ over $\mathbb{F}_p$ such that:

$$P_{\mathtt{LT}_{\mathcal{S}}}(X,Y) = Y(X-Y)(X+1)f(X,Y). \tag{2}$$

The following theorem describes the structure of $f(X,Y)$.

Theorem 2. *Let p be an odd prime and $\mathcal{S}=[0,p-1]$. Let $P_{\mathtt{LT}_{\mathcal{S}}}(X,Y)$ be the interpolation polynomial of $\mathtt{LT}_{\mathcal{S}}$ over $\mathbb{F}_p$ and $P_{\mathtt{LT}_{\mathcal{S}}}(X,Y) = Y(X-Y)(X+1)f(X,Y)$. Then, for any $z\in\mathbb{F}_p$ we have*

$$f(z,z) = f(z,0) = f(p-1,z). \tag{3}$$

As a consequence, there exists $(p-1)/2$ *polynomials* $f_i(X)$ *over* $\mathbb{F}_p$, $0 \leq i \leq (p-3)/2$, *such that:*

$$f(X,Y) = \sum_{i=0}^{(p-3)/2} f_i(X) Z^i, \tag{4}$$

with $Z = Y(X-Y)$ *and* $\deg(f_i(X)) = p-3-2i$.

Since our proof of Theorem 2 is quite long and with no real interest for the purpose of this work, we defer it to an extended version of this paper. In our experiments, we used the decompositions (4) of $f(X,Y)$ only for small p (between 3 and 7), which you can find in Appendix B.

Complexity analysis. In [37], the authors proposed to evaluate $P_{\mathtt{LT}_S}(X,Y)$ by evaluating each monomials separately before summing them up. Given $x, y \in \mathbb{F}_p$, they precompute the powers of x and y up to $p-1$ for a total of $2p-4$ non-scalar multiplications. Then, another $p-1$ non-scalar multiplications are needed to evaluate each monomial $((\sum_i c_i X^i)Y^j)_j$ where c_i's are scalars in $\mathbb{F}_p$, before summing them together to get the final result. Overall their evaluation of $P_{\mathtt{LT}_S}(X,Y)$ requires $3p-5$ non-scalar multiplications.

Following this idea and using the decomposition of $f(X,Y)$ given in Eq. (4) one needs:

- 2 multiplications to compute $(X+1)Z$ if $p \geq 3$ or 1 multiplication when $p = 2$;

and then for $p \geq 5$:

- $p-4$ multiplications to compute the X^i's for $2 \leq i \leq p-3$ required to compute the terms $f_i(X)$;
- $(p-5)/2$ multiplications to compute the Z^i for $2 \leq i \leq (p-3)/2$;
- $(p-5)/2$ multiplications to compute the products $f_i(X) \cdot Z^i$;
- 1 final multiplication $(X+1)Z \cdot f(X,Y)$.

Overall, at most $2p-6$ non-scalar multiplications are needed to homomorphically evaluate $P_{\mathtt{LT}_S}(X,Y)$ for $p \geq 5$. This number can be slightly reduced by optimizing the way of computing f_i's. For instance, it can be done with only 6 multiplications for $p = 7$, (see Appendix B), which is smaller than $2p-6 = 8$.

Overall, for $p \geq 5$, our method saves $p+1$ multiplications over the method of Tan et al. [37]. However, the complexity of the bivariate circuit remains linear in p which is unpractical for performing homomorphic comparisons using large digits (i.e. a large p).

3.2 Univariate interpolation of $\mathtt{LT}_S$

Unlike bivariate polynomials, it is possible to evaluate univariate polynomials of degree $p-1$ in $\mathcal{O}(\sqrt{p})$ non-scalar multiplications using the Paterson-Stockmeyer

algorithm [34]. The Paterson-Stockmeyer algorithm has been used in various works related to homomorphic encryption in order to speed-up polynomial evaluation. As a recent example, it was applied by Shaul et al [35] in the context of top-k selection, which uses small-number comparison as a subroutine. However, in our case the study of the structure of $\mathtt{LT}_{\mathcal{S}}$ as a univariate polynomial will allow us to speed-up its evaluation for large p beyond what could be achieved using only the Paterson-Stockmeyer algorithm.

To evaluate $\mathtt{LT}_S$ as a univariate polynomial, we compute the difference $x - y$ of the two input values and check its sign. To compute the sign function using finite field arithmetic, we need to split finite field elements into two classes: negative ($\mathbb{F}_p^-$) and non-negative ($\mathbb{F}_p^+$). In addition, for any $x, y \in \mathcal{S}$ the following property should hold:

$$x - y \in \begin{cases} \mathbb{F}_p^+ \text{ if } \mathtt{LT}_{\mathcal{S}}(x, y) = 0, \\ \mathbb{F}_p^- \text{ if } \mathtt{LT}_{\mathcal{S}}(x, y) = 1. \end{cases}$$

It is easy to see that these constraints are satisfied by $\mathcal{S} = [0, (p-1)/2]$. Let us split $\mathbb{F}_p$ into $\mathbb{F}_p^+ = [0, (p-1)/2]$ and $\mathbb{F}_p^- = [-(p-1)/2, -1]$. Notice that for any $x, y \in \mathcal{S}$, their difference $x - y$ belongs to $\mathbb{F}_p^-$ if and only if $x < y$.

Let $\chi_{\mathbb{F}_p^-}(z)$ be a function that outputs 1 if z is negative and 0 otherwise. According to Lemma 1, $\chi_{\mathbb{F}_p^-}(z)$ is equal to

$$\chi_{\mathbb{F}_p^-}(z) = \sum_{a=-\frac{p-1}{2}}^{-1} 1 - (z - a)^{p-1}.$$

Combining the above facts, the $\mathtt{LT}_{\mathcal{S}}$ function can be interpolated by the following polynomial over $\mathbb{F}_p$

$$Q_{\mathtt{LT}_{\mathcal{S}}}(X, Y) = \sum_{a=-\frac{p-1}{2}}^{-1} 1 - (X - Y - a)^{p-1}.$$

The following theorem describes all the coefficients of this interpolation polynomial.

Theorem 3. *For an odd prime p and $\mathcal{S} = [0, (p-1)/2]$, the $\mathtt{LT}_{\mathcal{S}}$ function can be interpolated by the following polynomial over $\mathbb{F}_p$*

$$Q_{\mathtt{LT}_{\mathcal{S}}}(X, Y) = \frac{p+1}{2}(X - Y)^{p-1} + \sum_{i=1, odd}^{p-2} c_i (X - Y)^i. \tag{5}$$

where $c_i = \sum_{a=1}^{\frac{p-1}{2}} a^{p-1-i}$.

