Efficient Private Comparison Queries over Encrypted Databases using Fully Homomorphic Encryption with Finite Fields

Benjamin Hong Meng Tan[1], Hyung Tae Lee[2], Huaxiong Wang[3], Shu Qin Ren[4], and Khin Mi Mi Aung[1]

[1] Institute for Infocomm Research, A*STAR, Singapore
[2] Division of Computer Science and Engineering, College of Engineering, Chonbuk National University, Republic of Korea
[3] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
[4] Continental Automotive Singapore Pte Ltd, Singapore

Abstract. To achieve security and privacy for data stored on the cloud, we need the ability to secure data in compute. Equality comparisons, "$x = y, x \neq y$", have been widely studied with many proposals but there is much room for improvement for order comparisons, "$x < y$, $x \leq y$, $x > y$ and $x \geq y$". Most protocols for order comparisons have some limitation, either leaking some information about the data or requiring several rounds of communication between client and server. In addition, little work has been done on retrieving with compound conditions, mixing several equality and order comparisons. Fully homomorphic encryption (FHE) promises the ability to compute arbitrary functions on encrypted data without sacrificing privacy and without communication, but its potential has yet to be fulfilled. Particularly, private comparisons for database queries using FHE are expensive to compute.

In this work, we design efficient private database query (PDQ) protocols which support order comparisons and compound conditions. To this end, we first present a private comparison algorithm on encrypted integers using FHE, which scales efficiently for the length of input integers, by applying techniques from finite field theory. Then, we consider two scenarios for PDQ protocols, the first for retrieving data based on one order comparison and the second based on a conjunction of one order and four equality conditions. The proposed algorithm and protocols are implemented and tested to determine their performance in practice. The proposed comparison algorithm takes about 20.155 seconds to compare 697 pairs of 64-bit integers using Brakerski-Gentry-Vaikuntanathan's leveled FHE scheme with single instruction multiple data (SIMD) techniques at more than 110 bits of security. This yields an amortized rate of just 29 milliseconds per comparison. On top of that, we show that our techniques achieve an efficient PDQ protocol for one order and four equality comparisons, achieving an amortized time and communication cost of 36 milliseconds and 154 bytes per database element.

Keywords: Private database queries, fully homomorphic encryption, secure comparison

1 Introduction

Databases are used by many companies ranging from small shops to large banks for various purposes. With the wide availability of cloud services such as Amazon Relational Database Service [4], reliable and scalable databases can be deployed with ease and without the need to spend large sums to procure and maintain necessary infrastructure. On the other hand, this invites a new set of challenges in retaining data privacy while taking full advantage of the cloud revolution. Challenges include preventing data on the cloud from being stolen by hackers and employees of cloud service providers and reducing the exposure of data by protecting the queries that are performed on them.

Private database queries (PDQ) have been active areas, with many proposals for protocols intended for equality comparisons [8, 21, 30, 31, 36, 38, 44] "$x = y, x \neq y$", but the performance of those for order comparisons, "$x < y, x \leq y, x > y, x \geq y$", have a lot of room for improvement. A wide variety of techniques have been applied to this problem; interactive solutions include [17, 21, 36], while non-interactive solutions can be sub-divided into two types, order-preserving (OPE)/revealing (ORE) encryption [1, 6, 7, 9, 12, 29, 33, 37, 39, 43] and fully homomorphic encryption (FHE) [14, 15, 26, 34].

The security of the PDQ protocol depends heavily on the techniques used in its realization. There has been some research [18, 28, 32] showing that access pattern and even communication volume leakage can lead to serious attacks against private database queries. In particular, these are generic attacks against

PDQs that do not depend on any system details and only their leakage profile. Therefore, it is important to minimize leakage in the design of PDQ protocols.

Besides access pattern leakage concerns, a non-interactive protocol for private comparisons is most desirable, allowing users to go offline after they have provided their inputs and only returning when they wish to collect the result. This minimizes communication costs such as uptime and bandwidth for users. However, there are limitations to current non-interactive solutions as well. Those based on OPE/ORE leak information about the encrypted data in exchange for very fast comparisons. On the other hand, using FHE either takes some time [14, 15, 26] or suffers limitations to the bit-length of compared integers [34].

For private database queries, there are other metrics that are important as well: How much space is required to store and send the database and query results as well as the amount of time required to query the database. In these two metrics, existing FHE-based approaches fall short. The work by Cheon et al. [14,15] gave single instruction multiple data (SIMD)-optimized algorithms based on Boolean circuits but only considered low bit-length inputs. Although they showed good performance for small integers, their approach incurs a lot of space overhead and requires more bandwidth to transfer encrypted data around. For the protocol proposed by Lu et al. [34] and an extension by Ishimaki and Yamana [26], they only encrypt around 10-16 bits per ciphertext of about 60 KB. Lu et al. [34]'s protocol is quite fast, reporting a time of about 30 milliseconds per comparison but such performance is restricted to integers of at most 16 bits. Ishimaki and Yamana [26] extended the protocol to support larger integers but its performance falls short of the original protocol.

For private database queries to make sense in practice, we need to consider larger integers, e.g. 64 bits or more, and large databases sizes. Thus, there is still room for improvement; to construct an efficient private comparison algorithm that can scale to batches of large integers with less space overhead.

One common point amongst all secure comparison methods using FHE is that they did not consider using finite extension fields, which is the native space for the most widely used SIMD-capable FHE schemes by Brakerski et al. [11] and Brakerski [10] and Fan and Vercauteren [20]. The main obstacle to its adoption is the lack of work into their potential use for computation, with only several papers published to date. Kim et al. [30, 31] proposed efficient methods to perform equality comparisons with finite fields but their techniques do not translate to order comparisons. Aside from that, Jäschke and Armknecht [27] explored encoding integers to finite fields to compute additions but found that it was most effective to use the base fields instead of extension fields. However, their findings have limited use as they did not consider the entire structure of finite fields; the fact that finite fields are $\mathbb{F}_p$-algebras.

1.1 Our Contributions and Strategy

In this work, we propose a new order-preserving method of encoding integers to finite field elements, called the vector of field elements (VFE). Then, we construct algorithms for equality and order comparisons for integers encoded in VFE. This is achieved by generalizing the idea of comparing the individual digits of numbers to field elements. In particular, we construct a novel and efficient algorithm to compare two field elements based on the lexicographic order of its basis elements. Then, we show applications of VFE encoding and algorithms by designing two PDQ protcols: The first is a simple information retrieval protocol which returns the values of database records whose keys satisfy an order condition, i.e. $\mathsf{key} < \mathsf{Q}$ for some query constant Q. The second highlights the composability of our techniques, retrieving values ($v_{\boldsymbol{\rho}}$) in the database whose keys $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_t)$, satisfy a compound condition, e.g. $\rho_0 < \mathsf{Q}_0$ `and` $\rho_1 = \mathsf{Q}_1$ `and` $\ldots$ `and` $\rho_z = \mathsf{Q}_z$ for some $z < t - 1$. Finally, we provide some experiment results on the performance of the algorithms and protocols.

A generalization of the binary representation, VFE encoding decomposes an integer in $\mathbb{Z} \cap [0, p^{n \cdot d})$ to a length-n vector of digits in $\mathbb{Z} \cap [0, p^d)$, which are then encoded to $\mathbb{F}_p^d$. This latter encoding, called field element (FE) encoding, simply takes the base-p representation of the digits $x = \sum_{i=0}^{d-1} x_i p^i$ and maps it to the field element $\sum_{i=0}^{d-1} x_i t^i$, where t is the root of some degree-d polynomial irreducible modulo p.

From this encoding, we exploit depth-free automorphisms, rotations and shifts on plaintext slots to construct algorithms for equality and order comparisons on VFE-encoded encrypted integers. First, the structure of the Boolean circuits for equality and order are abstracted to VFE-encoded inputs,

$$\mathsf{EQ}_{\mathsf{VFE}}(x,y) = \prod_{i=0}^{n-1} \mathsf{EQ}_{\mathsf{FE}}(x_i, y_i), \tag{1}$$

$$\mathsf{LT}_{\mathsf{VFE}}(x,y) = \mathsf{LT}_{\mathsf{FE}}(x_{n-1}, y_{n-1}) + \sum_{i=0}^{n-2} \mathsf{LT}_{\mathsf{FE}}(x_i, y_i) \prod_{j=i+1}^{n-1} \mathsf{EQ}_{\mathsf{FE}}(x_j, y_j). \tag{2}$$

Equations (1) and (2) express the result of comparisons ($\mathsf{EQ}_{\mathsf{VFE}}, \mathsf{LT}_{\mathsf{VFE}}$) with VFE-encoded inputs as an arithmetic circuit with modules that compute equality and order ($\mathsf{EQ}_{\mathsf{FE}}, \mathsf{LT}_{\mathsf{FE}}$) of FE-encoded inputs. $\mathsf{EQ}_{\mathsf{FE}}$ was previously considered by Kim et al. [31] and we simply apply it to our framework to realize equality comparisons with VFE encoding.