Proof. See Appendix C

Remark 1. The polynomial $Q_{\mathtt{LT}_{\mathcal{S}}}(X,Y)$ yields the interpolation polynomial of the sign function $\mathrm{sgn}_{\mathcal{S}'}$ defined on $\mathcal{S}' = [-(p-1)/2, (p-1)/2]$ as $\mathrm{sgn}_{\mathcal{S}'}(x) = 1$ if $x < 0$ and $\mathrm{sgn}_{\mathcal{S}'}(x) = 1$ if $x \geq 0$. In particular, we have

$$Q_{\mathrm{sgn}_{\mathcal{S}'}}(X) = Q_{\mathtt{LT}_{\mathcal{S}}}(X, 0)\,. \tag{6}$$

Complexity analysis. The above theorem implies that the less-than function can be expressed by a univariate polynomial of degree $p-1$. In general, such polynomials are evaluated in $p-1$ multiplications according to Horner's method.

To reduce the number of non-scalar multiplications, we can resort to the Paterson-Stockmeyer algorithm [34] that requires $\sqrt{2(p-1)}+\log_2(p-1)+\mathcal{O}(1)$ such multiplications. However, we can improve this complexity by exploiting the fact that the polynomial in (5) has only one coefficient with an even index, the leading one. Thus, if $Z = X - Y$, we can rewrite (5) as follows

$$\alpha_{p-1}Z^{p-1} + Z\sum_{i=0,\text{even}}^{p-3}\alpha_{i+1}Z^i = \alpha_{p-1}Z^{p-1} + Zg(Z^2)$$

where $\alpha_i = \sum_{a=1}^{\frac{p-1}{2}} a^{p-1-i}$ and $g(X)$ is a polynomial of degree $(p-3)/2$. To evaluate $g(X)$, the Paterson-Stockmeyer algorithm requires $\sqrt{p-3}+\log_2\left(\frac{p-3}{2}\right)+\mathcal{O}(1)$ non-scalar multiplications. Furthermore, the preprocessing phase of this algorithm computes the powers $Z^2, Z^4, \ldots, Z^{2k}$ and $Z^{4k}, Z^{8k}, \ldots, Z^{2^rk}$ with $2k(2^r - 1) = p-3$. We can use these powers to compute the leading term in r non-scalar multiplications, namely

$$Z^2 Z^{2k} Z^{4k} \cdots Z^{2^rk} = Z^{2+2k(2^r-1)} = Z^{2+p-3} = Z^{p-1}.$$

Since the optimal k is about $\sqrt{(p-3)/2}$, we obtain that r must be about $\log_2\sqrt{p-3}$. Hence, the total non-scalar complexity of evaluating (5) is equal to

$$\sqrt{p-3} + \frac{3\log_2(p-3)}{2} + \mathcal{O}(1).$$

Remark 2. A careful reader can notice that the leading term of (5) is equal to $(X-Y)^{p-1}$, which is the heaviest part of the equality circuit $\mathtt{EQ}_{\mathcal{S}}(X,Y)$. Thus, we can get $\mathtt{EQ}_{\mathcal{S}}(X,Y)$ almost for free (at the cost of one homomorphic subtraction) after evaluating $\mathtt{LT}_{\mathcal{S}}(X,Y)$, which saves $\mathcal{O}(\log(p-1))$ non-scalar multiplications.

This feature of the univariate circuit allows to compute all the equality operations while comparing large integers using the less-than function $\mathtt{LT}$ from (1). This saves $\mathcal{O}((d'-1)(k-1)\log(p-1))$ homomorphic multiplications, thus leading to a better running time than for the bivariate circuit.

The downside of the univariate circuit is that only $(1/2)^d$ of the plaintext space is used to encode input integers in comparison to the bivariate method.

3.3 Min/max function

Given the less-than function $\texttt{LT}$ defined on some set, one can compute the minimum of two elements x, y of this set in the following generic way

$$\begin{aligned}\min(x,y) &= x \cdot \texttt{LT}(x,y) + y \cdot (1 - \texttt{LT}(x,y)) \\ &= y + (x-y) \cdot \texttt{LT}(x,y)\,. \end{aligned} \tag{7}$$

Notice that the input difference $x - y$ naturally emerges in this expression, thus hinting that the univariate circuit from (5) might be useful here. Indeed, by replacing $X - Y$ with a variable Z we obtain the univariate polynomial representation of the minimum function on the set $\mathcal{S} = [0, (p-1)/2]$

$$\begin{aligned} Q_{\min_{\mathcal{S}}}(X,Y) &= Y + Z \cdot Q_{\texttt{LT}_S}(X,Y) \\ &= Y + \frac{p+1}{2} Z + \sum_{i=1}^{\frac{p-1}{2}} Z^{2i} \sum_{a=1}^{\frac{p-1}{2}} a^{p-2i} \\ &= \frac{p+1}{2}(X+Y) + \sum_{i=1}^{\frac{p-1}{2}} Z^{2i} \sum_{a=1}^{\frac{p-1}{2}} a^{p-2i} \\ &= \frac{p+1}{2}(X+Y) + g(Z^2)\,, \end{aligned}$$

where $g(X)$ is a polynomial of degree $(p-1)/2$. As a result, $\min_S(x,y)$ can be computed with $\mathcal{O}(\sqrt{p-1})$ non-scalar multiplications via the Paterson-Stockmeyer algorithm.

Following the above reasoning, the maximum function can be computed with the following polynomial

$$Q_{\max_{\mathcal{S}}}(X,Y) = \frac{p+1}{2}(X+Y) - \sum_{i=1}^{\frac{p-1}{2}} Z^{2i} \sum_{a=1}^{\frac{p-1}{2}} a^{p-2i}.$$

Remark 3. Maximum and minimum functions are basic building blocks in the design of neural networks. For example, one of the most popular activation functions in neural networks is the rectifier, or ReLU, which is equal to $\max(x, 0)$. By analogy with (7), we have $\max(x,0) = x \cdot (1 - \mathrm{sgn}_{\mathcal{S}'}(x))$ where $\mathcal{S}' = [-(p-1)/2, (p-1)/2]$ (see Remark 1). Thus, Eq. (6) yields the following interpolation polynomial of the ReLU function on $\mathcal{S}'$

$$\begin{aligned} Q_{\mathrm{ReLU}_{\mathcal{S}'}}(X) &= X \cdot (1 - Q_{\mathrm{sgn}_{\mathcal{S}'}}(X)) \\ &= \frac{p+1}{2} X - \sum_{i=1}^{\frac{p-1}{2}} X^{2i} \sum_{a=1}^{\frac{p-1}{2}} a^{p-2i}. \end{aligned}$$

3.4 Impact on the overall complexity

In this section, we summarize the complexities of our method as compared to the work of Tan et al. for the evaluation of the less-than function using the lexicographical order method described in Section 2.

For fixed p and d, Tan et al. [37, Section 4.4] determined that the depth of the circuit evaluating the less-than function of two b-bits integers is equal to

$$\lfloor \log_2 d \rfloor + \lfloor \log_2(p-1) \rfloor + \lfloor \log_2(\log_p 2^b)/d) \rfloor + 4. \tag{8}$$

Note that our algorithms do not decrease the depth of the circuit. Similarly, Tan et al. showed that the number of homomorphic multiplications required to evaluate the less-than function of two b-bit integers with the bivariate interpolation is

$$d \cdot (T + \lceil \log_2(p-1) \rceil + \lceil \log_2 d \rceil) + \lfloor \log_2(\log_p 2^b)/d) \rfloor + 2.$$

where T is the number of homomorphic multiplications required to evaluate the comparison circuit over $\mathbb{F}_p$. In the work of Tan et al. $T = 3p - 5$, while in our case it is $T = 2p - 6$.

The univariate method saves even more multiplications since one can extract $1 - \mathtt{EQ}_{\mathcal{S}}(x, y)$ while computing $\mathtt{LT}_{\mathcal{S}}(x, y)$. Hence, we obtain $\mathtt{EQ}_{\mathcal{S}}(x, y)$ almost for free when evaluating the lexicographical order (Remark 2). Thus, in this case the comparison of two b-bits integers requires

$$d \cdot (T + \lceil \log_2 d \rceil) + \lfloor \log_2(\log_p 2^b)/d) \rfloor + 2.$$

with $T \approx \sqrt{2p-4} + 3(\log_2(2p-4))/2$ by using the Paterson-Stockmeyer algorithm.

4 Applications

The previous results can help improving the performance of any task involving comparisons performed homomorphically such as private database queries, k-nearest neighbour search, top-k selection or step function evaluation in neural nets. In this section we choose to demonstrate the gain brought by our approach for sorting and min/max search which are subroutines needed for the aforementioned tasks.

4.1 Sorting

To demonstrate the efficiency of our comparison algorithms, we applied them to a popular computational task that demands multiple comparisons, sorting. The best homomorphic sorting algorithm in terms of running time is the direct sorting algorithm due to Çetin et al. [12]. For a given array $A = [a_0, \ldots, a_{N-1}]$, this algorithm computes a *comparison matrix* $\mathbf{L}$ defined by

$$\mathbf{L}_{ij} = \begin{cases} \mathtt{LT}(a_i, a_j) & \text{if } i < j, \\ 0 & \text{if } i = j, \\ 1 - \mathtt{LT}(a_j, a_i) & \text{if } i > j. \end{cases}$$

Example : for $A = [5, 1, 7, 2, 3]$, the matrix $L \in \{0,1\}^{5\times 5}$ is given by:

$$\mathbf{L} = \begin{pmatrix} 0\,0\,1\,0\,0 \\ 1\,0\,1\,1\,1 \\ 0\,0\,0\,0\,0 \\ 1\,0\,1\,0\,1 \\ 1\,0\,1\,0\,0 \end{pmatrix}$$

It is easy to see that the Hamming weight of the ith row of $\mathbf{L}$ is unique and equal to the array index of a_i after sorting the array A in the descending order. For example, the zero weight indicates that there are no elements of A bigger than a_i. Thus, a_i has a zero index in A after sorting; in other words, a_i is the maximum element of A.

Let A' be a sorted version of A in the descending order. To compute $A'[i]$ for any $i \in [0, N-1]$, we homomorphically select an element a_j such that $\mathtt{wt}(\mathbf{L}[j]) = i$. This can be done with the following sum

$$A'[i] = \sum_{j=0}^{N-1} \mathtt{EQ}_{[0,N-1]}(i, \mathtt{wt}\,(\mathbf{L}[j])) \cdot a_j\,. \tag{9}$$

Note that the equality function should be defined on the set $[0, N-1]$, which implies that N must be smaller than the plaintext modulus p.

Remark 4. Since the matrix $\mathbf{L}$ is defined by $N(N-1)/2$ elements, it can be costly to keep it in memory for large N. Instead, we can compute the Hamming weights of its rows by iteratively computing one comparison $\mathtt{LT}(a_i, a_j)$ with $i < j$ at a time. To achieve this, we create an array of size N initialized with zeros that eventually will store the Hamming weights. Then, we add the outcome of $\mathtt{LT}(a_i, a_j)$ to the ith element of this array and the result of $1 - \mathtt{LT}(a_i, a_j)$ to the jth element. In this approach, only N elements of the Hamming weight array are being kept in RAM.

The direct sorting algorithm requires $N(N-1)/2$ less-than operations to compute the matrix $\mathbf{L}$ and N^2 equality operations to compute sorted elements of A'. While computing equalities, we can reduce the total number of non-scalar multiplications if N is large enough. Recall $\mathtt{EQ}_{\mathcal{S}}$ needs $M = \log_2(p-1) + \mathtt{wt}(p-1) - 1$ non-scalar multiplications for any $\mathcal{S} \subseteq [0, p-1]$. Hence, to compute $\mathtt{EQ}_{[0,N-1]}(i, \mathtt{wt}\,(\mathbf{L}[j]))$ for all $i \in [0, N-1]$, we should perform NM multiplications. Using Lemma 3 (see Appendix A), we can rewrite:

$$\mathtt{EQ}_{[0,N-1]}\,(i, \mathtt{wt}\,(\mathbf{L}[j])) = 1 - \sum_{k=0}^{p-1} i^k \cdot \mathtt{wt}\,(\mathbf{L}[j])^{p-1-k}\,.$$

If we precompute the powers $\mathtt{wt}\,(\mathbf{L}[j])^{p-1-k}$, then we need only $p-2$ non-scalar multiplications to compute all the equalities $\mathtt{EQ}(i, \mathtt{wt}\,(\mathbf{L}[j]))$ as the index i is not encrypted. Hence, if $N > (p-2)/M$, this approach results in a smaller

number of non-scalar multiplications. Yet, this method introduces $p-1$ scalar multiplications (by powers i^k) and $p-2$ additions. However, these operations are much faster in HE schemes than non-scalar multiplication such that the gain from reducing non-scalar multiplications becomes dominant.

The main advantage of direct sorting is that its multiplicative depth is independent of the array length, namely $d = d(\texttt{LT}) + \lceil \log_2(p-1) \rceil + 1$ with $d(\texttt{LT})$ given in Equation (**??**).

This allows to avoid large encryption parameters and costly bootstrapping operations.

4.2 Minimum and maximum of an array

Another application of our comparison algorithms is concerned with finding a minimum (or maximum) element of an array. To find the minimum of an array with N elements, at least $N-1$ calls of the pairwise minimum function are required [21, Chapter 9], which can be achieved, for instance, by the *tournament method*.