Order comparisons with FE-encoded integers, on the other hand, have not been explored previously. To construct an efficient algorithm for FE-encoded integers, we start with a simple method that evaluates a bivariate polynomial representing the order comparison function. This polynomial can be derived using Lagrange interpolation and roughly has $\mathcal{O}(p^{2d})$ monomials. However, this approach does not scale well to fields with high extension degree, since the size of the polynomial is exponential in the degree.

For efficiency, we introduce a decomposition technique, based on similar ideas to VFE encoding. Exploiting the vector space structure of the extension field, we apply linear maps to reduce FE-encoded inputs to vectors of blocks, which are field elements in a smaller subspace $\mathcal{P}_r = \{\sum_{i=0}^{r-1} a_i t^i \mid a_i \in \mathbb{F}_p\}$, with $r = 1$ or 2 in practice. Field elements $\sum_{i=0}^{d} x_i t^i \in \mathbb{F}_{p^d}$ are mapped to the vector $(B_0^{(x)} = \sum_{i=0}^{r-1} x_i t^i, \ldots, B_{\ell-1}^{(x)} = \sum_{i=0}^{r-1} x_{(\ell-1)r+i} t^i) \in (\mathcal{P}_r)^\ell$, where $\ell = d/r$. Order and equality over these blocks are computed using a bivariate polynomial obtained by interpolating over $\mathcal{P}_r \times \mathcal{P}_r$ instead of $\mathbb{F}_{p^d} \times \mathbb{F}_{p^d}$. Equation (2) is then adapted to this setting, with blocks instead of FE-encoded digits, to complete the algorithm. With this, we reduce the number of monomials evaluated to $\mathcal{O}(d/r \cdot p^{2r})$ from $\mathcal{O}(p^{2d})$.

Applying this technique, we obtain an efficient order comparison algorithm for encrypted integers using VFE encoding and FHE with finite fields: When we use (p, d, n)-VFE encoding with blocks from $\mathcal{P}_r$, meaning integers are decomposed to base p^d and encoded as length-n $\mathbb{F}_{p^d}$-vectors, our comprehensive analysis shows that the proposed algorithm, $\mathsf{LT}_{\mathsf{VFE}}$, takes about $(2p-1)d + 2\lfloor \log n \rfloor + 2$ depth-free automorphisms, $(d/r)\big(3p^r - 5 + \log(d/r)\big) + \lfloor \log n \rfloor + 2$ multiplications and $\lfloor \log(p-1) \rfloor + \lfloor \log d \rfloor + \lfloor \log n \rfloor + 3$ depth.

To illustrate the efficiency of the proposed algorithm and protocols, we provide their implementation results for various choices of p, d, n and r. Based on our experiments using the HElib library by Halevi and Shoup [25], when $p = 5$, $d = 7$, $n = 4$ and $r = 1$, the proposed comparison algorithm performs best and takes about 20.155 seconds to compare 697 pairs of 64-bit integers, yielding an amortized rate of just 29 milliseconds for one comparison. On top of that, our techniques provide an efficient PDQ protocols for one order and four equality comparisons, achieving an amortized time and communication cost of 36 milliseconds and 154 bytes, respectively, per database element.

1.2 Related Work

The core of PDQs for comparison conditions is a private comparison protocol or algorithm. This can be realised with encryption schemes such as order-preserving/revealing (OPE/ORE) encryption and homomorphic encryption. OPE was introduced by Agrawal et al. [1] and is a type of encryption scheme that preserves the order of plaintexts after encryption. Following their work, many schemes were proposed [6, 7, 9, 12, 29, 33, 37, 39, 43] improving the performance and security of such schemes. ORE was proposed by Boneh et al. [9] as a generalization of OPE that uses an additional algorithm to reveal the order of encrypted plaintexts instead. The initial proposal, which would only reveal the order of encrypted plaintexts and nothing else, requires multilinear maps which are still impractical. However, more practical schemes [12, 13, 33] have been achieved at the expense of more leakage.

Unfortunately, OPE schemes, when applied to database queries, have been shown to be vulnerable to inference attacks by Naveed et al. [35] while the schemes mentioned above, except for [33, 39], were attacked by Durak et al. [19]. In fact, the premise for OPE and ORE has been shown to be a potential problem due to generic reconstruction attacks by Kellaris et al. [28] and Lacharité et al. [32].

Interactive comparison protocols using homomorphic encryption include Damgård et al. [17] and Gentry et al. [21]. They use interaction to achieve fast comparisons but it is unclear how they can be extended to support compound conditions. The direct method will be to run several protocols, one for each condition, and finally retrieve the results of those that satisfy the compound condition. This requires rounds of communication proportional to the number of conditions and can reveal partial information about the query if not handled properly.

FHE has also been applied to obtain non-interactive solutions. Cheon et al. [14, 15], Lu et al. [34] achieved reasonably fast comparisons on small integers of around 16 bits but suffer from large communication overhead as they do not pack many bits in a single ciphertext. Ishimaki and Yamana [26] extended the Lu et al. [34] protocol to handle larger integers but has lower performance than the original protocol even at small integers.

1.3 Outline

In the next section, some background and notations that are used throughout the paper are presented. After that, in Section 3, we introduce the vector of field elements (VFE) encoding technique and VFE-based circuits for comparisons. Following that, we design an algorithm to determine the order of elements in $\mathbb{F}_{p^d}$ in Section 4 to complement the equality comparison algorithm over $\mathbb{F}_{p^d}$. Next, in Section 5, we showcase applications of the techniques developed earlier with two efficient private database queries that enjoy low communication complexity. In Section 6, we evaluate the performance of the proposed protocols. Lastly, we provide concluding remarks in the last section.

2 Preliminaries

Notation. Throughout the paper, the set of integers from 1 to a is denoted by $[a]$ and $\lfloor a \rfloor$ (resp. $\lceil a \rceil$) denotes the largest (resp. smallest) integer that is smaller (resp. larger) than or equal to a real number a. In our work, we use finite extension fields $\mathbb{F}_{p^d}$ of characteristic p and extension degree d, $\mathbb{F}_{p^d} = \{(\sum_{i=0}^{d-1} \gamma_i t^i) \bmod g(x) \mid \gamma_i \in \mathbb{F}_p \text{ and } t \text{ is the root of an irreducible polynomial } g(x) \in \mathbb{F}_p[x]\}$. We consider the set $\{1, t, \dots, t^{d-1}\}$ as a basis of $\mathbb{F}_{p^d}$, when it is viewed as an $\mathbb{F}_p$-vector space. Logarithms in base-2 are denoted with log while those in base-w with $w \neq 2$ specified with a subscript, $\log_w$.

The security parameter is denoted by λ and for simplicity, it is assumed that all algorithms take it as an input. A function $\varepsilon : \mathbb{N} \to \mathbb{R}$ is negligible in λ, if for all positive polynomials $p(\cdot)$ and sufficiently large λ, $\varepsilon(\lambda) \leq \frac{1}{p(\lambda)}$. We use $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ to represent unspecified polynomials and negligible functions in λ respectively. A probabilistic polynomial-time (PPT) algorithm is a randomised algorithm that runs in time $\mathsf{poly}(\lambda)$.

2.1 Comparisons

In this work, we consider six comparisons, over some totally ordered set $(A, <_A)$. These are then divided into two categories, equality and order comparisons. Let $x, y \in A$. There are two equality comparisons, formally defined as follows.

$$\mathsf{EQ}_A(x,y) = \begin{cases} 1, & \text{if } x = y; \\ 0, & \text{otherwise,} \end{cases} \qquad \mathsf{NEQ}_A(x,y) = \begin{cases} 1, & \text{if } \neg(x = y); \\ 0, & \text{otherwise.} \end{cases}$$

The other four are order comparisons given, below as

$$\mathsf{LT}_A(x,y) = \begin{cases} 1, & \text{if } x <_A y; \\ 0, & \text{otherwise,} \end{cases} \qquad \mathsf{GT}_A(x,y) = \begin{cases} 1, & \text{if } y <_A x; \\ 0, & \text{otherwise,} \end{cases}$$

$$\mathsf{LEQ}_A(x,y) = \begin{cases} 1, & \text{if } \neg(y <_A x); \\ 0, & \text{otherwise,} \end{cases} \qquad \mathsf{GEQ}_A(x,y) = \begin{cases} 1, & \text{if } \neg(x <_A y); \\ 0, & \text{otherwise.} \end{cases}$$

Unless confusion arises, we omit the set A in order notations. In practice, only two of the six functions, LT and EQ, need to be considered. This is because GT can be derived by swapping the inputs to LT and the other three functions, NEQ, GEQ and LEQ, are complements of EQ, LT and GT respectively. We use the formulas $\mathsf{NEQ}(x, y) = 1 - \mathsf{EQ}(x, y)$, $\mathsf{GEQ}(x, y) = 1 - \mathsf{LT}(x, y)$ and $\mathsf{LEQ}(x, y) = 1 - \mathsf{GT}(x, y)$ to compute them.

Generalized Order Comparison. In all, we can define a general function which specifies the particular order comparison to be done with two additional bits of information, α, β. In Equation (3), α is a bit which decides the order which the inputs x and y are given to the LT function and β is a mask which inverts the boolean value of LT if set to 1.

$$\mathsf{Ord}(x, y, \alpha, \beta) = \beta - \mathsf{LT}(x + \alpha(y - x), y + \alpha(x - y)) = \begin{cases} \mathsf{LT}(x, y), & \text{if } \alpha = 0, \beta = 0; \\ \mathsf{GEQ}(x, y), & \text{if } \alpha = 0, \beta = 1; \\ \mathsf{GT}(x, y), & \text{if } \alpha = 1, \beta = 0; \\ \mathsf{LEQ}(x, y), & \text{if } \alpha = 1, \beta = 1. \end{cases} \tag{3}$$

2.2 Finite Extension Fields

This subsection establishes several important lemmas for working in plaintext spaces other than $\mathbb{F}_2$. The proofs are omitted due to space constraints but interested readers are referred to Section 3.2.2 of [31].

Definition 1 ([31, Definition 1]). *Let p be a prime and d and n be positive integers. A polynomial $f \in \mathbb{F}_{p^d}[x_1, x_2, \ldots, x_n]$ is a polynomial expression of a function $\varphi : (\mathbb{F}_{p^d})^n \to \mathbb{F}_{p^d}$ if $f(a_1, a_2, \ldots, a_n) = \varphi(a_1, a_2, \ldots, a_n)$ for all $(a_1, a_2, \ldots, a_n) \in (\mathbb{F}_{p^d})^n$.*

With Definition 1, we are able to relate functions to polynomials over $\mathbb{F}_{p^d}$. The following lemma guarantees that there is a unique polynomial expression with a particular property, min_φ, for a function $\varphi : (\mathbb{F}_{p^d})^n \to \mathbb{F}_{p^d}$.

Lemma 1 ([31, Lemma 1]). *Let p be a prime and d and n be positive integers. For any function $\varphi : (\mathbb{F}_{p^d})^n \to \mathbb{F}_{p^d}$, there exists a unique polynomial expression $min_\varphi \in \mathbb{F}_{p^d}[x_1, x_2, \ldots, x_n]$ of φ whose degree is at most $p^d - 1$ with respect to each variable.*

This unique polynomial expression min_φ is called the minimal polynomial expression of a function φ and it has a property that is very important for designing efficient algorithms to evaluate φ on encrypted data.

Lemma 2 ([31, Lemma 2]). *For a function $\varphi : (\mathbb{F}_{p^d})^n \to \mathbb{F}_{p^d}$ with n, p and d as in Lemma 1, the minimal polynomial expression of φ has the minimum total degree among all polynomial expressions of φ.*

The total degree of a polynomial determines the multiplicative depth required for evaluating that polynomial. Thus, Lemma 2 implies that we can evaluate a function φ with minimum multiplicative depth by evaluating min_φ using general FHE schemes.

After establishing a connection between a function and polynomial expressions, the next step is to find the minimal polynomial expression. We will use Lagrange interpolation to achieve this. Below, we provide a theorem of Lagrange interpolation for 2-variable functions defined on $\mathcal{P} \times \mathcal{P} \subseteq (\mathbb{F}_{p^d})^2$ for some $\mathcal{P} \subseteq \mathbb{F}_{p^d}$. Note that this can be extended to n-variable functions for arbitrary $n \geq 2$.

Theorem 1. *(Lagrange Interpolation) Given the outputs of a function φ on all points $(x, y) \in \mathcal{P} \times \mathcal{P} \subseteq \mathbb{F}_{p^d}^2$, a polynomial expression $f(x, y)$ of φ can be constructed as*

$$f(x, y) = \sum_{(x_i, y_j) \in \mathcal{P} \times \mathcal{P}} \varphi(x_i, y_j) \Big(\prod_{\substack{x_\alpha \neq x_i \\ x_\alpha \in \mathcal{P}}} \frac{x - x_\alpha}{x_i - x_\alpha} \Big) \Big(\prod_{\substack{y_\beta \neq y_j \\ y_\beta \in \mathcal{P}}} \frac{y - y_\beta}{y_j - y_\beta} \Big).$$

Then, $f(x, y)$ is the polynomial that evaluates to $\varphi(x^*, y^*)$ for any $(x^*, y^*) \in \mathcal{P} \times \mathcal{P}$. The degrees of x and y in f are at most $|\mathcal{P}| - 1$ each and so the total degree of f is at most $2|\mathcal{P}| - 2$.

The following corollary states that the polynomial $f(x, y)$ specified in Theorem 1 is the minimal polynomial expression of $\varphi(x, y)$ by Lemma 1.

Corollary 1. *The polynomial $f(x, y)$ as specified by Theorem 1 is the minimal polynomial expression of $\varphi(x, y)$, if $\mathcal{P} = \mathbb{F}_{p^d}$.*

Proof. Since the degrees of x and y in $f(x, y)$ are both at most $p^d - 1$, it is straightforward by Lemma 1.

Here, we introduce an important property of finite extension fields that will be used in the next section: linear maps on $(\mathbb{F}_p)^d$. See Theorem 10.4.4. in [40] for a proof of Lemma 3.

Lemma 3. *Let T be a $\mathbb{F}_p$-linear map on $\mathbb{F}_{p^d}$ for a prime p and a positive integer d. Denote by $\tau(x)$ the Frobenius map on $\mathbb{F}_{p^d}$ which sends x to x^p. There is a unique set of constants $\{\rho_0, \rho_1, ..., \rho_{d-1}\}, \rho_i \in \mathbb{F}_{p^d}$ for $i \in \{0, \dots, d-1\}$ such that*

$$T(\mu) = \zeta_T(\mu) = \sum_{i=0}^{d-1} \rho_i \tau^i(\mu).$$

Lastly, we state a simple theorem characterising the multiplicative group of $\mathbb{F}_{p^d}$, which has applications that we discuss in Section 3.2.

Theorem 2 (Fermat's Little theorem on $\mathbb{F}_{p^d}$). *Let p be a prime and $d > 0$. Then, for any $\alpha \in \mathbb{F}_{p^d} \backslash \{0\}$, we have*

$$\alpha^{p^d - 1} = 1. \tag{4}$$

2.3 Fully Homomorphic Encryption

Let $\overline{m}$ denote a ciphertext that encrypts the plaintext m. A leveled fully homomorphic encryption (FHE) scheme[5] is a 4-tuple of probabilistic polynomial time algorithms (KeyGen, Enc, Dec, Eval) as follows,

- $(pk, evk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, L)$: Takes as inputs security parameter λ and maximum depth L and outputs a public key pk, evaluation key evk and secret key sk.
- $c = \overline{m} \leftarrow \mathsf{Enc}(pk, m)$: Takes as inputs public key pk and plaintext $m \in \mathcal{P}$ for plaintext space $\mathcal{P}$, outputs a ciphertext c which is an encryption of m.
- $m' \leftarrow \mathsf{Dec}(sk, c)$: Takes as inputs secret key sk and ciphertext c and outputs a plaintext m'.
- $c' \leftarrow \mathsf{Eval}(evk, \varphi, \overline{m_1}, \overline{m_2}, \dots, \overline{m_n})$: Takes as inputs evaluation key evk, n-variate polynomial expression φ of total degree at most 2^L and n ciphertexts $\overline{m_1}, \dots, \overline{m_n}$ and outputs a ciphertext c' such that $c' = \overline{\varphi(m_1, \dots, m_n)}$.

Usually, Eval is performed with two sub-algorithms $\mathsf{EvalAdd}$ and $\mathsf{EvalMult}$ which correspond to addition and multiplication of encrypted plaintexts respectively.

- $c^+ \leftarrow \mathsf{EvalAdd}(evk, \overline{m_1}, \overline{m_2})$: Takes as inputs evaluation key evk and two ciphertexts $\overline{m_1}, \overline{m_2}$ and outputs a ciphertext c^+ such that $c^+ = \overline{m_1 + m_2}$.
- $c^\times \leftarrow \mathsf{EvalMult}(evk, \overline{m_1}, \overline{m_2})$: Takes as inputs evaluation key evk and two ciphertexts $\overline{m_1}, \overline{m_2}$ and outputs a ciphertext $c^\times$ such that $c^\times = \overline{m_1 \times m_2}$.