The tournament method consists of $\lceil \log N \rceil$ iterations. In each iteration, the input array is divided into pairs. If the array length is odd, one element is stashed for the next iteration. Then, the maximum of each pair is removed from the array. The algorithm stops when only one element is left; this is the minimum of the input array, see Figure 1. Unfortunately, the tournament method has a

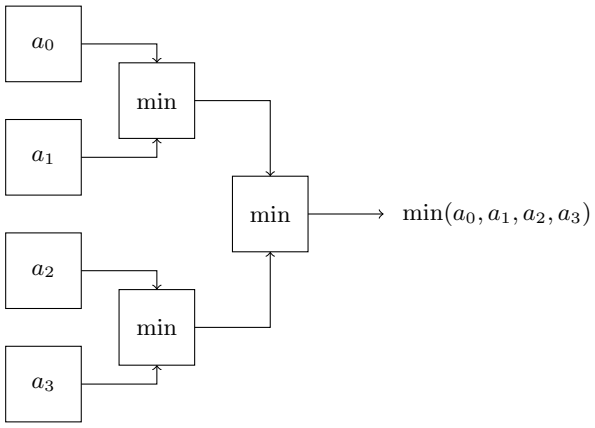

Fig. 1: The tournament method of finding the minimum of an array. In each stage, the array elements are divided into pairs. Only minimum of a pair go to the next stage.

big multiplicative complexity, namely $\lceil \log N \rceil \cdot d(\min(x, y))$. In the HE world, this enforces us to use either impractical encryption parameters [38] or a slow bootstrapping function.

To reduce the depth of the array minimum algorithm, we can combine the tournament method and direct sorting. Let $A = [a_0, \ldots, a_{N-1}]$ be an input array. First, we perform T iterations of the tournament algorithm, which leaves us with an array $A' = [a'_0, \ldots, a'_{N'-1}]$ of length $N' = \lceil N/2^T \rceil$ containing minimal elements of A. Then, we can find the minimum by computing the comparison table $\mathbf{L}$ as in direct sorting and extracting one of the minimal elements. If $M(f)$ is the non-scalar multiplicative complexity of a function f, then the total number of non-scalar multiplications to find the minimum of an array is approximately equal to

$$(N - N') \cdot M\left(\min(x, y)\right) + \frac{N'(N'-1)}{2} \cdot M\left(\texttt{LT}\right) + M(\texttt{Extraction}).$$

The extraction of a minimum element can be done with two methods. In the first approach, we use the fact that the Hamming weight of the comparison table row corresponding to the minimum is equal to $N' - 1$. Hence, we can retrieve the minimum as in (8)

$$\min(A') = \sum_{i=0}^{N'-1} \texttt{EQ}_{[0,N'-1]}(N' - 1, \texttt{wt}\left(\mathbf{L}[i]\right)) \cdot a'_i. \tag{10}$$

Here, the multiplicative depth is equal to

$$T \cdot d(\min(x, y)) + d(\texttt{LT}) + \lceil \log_2(p-1) \rceil + 1$$

which is independent of the input array length N. Since $\texttt{EQ}_{[0,N'-1]}$ need $\log_2(p-1) + \texttt{wt}(p-1) - 1$ non-scalar multiplications, then the number of non-scalar multiplications needed to extract the array minimum is equal to

$$M(\texttt{Extraction}) = N'(\log_2(p-1) + \texttt{wt}(p-1) - 1).$$

Shaul et al. [35] proposed another circuit to extract the array minimum that exploits the fact that the comparison table row related to the minimum contains only 1 except for the main diagonal entry. In other words, the product of $\prod_{j=1, j\neq i}^{N'} \mathbf{L}_{ij} = 1$ if and only if $a'_i = \min(A')$. Hence, the minimal element is equal to

$$\min(A') = \sum_{i=0}^{N'-1} a'_i \cdot \prod_{j=1, j\neq i}^{N'} \mathbf{L}_{ij}. \tag{11}$$

The resulting depth of this circuit amounts to

$$T \cdot d\left(\min(x, y)\right) + d(\texttt{LT}) + \lceil \log(N' - 1) \rceil + 1.$$

This extraction circuit requires the following number of multiplications

$$M(\texttt{Extraction}) = N'(N' - 2).$$

This implies that for small enough N', Shaul's circuit (10) has a smaller depth or/and a smaller multiplication complexity than the circuit in (9). Furthermore, Shaul's circuit supports any length $N' > p$, whereas (9) requires $N' \leq p$ such that $\mathtt{wt}(\mathbf{L}[i])$ do not overflow modulo p.

In the experiments conducted in Section 5, we use the best of these approaches for given N, T and p.

5 Implementation Results

We implemented the lexicographic order algorithm (`LT`, Eq. 1) using the BGV scheme [11]. The code is written in the HElib library [27]. For a fair comparison with the prior work, we also implemented the algorithm of Tan et al. [37]. The code is publicly available via `https://github.com/iliailia/comparison-circuit-over-fq`. In all the experiments, we used an average commodity laptop equipped with an Intel Dual-Core i5-7267U CPU (running at 3.1 GHz) and 8 GB of RAM. Multi-threading was turned off.

In the results presented below, the following notation is used:

p : the plaintext modulus of the BGV scheme;
q : the initial ciphertext modulus of the BGV scheme;
m : the cyclotomic order of the ring $\mathcal{R}$;
n : the degree of the ring $\mathcal{R}$ ($n = \phi(m)$);
ℓ : the number of SIMD slots;
d : the dimension of a slot subspace used for digit encoding (d' in Eq. 1);
l : the dimension of digit vectors encoding input integers over $\mathbb{F}_p^d$;
k : the number of input integers encoded in one ciphertext ($k = \lfloor \ell/l \rfloor$).

In all the experiments, the encryption parameters of BGV are chosen according to the following strategy. The plaintext modulus p is a prime number such that SIMD slots are isomorphic to a finite field. Next, we choose the order m of $\mathcal{R}$ with large enough n to support our homomorphic algorithms ($n > 12000$). The order of p modulo m should be as small as possible, which maximizes the number of SIMD slots, thus reducing the amortized running time of our algorithms. In addition, m is chosen to be a prime number or a product of a few primes. This constraint makes sure that the slot permutation group is cyclic or a product of a few cyclic groups, which results in a better performance (for more details, see [26, Appendix C.3]).

Every input integer is encoded into exactly one ciphertext. First, it is decomposed into a vector of l digits over $\mathbb{F}_p^d$ (see Section 2.2). Then each digit embedded into one SIMD slot. Thus, each ciphertext encrypts exactly $k = \ell/l$ integers.

Note that although our algorithms save some homomorphic multiplications as compared to [37], the depth of the circuits remains unchanged. Since the ciphertext size depends mainly on the depth of the circuit one wants to evaluate and the desired level of security, we obtain similar bandwidth requirements than previous works.

To compare our algorithms with the state of the art, we ran the lexicographic order algorithm (`LT`) to compute the less-than function on 64-bit integers. The results of these experiments are presented in Table 1. In addition, we ran the direct sorting and the array minimum algorithms from Section 4; see Tables 3 and 4, respectively.

We ran our algorithms with $p \in [3, 659]$. However, Table 1 contains the best results for small p's, where the bivariate and the univariate comparison circuits have a comparable running time. Table 2 contains the results with the best observed amortized running time. For the sorting and the array minimum applications (Tables 3 and 4), we showed only the results with p and m supporting reasonable security levels and giving the best running time for arrays of length $N = 64$.