Multiplicative Depth and Complexity. For n-variable polynomials, $f(x_1, \dots, x_n)$, we consider multipliations between monomials in the inputs as *multiplications* and those between coefficients of f and monomials in the inputs as *constant multiplications*. Then, we can view the evaluation of f as an arithmetic circuits with addition and multiplication gates corresponding to the operations on the plaintext space. Constant multiplications are simply scalings of the ciphertext by the constants. Our measure of multiplicative complexity of f will be the number of multiplications required to evaluate f, because it is the most expensive operation in FHE.

However, there is another important metric of complexity in FHE, multiplicative depth. This is defined as the maximum number of multiplications along any path from the inputs, $x_1, \dots, x_n$ to the outputs $f(x_1, \dots, x_n)$. For leveled FHE schemes, the parameter L, called the level, is used during setup to specify maximal multiplicative depth of arithmetic circuits that can be evaluated by the scheme.

[5] A leveled fully homomorphic encryption scheme is a homomorphic encryption scheme which supports L-depth circuits, where L is a parameter of the scheme.

Batching and Automorphisms on the Plaintext Space. Smart and Vercauteren [42] showed that some FHE schemes can support SIMD operations, also known as batching, through suitable parameter selection and the use of Chinese Remainder Theorem on number fields. In particular, the Brakerski-Gentry-Vaikuntanathan (BGV) [11], Brakerski (B) [10] and Fan-Vercauteren (FV) [20] schemes are compatible with these techniques.

When the plaintext modulus p is prime and not divisible by m, the cyclotomic polynomial modulus $\Phi_m(x)$ decomposes into ℓ irreducible factors $f_1(x), \ldots, f_\ell(x)$ of degree d modulo p. This induces an isomorphism, via the Chinese Remainder Theorem, between the algebra of the plaintext space $\mathbb{A}_p = \mathbb{F}_p[x]/\langle\Phi_m(x)\rangle$ and the product of ℓ finite fields $\mathbb{L}_i \cong \mathbb{F}_{p^d}$ for $i \in \{1, \ldots, \ell\}$,

$$\begin{aligned} \mathbb{A}_p &\cong \mathbb{F}_p[x]/\langle f_1(X)\rangle \times \cdots \times \mathbb{F}_p[x]/\langle f_\ell(x)\rangle \\ &= \mathbb{L}_1 \times \cdots \times \mathbb{L}_\ell \cong (\mathbb{F}_{p^d})^\ell. \end{aligned} \tag{5}$$

With this decomposition, plaintexts of compatible FHE schemes can be regarded as length-ℓ vectors $\mathbf{m} = (m_1, \ldots, m_\ell)$, where $m_i \in \mathbb{F}_{p^d}$ for $i \in \{1, \ldots, \ell\}$. Addition and multiplication on ciphertexts $\overline{\mathbf{m_1}}, \overline{\mathbf{m_2}}$ correspond to component-wise addition and multiplication over their encrypted plaintexts $m_{1,i}, m_{2,i}$ for $i \in \{1, \ldots, \ell\}$. Gentry et al. [22] described how to apply automorphisms on the plaintext space $\mathbb{A}_p$ without adding depth to the circuits for the BGV FHE scheme, adding new sub-algorithms for Eval, with an augmented evaluation key evk. These automorphisms include rotation and shift of the entries of the plaintext vector and component-wise Frobenius map evaluations:

- $c^\star \leftarrow \mathsf{EvalShift}(evk, \overline{\mathbf{m}}, i)$: Takes as inputs evaluation key evk, encrypted vector $\overline{\mathbf{m}} = \overline{(m_1, \ldots, m_\ell)}$ and index i and outputs a ciphertext $c^\star$ such that $c^\star = \overline{(m_i, \ldots, m_\ell, 0, \ldots, 0)}$.
- $c^\star \leftarrow \mathsf{EvalRotate}(evk, \overline{\mathbf{m}}, i)$: Takes as inputs evaluation key evk, encrypted vector $\overline{\mathbf{m}} = \overline{(m_1, \ldots, m_\ell)}$ and index i and outputs a ciphertext $c^\star$ such that $c^\star = \overline{(m_i, \ldots, m_\ell, m_1, \ldots, m_{i-1})}$.
- $c^\star \leftarrow \mathsf{EvalFrobenius}(evk, \overline{\mathbf{m}}, i)$: Takes as inputs evaluation key evk, encrypted vector $\overline{\mathbf{m}} = \overline{(m_1, \ldots, m_\ell)}$ and exponent i and outputs a ciphertext $c^\star$ such that $c^\star = \overline{(m_1^{p^i}, \ldots, m_\ell^{p^i})}$.

For $\mathsf{EvalShift}$ and $\mathsf{EvalRotate}$, using a negative index $-i$ corresponds to shifting or rotating to the right instead.

Furthermore, there is a software library for homomorphic encryption, HElib [25], by Halevi and Shoup that implements the necessary algorithms to fully utilize the plaintext space with BGV as the base FHE scheme.

Parameters vs. Performance. For leveled FHE schemes, the parameters for maximum depth L and security level λ both affect performance. Supporting deeper circuits with bigger L means larger parameter sizes, thereby increasing the time required to evaluate circuits and ciphertext sizes. Thus, it is important for algorithms to have minimal depth for optimal FHE performance. The impact of the parameter sizes on performance can be seen in the results we present in Section 6.

2.4 Batched Boolean Circuit Approach for Comparisons

Cheon et al. [14, 15] described how to apply the SIMD techniques by Smart and Vercauteren [42] to efficiently evaluate boolean circuits for equality, comparison and full adders for n-digit inputs. In particular, they used the plaintext modulus $p = 2$ which yields the plaintext space $\mathbb{L}_1 \times \cdots \times \mathbb{L}_\ell$, where each $\mathbb{L}_i \cong \mathbb{F}_{2^d}$ for some $d \geq 1$. However, they did not use all of $\mathbb{F}_{2^d}$, restricting themselves to the base field $\mathbb{F}_2$ where addition corresponds to the exclusive-or (XOR) gate and multiplication to the AND gate.

For any n that divides ℓ, we can fit up to ℓ/n-many n-bit integers into a single ciphertexts and exploit SIMD functionality to simultaneously evaluate comparisons on multiple integers. Let $x = \sum_{i=0}^{n-1} x_i 2^i$ and $y = \sum_{i=0}^{n-1} y_i 2^i$ be two n-digit integers, then we have

$$\mathsf{EQ}(x, y) = \prod_{i=0}^{n-1} (1 - (x_i - y_i)), \tag{6}$$

$$\mathsf{LT}(x, y) = x_{n-1}(1 - y_{n-1}) + \sum_{i=0}^{n-2} x_i(1 - y_i) \prod_{j=i+1}^{n-1} (1 - (x_i - y_i)). \tag{7}$$

SIMD-Optimized Evaluation of Running Products. With all the bits of the encoded integer encrypted in a single ciphertext, there are some optimizations that can be done to reduce the number of multiplications needed to evaluate running products, i.e. vectors of the form $\boldsymbol{\pi}_{\mathbf{x}} = (x_0, x_0x_1, \ldots, \prod_{i=0}^{n-1} x_i)$ for $\mathbf{x} = (x_0, \ldots, x_{n-1})$. The most efficient way, proposed by Cheon et al. [14,15] is to do $\log n$-many shift-multiply steps as illustrated below.

Let $*$ denote component-wise multiplication of vectors and

$$\mathbf{x}^{(i)} = (x_0^{(i)}, \ldots,\ x_i^{(i)}, \ldots, x_{n-1}^{(i)}) = (\underbrace{1, \ldots, 1}_{i \text{ copies}},\ x_0, \ldots, x_{n-1-i}) \tag{8}$$

for $\mathbf{x} = (x_0, \ldots, x_{n-1})$ and define the following sub-routine

$$\mathsf{shift\_mul}(\mathbf{x}, i) = \mathbf{x} * \mathbf{x}^{(i)}.$$

$\boldsymbol{\pi}_{\mathbf{x}}$ is obtained by iteratively applying $\mathsf{shift\_mul}$ as follows,

1. set $\mathbf{z}_0 = \mathbf{x}$;
2. for $0 \le i \le \lfloor \log n \rfloor$, compute $\mathbf{z}_{i+1} = \mathsf{shift\_mul}(\mathbf{z}_i, 2^i)$;
3. set $\boldsymbol{\pi}_{\mathbf{x}} = \mathbf{z}_{\lfloor \log n \rfloor + 1}$.

We prove the correctness of this via induction. For the base case, where $i = 0$, we have $\mathbf{z}_0 = \mathbf{x}$ and the first $2^0 = 1$ entries of $\mathbf{z}_0$ store the correct running products. $\mathbf{z}_1 = \mathbf{x} * \mathbf{x}^{(1)} = (x_0, x_1^{(1)} \cdot x_1, \ldots, x_{n-1}^{(1)} \cdot x_{n-1})$ and since $x_j^{(i)} = x_{j-i}$, $\mathbf{z}_1 = (x_0, x_0x_1, \ldots, x_{n-2}x_{n-1})$ and we have the correct running products in the first $2^1 = 2$ entries of $\mathbf{z}_1$, with the j-th entry containing x_j and the term preceeding it x_{j-1} for $j \ge 2^1$.

Now, suppose that at the i-th step, $\mathbf{z}_i$ has j-th entry $\prod_{k=0}^{j} x_k$ for $j < 2^i$ and $\prod_{k=0}^{2^i-1} x_{j-k}$ for $2^i \le j \le n-1$; we have $\mathbf{z}_i = (\underbrace{x_0, x_0x_1, \ldots, \prod_{j=0}^{2^i-1} x_j}_{2^i \text{ terms}}, \prod_{j=0}^{2^i-1} x_{2^i+j}, \ldots, \prod_{j=0}^{2^i-1} x_{n-2^i+j})$, i.e. correct running products in the first 2^i entries and the product of x_j with the $2^i - 1$ entries before it for all other $2^i \le j \le n-1$. Then, $\mathbf{z}_i^{(2^i)} = (\underbrace{1,\ \ldots,\ 1}_{2^i \text{ terms}},\ \underbrace{x_0, \ldots,\ \prod_{j=0}^{2^i-1} x_j}_{2^i \text{ terms}},\ \prod_{j=0}^{2^i-1} x_{2^i+j},\ \ldots,\ \prod_{j=0}^{2^i-1} x_{n-(2^{i+1})+j})$ and $\mathbf{z}_{i+1} = \mathbf{z}_i * \mathbf{z}_i^{(2^i)}$ yields the correct running products $\prod_{k=0}^{j} x_k$ for $j < 2^{i+1}$ and $\prod_{k=0}^{2^{i+1}-1} x_{j-k}$ for $2^{i+1} \le j \le n-1$. Therefore, all $\mathbf{z}_i$ for $i \ge 0$ have the correct form and after the $\lfloor \log n \rfloor + 1$ steps, $\mathbf{z}_{\lfloor \log n \rfloor + 1}$ will have the correct set of running products. Further $\mathsf{shift\_mul}$ of $\mathbf{z}_{\lfloor \log n \rfloor + 1}$ will only multiply it with the $(1, \ldots, 1)$ vector and remain unchanged.

Note that $\mathbf{x}^{(i)}$ is simply computed by shifting the vector to the right by i, yielding $(0, \ldots, 0, x_0, \ldots, x_{n-i-1})$, and adding the vector $(1, \ldots, 1, 0, \ldots, 0)$, where the first i entries are 1's and the rest 0's.

Lastly, we describe an important procedure, $\mathsf{replicate\_first}(\mathbf{x})$ for $\mathbf{x} = (x_1, \ldots, x_n)$, that produces a vector $\mathbf{x}' = (\underbrace{x_1, \ldots, x_1}_{n \text{ times}})$. This is achieved by first multiplying the mask $(1, 0, \ldots, 0)$ to $\mathbf{x}$ to obtain $\mathbf{x}^\dagger = (x_1, 0, \ldots, 0)$ and then iteratively applying shift and adds to $\mathbf{x}^\dagger$.

3 Comparisons with Vectors of Field Elements

First, we introduce a generalization of encoding of integers as binary vectors, which are vectors whose elements lie in $\mathbb{F}_2$. This technique, called vector of field elements (VFE) encoding, represents integers as a $\mathbb{F}_{p^d}$-vector. Then, we show how arithmetic circuits in $\mathbb{F}_{p^d}$ can be exploited to compute comparisons over VFE-encoded integers in three phases:

1. We formulate VFE-based circuits for EQ, LT from component functions over $(\mathbb{F}_{p^d})^2$;
2. We derive a VFE-based equality algorithm by exploiting an algorithm for $\mathsf{EQ}_{\mathsf{FE}}$ from Kim et al. [31];
3. We design an efficient algorithm for $\mathsf{LT}_{\mathsf{FE}}$ in Section 4.

These algorithms can then be adapted for use on encrypted data by replacing each gate with their counterparts in the FHE scheme, $\mathsf{EvalAdd}, \mathsf{EvalMult}, \mathsf{EvalShift}$, $\mathsf{EvalRotate}$ and $\mathsf{EvalFrobenius}$.

3.1 The Vector of Field Elements Encoding

To exploit the complete plaintext space that available in Smart-Vercautren SIMD compatible FHE schemes, we propose a new encoding, called *vector of field elements* (VFE). Recall from Equation (5) that plaintexts in such FHE schemes correspond to vectors in $(\mathbb{F}_{p^d})^\ell$. Unlike in the Boolean circuit case, where $p = 2$ and $d = 1$, there is no restriction on p except that it is prime.

For any integer $a \in \mathbb{Z} \cap [0, p^{n\cdot d})$, we express a in base-p, i.e. $a = \sum_{i=0}^{n-1} \sum_{j=0}^{d} a_{i,j} p^{i\cdot d+j}$ and group coefficients into sub-vectors of length d. In essence, we encode a as a vector $\mathbf{a} = (\mathbf{a}_0, \ldots, \mathbf{a}_{n-1})$, where each $\mathbf{a}_i = (a_{i,0}, \ldots, a_{i,d-1})$ with $a_{i,j} \in \mathbb{F}_p$ for $i \in \{0, \ldots, n-1\}$ and $j \in \{0, \ldots, d-1\}$. Each $\mathbf{a}_i$ is then encoded as elements in $\mathbb{F}_{p^d}$. For simplicity, we use the power basis $\{1, t, \ldots, t^{d-1}\}$ and denote the encoding of $\mathbf{a}_i$ by the function $\mathsf{FE.enc}$, where

$$\begin{aligned} \mathsf{FE.enc}:\ (\mathbb{F}_p)^d &\longrightarrow \mathbb{F}_{p^d} \\ \boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_{d-1}) &\longmapsto \sum_{j=0}^{d-1} \alpha_j t^j. \end{aligned}$$

Then, the VFE encoding of an integer a, denoted by $\mathsf{VFE.enc}$, is

$$\begin{aligned} \mathsf{VFE.enc}:\ \mathbb{Z} \cap [0, p^{n\cdot d}) &\longrightarrow (\mathbb{F}_{p^d})^n \\ a &\longmapsto \mathbf{a} = \big(\mathsf{FE.enc}(\mathbf{a}_0), \ldots, \mathsf{FE.enc}(\mathbf{a}_{n-1})\big). \end{aligned}$$

We note here that the order that is inherent in the integers is preserved in the FE and VFE encoding, as in the case of binary decomposition. First, it is evident that for integers in $\{0, 1, \ldots, p^d - 1\}$, their order over the integers correspond to the lexicographic ordering of elements in $\mathbb{F}_{p^d}$ where $1 <_{\mathbb{F}_{p^d}} t <_{\mathbb{F}_{p^d}} \ldots <_{\mathbb{F}_{p^d}} t^{d-1}$. This then extends to integers in $\{0, 1, \ldots, p^{n\cdot d} - 1\}$ with the ordering over $(\mathbb{F}_{p^d})^n$, where $(1, 0, \ldots, 0) <_{(\mathbb{F}_{p^d})^n} (0, 1, \ldots, 0) <_{(\mathbb{F}_{p^d})^n} \ldots <_{(\mathbb{F}_{p^d})^n} (0, \ldots, 0, 1)$.

3.2 VFE-Based Comparison Functions

As mentioned in Section 2.1, only two comparison functions, EQ and LT are needed to obtain all six possibilities. To obtain the VFE-based comparison functions, we first observe that the Boolean circuits of Equations (6) and (7) can be abstracted as follows.