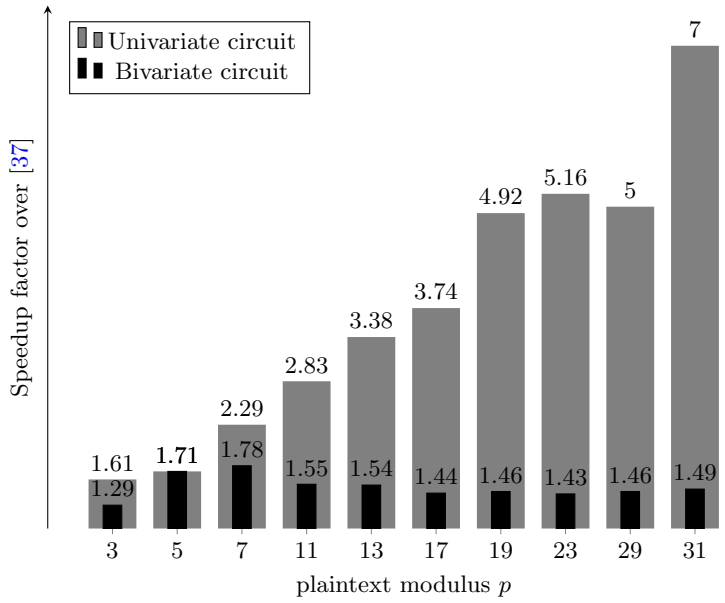

Fig. 2: **Less-than function via different methods.** The running time speedup factor of our lexicographic order algorithms over the algorithm of Tan et al. [37]. These factors are computed using the data from Table 1.

As shown in Table 1 and Figure 2, our lexicographic order circuits have a better running time than the prior work of Tan et al. [37] even for small plaintext moduli. In particular, using our bivariate circuit, the less-than function is 1.71 times faster than with the circuit of Tan et al. on their fastest set of parameters ($(p, d, l) = (5, 7, 4)$). The best running time per integer was achieved by our univariate circuit, which outperforms any bivariate circuit for any $p > 5$ at the cost of larger encryption parameters. As shown in Figure 2, the speedup

(p, m, n)	Type	(d, l)	$\log_2 q$	λ	k	Total time, s	Amortized time per slot, ms
$(3, 34511, 34510)$	[37]	$(6, 7)$	324	298	290	26.24	90
	B	$(6, 7)$	324	298	290	20.17	70
	U	$(16, 4)$	472	189	507	28.45	56
$(5, 19531, 19530)$	[37]	$(7, 4)$	324	155	697	24.97	36
	B	$(7, 4)$	324	155	697	14.50	21
	U	$(7, 6)$	354	141	465	9.89	21
$(7, 20197, 19116)$	[37]	$(6, 4)$	354	137	531	37.50	71
	B	$(6, 4)$	354	137	531	21.35	40
	U	$(8, 4)$	406	110	531	16.53	31
$(11, 15797, 15796)$	[37]	$(5, 4)$	342	162	359	35.20	99
	B	$(5, 4)$	342	162	359	22.76	64
	U	$(5, 5)$	378	145	287	9.79	35
$(13, 30941, 30940)$	[37]	$(5, 4)$	354	338	1547	82.02	54
	B	$(5, 4)$	354	338	1547	54.05	35
	U	$(4, 6)$	378	313	1031	15.56	16
$(17, 41761, 41760)$	[37]	$(4, 4)$	413	402	1305	130.81	101
	B	$(4, 4)$	413	402	1305	91.14	70
	U	$(7, 3)$	472	344	1740	45.29	27
$(19, 29989, 29988)$	[37]	$(4, 4)$	378	302	833	101.75	123
	B	$(4, 4)$	378	302	833	69.92	84
	U	$(5, 4)$	385	296	833	20.76	25
$(23, 37745, 30192)$	[37]	$(5, 3)$	413	275	838	194.41	232
	B	$(5, 3)$	413	275	838	135.40	162
	U	$(9, 2)$	456	245	1258	56.49	45
$(29, 18157, 17820)$	[37]	$(5, 3)$	360	175	990	103.28	105
	B	$(5, 3)$	360	175	990	70.82	72
	U	$(6, 3)$	413	150	990	19.98	21
$(31, 52053, 34700)$	[37]	$(5, 3)$	512	252	2313	437.11	189
	B	$(5, 3)$	512	252	2313	293.27	127
	U	$(4, 4)$	512	252	1735	46.58	27

Table 1: **Less-than function via different methods.** The running time of our lexicographic order algorithms and the algorithm of Tan et al. [37] to compare 64-bit integers with encryption parameters supporting λ bits of security. The second column (Type) indicates which comparison circuit is used: the univariate (U), bivariate from this work (B) or bivariate one from [37]. The total time is averaged over 50 trials.

factor of the lexicographic order with the univariate circuit over [37] is increasing with the plaintext modulus. This trend is perturbed when $p = 23$ and 29 because the structure of the related slot permutation groups introduces additional

(p, m, n)	(d, l)	$\log_2 q$	λ	k	Total time, s	Amortized time per slot, ms
$(131, 17293, 17292)$	$(3, 4)$	431	96	1441	16.07	11
$(167, 28057, 28056)$	$(3, 4)$	494	146	2338	30.69	13
$(173, 30103, 30102)$	$(2, 5)$	521	148	2006	24.57	12

Table 2: **Less-than function via the univariate circuit.** The best empirical running time of our lexicographic order algorithm to compare 64-bit integers with encryption parameters supporting λ bits of security. The algorithm is based on the univariate circuit. The total time is averaged over 50 trials.

computational overhead and ciphertext noise, thus leading to larger encryption parameters.

Table 2 shows that for $p = 131$, the univariate circuit takes only 11 milliseconds to compare two 64-bit integers, which is more than 3 times faster than the best running time achieved by the circuit of Tan et al.

N	$\log_2 q$	#Trials	Average total time, s	Amortized time per slot, ms	Amortized time per slot, ms [12]
8-bit integers ($d = 2$, $l = 1$, $k = 9352$)					
4	589	32	186.28	20	140
8	599	20	867.46	93	690
16	599	20	3652.23	391	3140
32	599	20	14769.23	1579	13900
64	604	10	60351.02	6453	60000
32-bit integers ($d = 3$, $l = 2$, $k = 4676$)					
4	659	20	299.17	64	200
8	671	20	1356.19	290	944
16	671	20	5700.12	1219	4280
32	684	20	23017.03	4922	18600
64	684	10	89972.27	19241	49700

Table 3: **Sorting.** The running time needed to sort N 8-bit or 32-bit integers with $p = 167$, $m = 28057$ and $n = 28056$. The minimal security level is 92 bits according to the LWE estimator [3]. Note that the amortized timing per slot from [12] is obtained with the LTV scheme [32], which was attacked by Albrecht et al. [2].