For $x, y \in \mathbb{Z} \cap [0, 2^n)$ such that $x = \sum_{i=0}^{n-1} x_i 2^i$ and $y = \sum_{i=0}^{n-1} y_i 2^i$,

$$\mathsf{EQ}(x, y) = \prod_{i=0}^{n-1} \mathsf{EQ}_{\mathbb{F}_2}(x_i, y_i), \tag{9}$$

$$\mathsf{LT}(x, y) = \underbrace{\mathsf{LT}_{\mathbb{F}_2}(x_{n-1}, y_{n-1})}_{\text{Case (a)}} + \underbrace{\sum_{i=0}^{n-2} \mathsf{LT}_{\mathbb{F}_2}(x_i, y_i) \prod_{j=i+1}^{n-1} \mathsf{EQ}_{\mathbb{F}_2}(x_j, y_j)}_{\text{Case (b)}}, \tag{10}$$

where $\mathsf{EQ}_{\mathbb{F}_2}(u, v) = 1 - (u - v)$ and $\mathsf{LT}_{\mathbb{F}_2}(u, v) = (1 - u)v$ for $u, v \in \mathbb{F}_2$.

The logic expressed in Equation (9) is that x and y are equal only if all their bits are equal. For Equation (10), we can break down the cases where $x <_{\mathbb{Z}} y$ according to the digit pairs, (x_i, y_i):

(a) For the most significant bit, i.e. $i = n - 1$, $(x_{n-1} <_{\mathbb{F}_2} y_{n-1}) \implies (x <_{\mathbb{Z}} y)$.

(b) Otherwise, for all other bits $i < n - 1$, if $x_j = y_j$ for all $i + 1 \le j < n$, then $(x_i <_{\mathbb{F}_2} y_i) \implies (x <_{\mathbb{Z}} y)$.

Notice that all possible cases are pair-wise disjoint, which allows us to simply sum the results of each to derive Equation (10). From Equations (9) and (10), we generalize them to the following EQ and LT functions for VFE-encoded integers $x, y \in \mathbb{Z} \cap [0, p^{n\cdot d})$ with $\mathbb{F}_{p^d}$.

$$\mathsf{EQ}_{\mathsf{VFE}}(x, y) = \prod_{i=0}^{n-1} \mathsf{EQ}_{\mathsf{FE}}(x_i, y_i), \tag{11}$$

$$\mathsf{LT}_{\mathsf{VFE}}(x, y) = \mathsf{LT}_{\mathsf{FE}}(x_{n-1}, y_{n-1}) + \sum_{i=0}^{n-2} \mathsf{LT}_{\mathsf{FE}}(x_i, y_i) \prod_{j=i+1}^{n-1} \mathsf{EQ}_{\mathsf{FE}}(x_j, y_j), \tag{12}$$

where $\mathsf{VFE.enc}(x) = (x_0, \dots, x_{n-1})$ and $\mathsf{VFE.enc}(y) = (y_0, \dots, y_{n-1})$. Note that when evaluated with SIMD techniques, the result would be of the form $(\mathsf{EQ/LT}(x, y), \alpha_1, \dots, \alpha_{n-1})$ where α_i are some irrelevant interemediate results.

Correctness. Equation (11) captures the idea that two integers are equal if every component of their VFE encoding are equal. This is straightforward to verify since equality between $x_i = \sum_{j=0}^{d-1} x_{i,j} t^j$ and $y_i = \sum_{j=0}^{d-1} y_{i,j} t^j$ implies that their pre-image under $\mathsf{FE.enc}$, $(x_{i,0}, \dots, x_{i,d-1})$ and $(y_{i,0}, \dots, y_{i,d-1})$ are equal. Extending this to every component of $\mathsf{VFE.enc}(x)$ and $\mathsf{VFE.enc}(y)$ yields the desired conclusion that $x = y$ over the integers.

As for Equation (12), its correctness follows in a similar manner as the correctness of Equation (10). The key is that the order of the integers between 0 and $p^{n \cdot d} - 1$ induces a corresponding ordering of elements in the range of $\mathsf{VFE.enc}$, $(\mathbb{F}_{p^d})^n$. Then, we have the following possibilities for $x < y$:

(a) For the most significant entry, i.e. $i = n - 1$, $(x_{n-1} <_{\mathbb{F}_{p^d}} y_{n-1}) \implies (x <_{\mathbb{Z}} y)$.

(b) Otherwise, for all other entries $i < n-1$, if $x_j = y_j$ for all $i+1 \leq j < n$, then $(x_i <_{\mathbb{F}_{p^d}} y_i) \implies (x <_{\mathbb{Z}} y)$.

With this framework, we reduce the problem of comparing VFE-encoded elements to comparing FE-encoded elements. Equality comparisons for FE-encoding is straightforward since it simply requires checking if two field elements are equal. Order comparisons are more involved and we defer discussion on it to Section 4.

Complexity Analysis. We apply the SIMD techniques of Section 2.3 to optimize the evaluation of $\mathsf{EQ}_{\mathsf{VFE}}, \mathsf{LT}_{\mathsf{VFE}}$. VFE-encoded inputs are encrypted in the slots of a single ciphertext, i.e. there is some ciphertext c_β such that $c_\beta = \overline{(\beta_{n-1}, \dots, \beta_0)}$, where $\mathsf{VFE.enc}(\beta) = (\beta_0, \dots, \beta_{n-1})$ for $\beta \in \{x, y\}$. From this, when we evaluate a single function h on ciphertexts $\overline{(x_{n-1}, \dots, x_0)}$ and $\overline{(y_{n-1}, \dots, y_0)}$, we will obtain $\overline{(h(x_{n-1}, y_{n-1}), \dots, h(x_0, y_0))}$ for $h \in \{\mathsf{EQ}_{\mathsf{FE}}, \mathsf{LT}_{\mathsf{FE}}\}$, an encryption of the result of h evaluated at the two encrypted vectors. We consider the complexity of the VFE-based algorithms in terms on the number of invocations of component functions $\mathsf{EQ}_{\mathsf{FE}}, \mathsf{LT}_{\mathsf{FE}}$, depth-free automorphisms and multiplications.

For $\mathsf{EQ}_{\mathsf{VFE}}$, we apply $\mathsf{EQ}_{\mathsf{FE}}$ once to obtain the vector $(\mathsf{EQ}_{\mathsf{FE}}(x_{n-1}, y_{n-1}), \dots, \mathsf{EQ}_{\mathsf{FE}}(x_0, y_0))$. Then, we use a variant of SIMD-optimized running product evaluation, replacing $\mathsf{shift\_mul}$ with $\overline{\mathsf{rotate\_mul}(\mathbf{x}, i)} = \mathbf{x} * (x_{n-i}, \dots, x_{n-1}, x_0, \dots, x_{n-i-1})$. This yields the final encrypted result $\underbrace{\overline{(\mathsf{EQ}_{\mathsf{VFE}(x,y)}, \dots, \mathsf{EQ}_{\mathsf{VFE}}(x, y))}}_{n \text{ times}}$.
The procedure above uses $\lfloor \log n \rfloor + 1$ multiplications and rotations. Therefore, $\mathsf{EQ}_{\mathsf{VFE}}$ uses a total of $\lfloor \log n \rfloor + 1$ multiplications and depth-free automorphisms and one invocation of $\mathsf{EQ}_{\mathsf{FE}}$. If we take the results of the $\mathsf{EQ}_{\mathsf{FE}}$ comparisons as fresh inputs, then the computation of the product in Equation (11) has $\lfloor \log n \rfloor + 1$ depth.

As for $\mathsf{LT}_{\mathsf{VFE}}$, we start with one call of $\mathsf{EQ}_{\mathsf{FE}}$ and $\mathsf{LT}_{\mathsf{FE}}$ to get the component-wise comparisons of $\mathsf{VFE.enc}(x)$ and $\mathsf{VFE.enc}(y)$, $(\mathsf{EQ}_{\mathsf{FE}}(x_{n-1}, y_{n-1}), \dots, \mathsf{EQ}_{\mathsf{FE}}(x_0, y_0))$ and $(\mathsf{LT}_{\mathsf{FE}}(x_{n-1}, y_{n-1}), \dots, \mathsf{LT}_{\mathsf{FE}}(x_0, y_0))$, needed by the algorithm. Then, we adjust the vector of equality comparisons to $(1, \mathsf{EQ}_{\mathsf{FE}}(x_{n-1}, y_{n-1}), \dots, \mathsf{EQ}_{\mathsf{FE}}(x_1, y_1))$ by applying a right shift and then adding $(1, 0, \cdots, 0)$ to the result. Then, we apply the SIMD-optimized running product evaluation on the vector $(1, \mathsf{EQ}_{\mathsf{FE}}(x_{n-1}, y_{n-1}), \dots, \mathsf{EQ}_{\mathsf{FE}}(x_1, y_1))$ which requires $\lfloor \log n \rfloor + 1$ multiplications and depth-free automorphisms. Finally, we multiply the running product with $(\mathsf{LT}_{\mathsf{FE}}(x_{n-1}, y_{n-1}), \dots, \mathsf{LT}_{\mathsf{FE}}(x_0, y_0))$ to obtain the individual terms in Equation (12) and apply $\lfloor \log n \rfloor + 1$ shift-and-adds to obtain the final result. Overall, $\mathsf{LT}_{\mathsf{VFE}}$ uses $2\lfloor \log n \rfloor + 3$ depth-free automorphisms, $\lfloor \log n \rfloor + 2$ multiplications and one call to each of $\mathsf{EQ}_{\mathsf{FE}}$ and $\mathsf{LT}_{\mathsf{FE}}$. Like before, taking the outputs of $\mathsf{LT}_{\mathsf{FE}}$ and $\mathsf{EQ}_{\mathsf{FE}}$ as fresh inputs, the algorithm has $\lfloor \log n \rfloor + 2$ depth.

Equality Comparisons with VFE. Two studies have been done on the efficiency of a finite extension field based equality function. The first, by Kim et al. [31], analysed the theoretical depth of such a function and showed that depth-free Frobenius maps can be used to drastically reduce the depth of finite field equality tests, requiring only about $\log p + \log d$ which is comparable to Boolean circuits. The other, by Kim et al. [30], implemented the low-depth algorithm and found that it is quite efficient in practice.

Their key insight is that the Fermat's Little Theorem (Theorem 2) can be used to derive an equality function over elements in $\mathbb{F}_{p^d}$. For any $x, y \in \mathbb{F}_{p^d}$, we know that $x - y = 0 \iff x = y$. Therefore, from Equation (4),

$$\mathsf{EQ}_{\mathsf{FE}}(x, y) = 1 - (x - y)^{p^d - 1} = \begin{cases} 1, & \text{if } x = y; \\ 0, & \text{otherwise.} \end{cases}$$

With depth-free Frobenius map evaluations, we compute $\alpha^{p^d-1} = \alpha^{(p-1)(p^{d-2}+p^{d-3}+\cdots+p+1)}$ optimally in two steps. First, obtain α^{p-1} with $\lceil \log p \rceil$ repeated multiplications and then raise it by $p^{d-2}+p^{d-3}+\cdots+p+1$ with roughly $\lceil \log d \rceil$ repeated Frobenius map evaluations and multiplications. Thus, the VFE-based equality comparison can be obtained with $\lceil \log d \rceil + \lfloor \log n \rfloor + 1$ depth-free automorphisms, $\lceil \log p \rceil + \lceil \log d \rceil + \lfloor \log n \rfloor + 1$ multiplications and depth with SIMD optimizations.

4 An Algorithm for Order Comparisons

Here, we propose an algorithm that is parameterized by r, representing a block size, for evaluating order comparisons between elements in $\mathbb{F}_{p^d}$. The key to this algorithm is a decomposition technique that lets us transform the problem from the large field $\mathbb{F}_{p^d}$ to a small subset $\mathcal{P}_r \subset \mathbb{F}_{p^d}$, where $\mathcal{P}_r = \{\sum_{i=0}^{r-1} a_i t^i \mid a_i \in \mathbb{F}_p\}$ for some r that divides d. In fact, we transform elements in $\mathbb{F}_{p^d}$ into vectors in $(\mathcal{P}_r)^{d/r}$ and apply the principles established in Section 3.2 to evaluate $\mathsf{LT}_{\mathsf{FE}}$.

4.1 Limitations with a Naive Use of Lagrange Interpolation for Order Comparisons

Since the order comparison $\mathsf{LT}_{\mathbb{F}_{p^d}}(x, y)$ for $x, y \in \mathbb{F}_{p^d}$ is a function of two variables, Lagrange interpolation (refer to Theorem 1) can be used to find a 2-variable polynomial $f(x, y)$ that evaluates to it. However, there are huge drawbacks with this approach.

As $|\mathbb{F}_{p^d}| = p^d$, the polynomial $f(x, y)$ constructed from Theorem 1 has degree at most $p^d - 1$ in each variable. Using depth-free Frobenius maps, $f(x, y)$ can be evaluated with an arithmetic circuit of only about $\log p + \log d$ depth. Unfortunately, however the low depth we can achieve is insufficient for an efficient algorithm, with $\mathcal{O}(p^{2d})$ coefficients in $f(x, y)$, computing $f(x, y)$ requires too many multiplications to be efficient for large d. Therefore, to support arbitrary d, we need an alternative.

For an efficient order comparison algorithm, we choose to use Lagrange interpolation differently. Let $r > 0$ such that r divides d,[6] then $\mathcal{P}_r$ is a subspace of $\mathbb{F}_{p^d}$ corresponding to vectors of the form $(a_0, \ldots, a_{i-1}, 0, \ldots, 0)$. When order comparisons are restricted to elements in $\mathcal{P}_r$, the polynomial that evaluates it, denoted by $f_{\mathcal{P}_r}(x, y)$, only has degree at most $|\mathcal{P}_r| - 1 = p^r - 1$ in each variable. This requires about $\log p + \log r$ depth and $\mathcal{O}(p^{2r})$ constant multiplications which is manageable for small r.

4.2 Decomposition: From $\mathbb{F}_{p^d}$ to $(\mathcal{P}_r)^{d/r}$

Since $\mathbb{F}_{p^d}$ is a d-dimensional vector space with $\mathcal{P}_r$ as a subspace, there exists a set of linear maps $\{T_{i,r}\}_{i=0}^{n-r}$ that sends $\alpha = \sum_{j=0}^{d-1} \alpha_j t^j$ to $T_{i,r}(\alpha) = \sum_{j=0}^{r-1} \alpha_{i+j} t^j$. Then, we define the map, Decomp_r, from $\mathbb{F}_{p^d}$ to $(\mathcal{P}_r)^{d/r}$ as follows,

$$\begin{aligned} \mathsf{Decomp}_r :\ & \mathbb{F}_{p^d} \longrightarrow (\mathcal{P}_r)^{d/r} \\ & \alpha = \sum_{i=0}^{d-1} \alpha_i t^i \longmapsto (T_{0,r}(\alpha) = \sum_{i=0}^{r-1} \alpha_i t^i,\ T_{1,r}(\alpha) = \sum_{i=0}^{r-1} \alpha_{r+i} t^i, \ldots, T_{d/r-1,r}(\alpha) = \sum_{i=0}^{r-1} \alpha_{d-r+i} t^i). \end{aligned}$$

For simplicity, we denote the output of $\mathsf{Decomp}_r(\alpha) = (B_0^\alpha, \ldots, B_{d/r-1}^\alpha)$. This map decomposes the coefficients of α into d blocks embedded in $\mathcal{P}_r$, with the order induced by the initial encoding $\mathsf{FE.enc}$ being the lexicographic order on $(\mathcal{P}_r)^{d/r}$. Evaluating $\mathsf{Decomp}(\alpha)$ for any $\alpha \in \mathbb{F}_{p^d}$ is straightfoward from Lemma 3.

Concretely, we first determine the constants $\{\rho_{0,j}, \ldots, \rho_{d-1,j}\}$ that represent the linear map $T_{j\cdot r,r}$ for each $0 \leq j \leq d/r - 1$, which is guaranteed to exist by Lemma 3. Then, we apply powers of the Frobenius map to α, obtaining $\{\alpha, \alpha^p, \ldots, \alpha^{p^d-1}\}$. Finally, we take appropriate linear combinations of them according to the constants $\{\rho_{0,j}, \ldots, \rho_{d-1,j}\}_{0 \leq j \leq d/r-1}$ as follows,

$$\mathsf{Decomp}_r(\alpha) = (\sum_{i=0}^{d-1} \rho_{i,0} \alpha^{p^i},\ \ldots,\ \sum_{i=0}^{d-1} \rho_{i,d/r-1} \alpha^{p^i}). \tag{13}$$

[6] This restriction is only done to simplify the exposition.

4.3 An FE-Based Order Comparison Algorithm

The map Decomp_r allows us to apply the principles of Equation (12), that comparing the order of two elements in $\mathbb{F}_{p^d}$ can be broken down to evaluating order and equality of their components. Notice that the order on $\mathbb{F}_{p^d}$ naturally translates to the lexicographic order on $(\mathcal{P}_r)^{d/r}$. As a result, we obtain the following equation, for $x, y \in \mathbb{F}_{p^d}$,

$$\mathsf{LT}_{\mathsf{FE}}(x, y) = \mathsf{LT}_{\mathcal{P}_r}(B^x_{d/r-1}, B^y_{d/r-1}) + \sum_{i=0}^{d/r-2} \mathsf{LT}_{\mathcal{P}_r}(B^x_i, B^y_i) \prod_{j=i+1}^{d/r-1} \mathsf{EQ}_{\mathcal{P}_r}(B^x_j, B^y_j), \tag{14}$$

where $\mathsf{Decomp}_r(\alpha) = (B^\alpha_0, \ldots, B^\alpha_{d-1})$ for $\alpha \in \{x, y\}$.