Table 3 illustrates that our homomorphic sorting implementation achieves the best running time in the existing literature. Note that the best result [12] in this area is based on an SHE scheme that was successfully attacked by Albrecht et al. [2]. Hence, this result is hard to compare directly to our work.

N	T	$\log_2 q$	#Trials	Average total time, s	Amortized time per slot, ms
8-bit integers ($d = 3$, $l = 1$, $k = 5220$)					
2	1	232	20	12.35	2.37
4	2	406	20	50.55	9.68
8	3	579	20	151.75	29.07
16	4	766	20	386.87	74.11
32	3	825	20	883.68	169.29
64	3	854	20	2111.56	404.51
32-bit integers ($d = 6$, $l = 2$, $k = 2610$)					
2	1	406	20	37.71	14.54
4	2	753	20	157.80	60.46
8	1	825	20	506.24	193.96
16	1	839	20	1694.15	649.10
32	1	854	20	6440.27	2467.54
64	1	884	20	24986.04	9573.20

Table 4: **Array minimum.** The running time needed to find the minimum of N 8-bit or 32-bit integers with $p = 17$, $m = 41761$ and $n = 41760$. The parameter T denotes the number of the tournament method stages. The minimal security level is 121 bits according to the LWE estimator [3].

The existing literature on homomorphic array minimum/maximum algorithms [38,35] is based on the techniques described in Section 4.2. Hence, our improvement of comparison circuits automatically results in a better performance over these works. For example, Togan et al. [38] needed 346.9 seconds to find maximal elements of 960 arrays of 16 8-bit integers, which is 361 milliseconds per array. Our work can perform this task in 74 milliseconds, see Table 4. To find the minimum of 64 32-bit integers, our array minimum algorithm requires about 9.5 seconds.

5.1 Comparison to other HE schemes

As mentioned in the introduction, there are three types of HE schemes suitable in different use cases including TFHE, CKKS and BGV/BFV. Our algorithms are designed for BGV/BFV, which support SIMD packing and are the most efficient FHE schemes for exact computation in arithmetic circuits. However, the amortized running time per data value of our comparison algorithms is comparable to efficient FHE schemes for binary circuits (TFHE) and HE supporting approximate arithmetic over complex numbers (CKKS).

In Table 5, our implementation of the less-than function is compared to the implementations of this function in TFHE [19,20] and CKKS [15].

Chilloti et al. [19,20] constructed a deterministic weighted automata that can compute the maximum function using the TFHE scheme. The same automata

can compute the less-than function without any performance loss. In this case, the running time of the less-than function of two b-bit integers takes $170b$ microseconds on a hardware similar to ours and with the encryption parameters supporting at least 152 bits of security. This security level might be lowered by a recent attack of Espitau et al. [23]. Note that this running time is achieved in the leveled mode of TFHE, i.e. without bootstrapping. Unfortunately, we could not manage to run the code of this implementation on our machine.

Cheon et al. [15] designed a polynomial approximation of the less-than function over real numbers. Since the precision of this approximation depends on the ciphertext noise which cannot be reduced in CKKS, only a few consecutive comparisons are possible to perform correctly unlike in TFHE and BGV/BFV. Nevertheless, the number of SIMD slots in CKKS is always half of the ring dimension ($n/2$), which significantly reduces the amortized running time per value.

We ran the comparison method of Cheon et al. on our machine with multithreading turned off. Since the implementation in [15] uses parallelization with 8 cores, our running time of this method presented in Table 5 is significantly larger than in [15]. The encryption parameters of the CKKS scheme are set to support a security level of at least 128 bits.

Our implementation runs with $p = 131$, $m = 17293$ ($n = 17292$), which corresponds to 5764 integers of 8-16 bits or 2882 20-bit integers encoded into one ciphertext. The encryption parameters corresponds to a security level of at least 126 bits.

As shown in Table 5, our algorithm for the less-than function demonstrates a similar performance as the TFHE-based implementation and is up to 2 times faster than the CKKS-based work. This means that in use cases that involve arithmetic and non-arithmetic functions (e.g. artificial neural networks) one might resort to only an HE scheme supporting exact arithmetic circuits (i.e. BGV/BFV) instead of combining it with an HE scheme efficient for a non-arithmetic part of the computation (i.e. TFHE).

6 Conclusion

In this work, we constucted more efficient homomorphic circuits of comparison operations for the BGV and BFV FHE schemes. Our results are based on structural properties of comparison functions over finite fields. We proved that less-than functions of two input variables x and y can be represented either by bivariate polynomials or univariate polynomials (in variable $z = x - y$) with multiple zero coefficients, which simplifies computation. Moreover, our computation of the univariate less-than functions yields the output of the equality function almost for free, which allowed us to speed up the lexicographic order of vectors over finite fields and thus comparison of large integers encoded by these vectors.

The implementation of our circuits in HElib is faster than the state-of-the-art work [37] by more than a factor of 3. Furthermore, the running time of our circuits is comparable to implementations of comparison algorithms in TFHE,

Bit length	FHE scheme	Total time, s	Amortized time per slot, ms
8	TFHE	0.001*	1.36*
	CKKS	89.61	1.37
	BGV(this paper)	7.09	1.23
12	TFHE	0.002*	2.04*
	CKKS	127.54	1.95
	BGV(this paper)	7.09	1.23
16	TFHE	0.003*	2.72*
	CKKS	296.96	4.53
	BGV(this paper)	12.11	2.10
20	TFHE	0.003*	3.4*
	CKKS	373.76	5.70
	BGV(this paper)	8.66	3.01

Table 5: **Comparison with other HE schemes.** Total and amortized running time of the less-than function implemented with different HE schemes including TFHE, CKKS and BGV. Note that TFHE does not support SIMD packing, which implies that the total and amortized running time of this scheme are the same. The encryption parameters of each scheme are set to support the following security levels: 152 bits for TFHE (might be smaller due to [23]), 128 bits for CKKS and 126 bits for BGV.
*TFHE timings are estimated from [20].

which is believed to be the most efficient FHE scheme for non-arithmetic homomorphic computations. As a side contribution, we applied our comparison algorithms to the tasks of sorting and array minimum search. In both cases we achieved the best running time present in the literature. For instance, our sorting algorithm can sort 4676 batches of 64 32-bit integers in about 25 hours, which results in an amortized time equal to about 19 seconds per batch. Our minimum circuit can find minimal elements of 2610 batches of 64 32-bit integers in approximately 7 hours; the amortized time is 9.5 seconds per batch.

We hope that this work will draw attention to the study of arithmetic circuits over finite fields representing non-arithmetic functions over integers, thus leading to practically efficient homomorphic implementations of useful algorithms.

References

1. Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

2. Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 153–178. Springer, Heidelberg, August 2016.
3. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.
4. Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society Press, May 2018.
5. Pascal Aubry, Sergiu Carpov, and Renaud Sirdey. Faster Homomorphic Encryption is not Enough: Improved Heuristic for Multiplicative Depth Minimization of Boolean Circuits. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, pages 345–363, Cham, 2020. Springer International Publishing.
6. J. Bajard, P. Martins, L. Sousa, and V. Zucca. Improving the Efficiency of SVM Classification With FHE. *IEEE Transactions on Information Forensics and Security*, 15:1709–1722, 2020.
7. Joppe W. Bos, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Privacy-friendly forecasting for the smart grid using homomorphic encryption and the group method of data handling. In Marc Joye and Abderrahmane Nitaj, editors, *Progress in Cryptology - AFRICACRYPT 2017*, pages 184–201, Cham, 2017. Springer International Publishing.
8. Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020.
9. Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 483–512, Cham, 2018. Springer International Publishing.
10. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Heidelberg, August 2012.
11. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS 12, page 309325, New York, NY, USA, 2012. Association for Computing Machinery.
12. Gizem S. Çetin, Yarkin Doröz, Berk Sunar, and Erkay Savas. Depth optimized efficient homomorphic sorting. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 61–80. Springer, Heidelberg, August 2015.
13. J. H. Cheon, M. Kim, and M. Kim. Optimized Search-and-Compute Circuits and Their Application to Query Evaluation on Encrypted Data. *IEEE Transactions on Information Forensics and Security*, 11(1):188–199, 2016.
14. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.
15. Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. Cryptology ePrint Archive, Report 2019/1234, 2019. `https://eprint.iacr.org/2019/1234`, to appear in ASIACRYPT 2020.

16. Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun-Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 415–445. Springer, Heidelberg, December 2019.
17. Jung Hee Cheon, Miran Kim, and Myungsun Kim. Search-and-compute on encrypted data. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 142–159, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
18. 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, *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
19. 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, Heidelberg, December 2017.
20. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.
21. Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.
22. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
23. Thomas Espitau, Antoine Joux, and Natalia Kharchenko. On a hybrid approach to solve small secret LWE. Cryptology ePrint Archive, Report 2020/512, 2020. `http://eprint.iacr.org/2020/512`.
24. Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://eprint.iacr.org/2012/144`.
25. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
26. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully Homomorphic Encryption with Polylog Overhead. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 465–482, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
27. HElib: An implementation of homomorphic encryption (1.1.0). `https://github.com/homenc/HElib`, October 2020. IBM.
28. Shizuo Kaji, Toshiaki Maeno, Koji Nuida, and Yasuhide Numata. Polynomial expressions of p-ary auction functions. *Journal of Mathematical Cryptology*, 13(2):69–80, 2019.
29. M. Kim, H. T. Lee, S. Ling, and H. Wang. On the Efficiency of FHE-Based Private Queries. *IEEE Transactions on Dependable and Secure Computing*, 15(2):357–363, 2018.
30. Miran Kim and Kristin Lauter. Private Genome Analysis Through Homomorphic Encryption. *BMC medical informatics and decision making*, 15, December 2015.
31. Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1986.

32. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012.
33. H. Narumanchi, D. Goyal, N. Emmadi, and P. Gauravaram. Performance analysis of sorting of fhe data: Integer-wise comparison vs bit-wise comparison. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 902–908, 2017.
34. Michael S Paterson and Larry J Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.
35. Hayim Shaul, Dan Feldman, and Daniela Rus. Secure k-ish nearest neighbors classifier. *Proceedings on Privacy Enhancing Technologies*, 2020(3):42–61, 2020.
36. N. P. Smart and F. Vercauteren. Fully Homomorphic SIMD Operations. *Des. Codes Cryptography*, 71(1):5781, April 2014.
37. B. H. M. Tan, H. T. Lee, H. Wang, S. Q. Ren, and A. M. M. Khin. Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2020.
38. Mihai Togan, Luciana Morogan, and Cezar Plesca. Comparison-based applications for fully homomorphic encrypted data. *Proceedings of the Romanian Academy-Series A: Mathematics, Physics, Technical Sciences, Information Science*, 16:329, 2015.
39. Andrew C. Yao. Protocols for Secure Computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, page 160164, USA, 1982. IEEE Computer Society.

A Proof of Theorem 1

To prove Theorem 1 we need the following lemmas.

Lemma 3. *For all* $(a, b) \in \mathbb{Z}^2$ *we have:*

$$(a-b)^{p-1} = \sum_{i=0}^{p-1} a^i b^{p-1-i} \mod p.$$

Proof. Using the binomial theorem we obtain

$$(a-b)^{p-1} = \sum_{i=0}^{p-1} \binom{p-1}{i} a^i (-b)^{p-1-i}.$$

Computing the binomial coefficient modulo p

$$\begin{aligned}\binom{p-1}{i} &= \frac{(p-1)!}{i!(p-1-i)!} \\ &= \frac{(p-1)(p-2)\dots(i+1)}{1\cdot 2\dots(p-(1+i))} \\ &= (-1)^{p-1-i} \mod p\,,\end{aligned}$$

we prove the lemma.

Lemma 4. *Let $P(X)$ be a polynomial of degree d less than $p-1$. For any prime number $p > 2$, it holds*

$$\sum_{a=0}^{p-1} P(a) = 0 \mod p.$$

Proof. Since $\sum_{a=0}^{p-1} b = 0 \mod p$ for any $b \in \mathbb{F}_p$, it is enough to prove that the sum $\sum_{a=0}^{p-1} a^n = 0 \mod p$ for any $0 \leq n < p-1$. Since the case $n = 0$ is straightforward, let us assume $n > 0$. Let g be a primitive element of $\mathbb{F}_p$. Since $p > 2$, we have $g \neq 1$. Thus, we can rewrite the above sum as follows.

$$\sum_{i=1}^{p-1} g^{in} = \frac{g^{pn} - g^n}{g^n - 1}.$$

Since $g^{pn} \equiv g^n \mod p$, the sum turns into zero modulo p.

Now, we have all the ingredients to prove Theorem 1.

Proof (Proof of Theorem 1). Assume that all computations are done modulo p. Using Lemma 3, we obtain that $P_{\mathtt{LT}_{\mathcal{S}}}(X, Y)$ is equal to

$$\sum_{a=0}^{p-2} \left(1 - \sum_{i=0}^{p-1} X^i a^{p-1-i}\right) \sum_{b=a+1}^{p-1} \left(1 - \sum_{j=0}^{p-1} Y^j b^{p-1-j}\right)$$

Let us expand this expression distributively.

$$\sum_{a=0}^{p-2} \sum_{b=a+1}^{p-1} 1 - \sum_{i=0}^{p-1} X^i a^{p-1-i} - \sum_{i=0}^{p-1} Y^i b^{p-1-i} + \sum_{i=0}^{p-1} \sum_{j=0}^{p-1} X^i Y^j a^{p-1-i} b^{p-1-j}.$$

Let us compute individual polynomial coefficients. The constant term is equal to

$$\sum_{a=0}^{p-2} \sum_{b=a+1}^{p-1} 1 - a^{p-1} - b^{p-1} + a^{p-1} b^{p-1}$$
$$= \sum_{a=0}^{p-2} \sum_{b=a+1}^{p-1} 1 - a^{p-1} - 1 + a^{p-1} = 0\,.$$

Coefficients by X^i with $i > 0$ can be computed as follows

$$\sum_{a=0}^{p-2} \sum_{b=a+1}^{p-1} \left(-a^{p-1-i}\right) + a^{p-1-i} b^{p-1} = 0\,.$$

Next, we compute coefficients by Y^i with $i > 0$.

$$\begin{aligned}
&-\sum_{a=0}^{p-2}\sum_{b=a+1}^{p-1} b^{p-1-i} - a^{p-1}b^{p-1-i} \\
&\quad = -\sum_{b=1}^{p-1} b^{p-1-i} - \sum_{a=1}^{p-2}\sum_{b=a+1}^{p-1} b^{p-1-i} - b^{p-1-i} \\
&\quad = -\sum_{b=1}^{p-1} b^{p-1-i}.
\end{aligned}$$

If $i = p-1$, this sum is equal to 1. According to Lemma 4, it is 0 if $i < p-1$.

To compute coefficients by X^iY^j with $i, j > 0$, we will use Faulhaber's formula below

$$\sum_{k=1}^{n} k^e = \frac{1}{e+1}\sum_{i=1}^{e+1}(-1)^{\delta_{ie}}\binom{e+1}{i}B_{e+1-i}\cdot n^i\,,$$

where δ_{ie} is the Kronecker delta and B_i is the ith Bernoulli number. This implies that there exist a polynomial $P(X) \in \mathbb{F}_p[X]$ of degree $e+1$ such that

$$\sum_{k=1}^{n} k^e = P(n). \tag{12}$$

Note that $P(0) = 0$. The coefficient by X^iY^j for some positive i and j is equal to

$$\sum_{a=0}^{p-2}\sum_{b=a+1}^{p-1} a^{p-1-i}b^{p-1-j} = \sum_{b=1}^{p-1} b^{p-1-j}\sum_{a=0}^{b-1} a^{p-1-i}\,.$$

According to (11), there exist a polynomial $P_i(X)$ of degree $p-i$ such that $\sum_{a=0}^{b-1} a^{p-1-i} = P_i(b)$. Since $Q_{ij}(X) = X^{p-1-j}P_i(X)$ has degree $2p-1-i-j$, Lemma 4 implies that if $i + j > p$, then

$$\begin{aligned}
\sum_{b=1}^{p-1} b^{p-1-j}\sum_{a=0}^{b-1} a^{p-1-i} &= \sum_{b=1}^{p-1} b^{p-1-j}P_i(b) \\
&= \sum_{b=1}^{p-1} Q_{ij}(b) = \sum_{b=0}^{p-1} Q_{ij}(b) - Q_{ij}(0) = 0.
\end{aligned}$$

Thus, all the coefficient X^iY^j with $i + j > p$ are zero, which means that the total degree of $P_{\mathtt{LT}_{\mathcal{S}}}(X, Y)$ is at most p.

In addition, we consider the case when $i = j$ and $i, j \le (p-1)/2$. Let us consider the following sum

$$\sum_{a=0}^{p-1} a^{p-1-i}\sum_{b=0}^{p-1} b^{p-1-i} = 0\,.$$

We can rewrite it as follows

$$\sum_{a=0}^{p-1} a^{p-1-i} \sum_{b=0}^{p-1} b^{p-1-i}$$
$$= 2 \sum_{a=0}^{p-2} a^{p-1-i} \sum_{b=a+1}^{p-1} b^{p-1-i} + \sum_{a=0}^{p-1} a^{2(p-1-i)} .$$

This implies that

$$\sum_{a=0}^{p-2} a^{p-1-i} \sum_{b=a+1}^{p-1} b^{p-1-i} = -\frac{1}{2} \sum_{a=0}^{p-1} a^{2(p-1-i)} .$$

Note that the inverse of 2 is well defined modulo p since p is an odd prime. If $i < (p-1)/2$, then Lemma 4 says the sum on the right side is zero. Thus, the coefficient by $X^i Y^i$, which is exactly the sum on the left side, is equal to zero. If $i = (p-1)/2$, the above equality yields that the coefficient by $(XY)^{(p-1)/2}$ is equal to $-(p-1)/2$.

B Decomposition of $f(X, Y)$ for $3 \leq p \leq 7$

Let $Z = Y(X - Y)$. One non-scalar multiplication is needed to compute Z.

p=3.

$$f(X, Y) = 2 .$$

Since the polynomial $f(X, Y)$ is constant, it can be computed without any homomorphic multiplication.

p=5.

$$f(X, Y) = 4X^2 + 4X + Z .$$

Two non-scalar multiplications are needed to compute X^2 and Z.

p=7.

$$f(X, Y) = 1 + 4X(X + 1) + 6[X(X + 1)]^2 + (X^2 + 3X)Z + 6Z^2$$

In this case, four non-scalar multiplications are needed (indicated in bold) when rewritten as follows

$$f(X, Y) = 1 + 2(\boldsymbol{X}^2 + X) \cdot [2 + 3(X^2 + X)] + \boldsymbol{Z} \cdot [(X^2 + 3X) + 6Z] .$$

C Proof of Theorem 3

Let $Z = X - Y$. Thus we can rewrite $Q_{\mathtt{LTS}}(X, Y)$ as the univariate function $\chi_{\mathbb{F}_p^-}$, namely

$$Q_{\mathtt{LTS}}(X, Y) = \chi_{\mathbb{F}_p^-}(Z) = \sum_{a=-\frac{p-1}{2}}^{-1} 1 - (Z - a)^{p-1}.$$

Thanks to Lemma 3, we can expand $(Z-a)^{p-1}$ and obtain

$$\sum_{a=-\frac{p-1}{2}}^{-1} 1 - \sum_{i=0}^{p-1} Z^i a^{p-1-i} = \sum_{i=1}^{p-1} Z^i \sum_{a=-\frac{p-1}{2}}^{-1} (-a^{p-1-i}).$$

If i is even and $i < p-1$, then the ith coefficient is equal to

$$-\sum_{a=-\frac{p-1}{2}}^{-1} a^{p-1-i} = -\sum_{a=1}^{\frac{p-1}{2}} a^{p-1-i} = -\frac{1}{2} \sum_{a=-\frac{p-1}{2}}^{\frac{p-1}{2}} a^{p-1-i}.$$

This coefficient is equal to 0 for any even $0 < i < p-1$ thanks to Lemma 4. The $(p-1)$-th coefficient is equal to $-(p-1)/2 = (p+1)/2 \bmod p$. If i is odd, then we can rewrite the ith coefficient in the following way

$$-\sum_{a=-\frac{p-1}{2}}^{-1} a^{p-1-i} = \sum_{a=1}^{\frac{p-1}{2}} a^{p-1-i},$$

which finishes the proof.