From Equation (14), we get the framework to evaluate $\mathsf{LT}_{\mathsf{FE}}$ using Decomp_r, $\mathsf{LT}_{\mathcal{P}_r}$ and $\mathsf{EQ}_{\mathcal{P}_r}$. The algorithm can be split into three steps, for two inputs $x, y \in \mathbb{F}_{p^d}$,

1. process x and y with Decomp_r to get the blocks $(B^x_0, \ldots, B^x_{d/r-1})$ and $(B^y_0, \ldots, B^y_{d/r-1})$;
2. evaluate $\mathsf{LT}_{\mathcal{P}_r}(B^x_i, B^y_i)$ and $\mathsf{EQ}_{\mathcal{P}_r}(B^x_i, B^y_i)$ for $0 \le i \le d/r - 1$;
3. combine the separate results according to Equation (14).

Analysis of the Algorithm $\mathsf{LT}_{\mathsf{FE}}$ with $\mathcal{P}_r$. The correctness of the algorithm follows from the discussions in Section 3.2, just replacing FE with $\mathcal{P}_r$.

We analyze the complexity of the proposed $\mathsf{LT}_{\mathsf{FE}}$ algorithm step by step. First, we decompose the FE-encoded integers x, y with Decomp_r. This yields a length-d/r vector which is obtained by evaluating d/r different linear maps on the FE-encoded inputs per Equation (13). For optimal performance, we first compute the p-th powers of x and y needed, $(x^0, x^p, \ldots, x^{p^{d-1}})$ and $(y^0, y^p, \ldots, y^{p^{d-1}})$, and then combine appropriate linear combinations of them to evaluate the maps $T_{i,r}$ for $0 \le i \le d/r - 1$. Thus, this step only requires $d - 1$ many depth-free automorphisms.

Next, we evaluate equality and order comparisons on the blocks computed in the first step. These are done by evaluating two bivariate polynomials, for $\mathsf{EQ}_{\mathcal{P}_r}$ and $\mathsf{LT}_{\mathcal{P}_r}$, at the "points" (B^x_i, B^y_i), for $0 \le i \le d/r - 1$. From Theorem 1, each of these polynomials has degree $p^r - 1$ in each variable, and therefore we need to compute the monomials $x^2, \ldots, x^{p^r-1}$ and $y^2, \ldots, y^{p^r-1}$. As with the first step, the optimal way is to obtain these monomials once and compute appropriate linear combinations to obtain their evaluations at each polynomial. Overall, this step uses $(p-1)(2\lfloor \log_p p^r - 1 \rfloor)$ depth-free automorphisms and $3p^r - 5$ multiplications per block. Its overall depth is $\log(\lfloor \log_p(p^r - 1) \rfloor + 1) + \lfloor \log(p-1) \rfloor + 1$ using the depth-optimized polynomial evaluation technique described in Appendix A.

Finally, we evaluate Equation (14); note that there no SIMD is used in this computation. Considering the optimized running product evaluation from Section 2.4, in the second step, it iteratively computes $\mathbf{z}_{i+1} = \mathsf{shift\_mul}(\mathbf{z}_i, 2^i)$ for $0 \le i \le \lfloor \log d/r - 1 \rfloor$. In each of this iterations, without SIMD, we need $n - i$ multiplications, since there are i '1's which do not need any processing. Therefore, we require $\sum_{i=0}^{\lfloor \log(d/r-1) \rfloor} (d/r - 1) - i \approx (d/r)\log(d/r)$ multiplications and $\lfloor \log d/r - 1 \rfloor$ depth in Step 3.

In total, $\mathsf{LT}_{\mathsf{FE}}$ requires $d - 1 + 2(d/r)(p-1)\lfloor \log_p p^r - 1 \rfloor \le (2p-1)d - 1$ depth-free automorphisms, $(d/r)\big(3p^r - 5 + \log(d/r)\big)$ multiplications and $\log(\lfloor \log_p(p^r - 1) \rfloor + 1) + \lfloor \log(p-1) \rfloor + \lfloor \log d/r - 1 \rfloor + 1 \le \lfloor \log(p-1) \rfloor + \lfloor \log d \rfloor + 1$ depth.

Balancing Trade-offs with r. In the choice of r, we are essentially balancing the number of blocks we are to process and the complexity of the second step. In the second step, choosing larger r means evaluating higher degree polynomials in $f_{\mathcal{P}_r}$ and $h_{\mathcal{P}_r}$ which are more expensive. Larger r, however, also reduces the number of blocks that have to be processed, reducing the number of linear maps evaluated in the first step and computing a shorter vector of running products in the last step. There are fewer blocks in the second step as well, but the larger polynomials to be evaluated means that it would take longer in general. We will look into the efficiency of our comparison algorithm with respect to the choice of r in Section 6.

4.4 Analysis of a VFE-Based Order Comparison Algorithm

Putting the FE-based order comparison algorithm with the VFE-based framework, we obtain the complete VFE-based order comparison algorithm. With a parameterized FE-based algorithm, the VFE-based algorithm depends on three parameters, vector length n, field degree d and block size r.

The integers from $x, y \in \mathbb{Z} \cap [0, p^{nd})$ are VFE-encoded to $\mathsf{VFE.enc}(x) = (x_0, \ldots, x_{n-1})$ and $\mathsf{VFE.enc}(y) = (y_0, \ldots, y_{n-1})$. These vectors are encrypted in their own ciphertexts and then compared with the algorithm, which consists of evaluating $\mathsf{LT}_{\mathsf{FE}}$ and $\mathsf{EQ}_{\mathsf{FE}}$ with the two ciphertexts as inputs and then aggregating the results according to Equation (12). This is straightforward as the algorithms are described as arithmetic circuits and is easily adapted to the homomorphic operations given by $\{\mathsf{EvalAdd}, \mathsf{EvalMult}, \mathsf{EvalShift}, \mathsf{EvalFrobenius}\}$.

By combining the analysis in Section 3.2 with that in Section 4.3, the VFE-based comparison algorithm with $\mathcal{P}_r$ takes approximately $(2p-1)d + 2\lfloor \log n \rfloor + 2$ depth-free automorphisms, $(d/r)\big(3p^r - 5 + \log(d/r)\big) + \lfloor \log n \rfloor + 2$ multiplications and $\lfloor \log(p-1) \rfloor + \lfloor \log d \rfloor + \lfloor \log n \rfloor + 3$ depth.

5 Private Database Queries with VFE

To highlight the utility of the VFE-based comparisons, we apply them to construct two FHE-based private database query (PDQ) protocols. Such protocols allow users to outsource their database to a cloud while maintaining the privacy and security of their data (and query) through the use of cryptography. The first is a simple application of the techniques from previous sections to retrieve records whose key satisfy an order condition. The second targets more complex compound comparison queries, which shows how the composability of VFE-based algorithms enhances the privacy of PDQ protocols without much overhead. We use (p, d, n)-VFE to denote that integers are VFE-encoded to elements in $(\mathbb{F}_{p^d})^n$.

5.1 Model of the Private Database Query Protocols

Databases $\mathcal{D}$ are modeled as key-value stores, i.e. $\mathcal{D} = \{(v_{\boldsymbol{\rho}_i}, \boldsymbol{\rho}_i)\}$ in this work. The value $v_{\boldsymbol{\rho}} \in \mathbb{Z} \cap [1, p^c)$ is associated with its key $\boldsymbol{\rho}_i = (\rho_{i,1}, \ldots, \rho_{i,k})$, where $\rho_i \in \mathbb{Z} \cap [0, p^c)$ for $i \in \{1, \ldots, k\}$. We assume that $v_{\boldsymbol{\rho}} \neq 0$ for the correctness of our protocols. However, there may be cases where $v = 0$ is needed in practice and we can use the range $\mathbb{Z} \cap [p^c, 2p^c)$ instead.

System and Security Model. The PDQ system model in our setting consists of a user and server. Before the start of the protocol, the user who owns the database encrypts it and stores that encrypted form on the server. When needed, the user sends a query to the server and receives a response with the result of the query performed over the encrypted database stored at the server. There is only a single round of interaction between the user and server.

During the execution of the protocol, the client desires that no partial information about their data and query be revealed to the server or adversary. The adversary model is restricted to the semi-honest case, where all players faithfully follow the actions of the protocol as described. However, they may try to gain additional information, besides the result of the protocol, from transcripts of the execution of the protocol.

Due to space limitations, the formal definitions of PDQ protocols are omitted. We refer interested readers to [23, Chapter 7] for details of the semi-honest adversary model and [30] for the security definitions for PDQ protocols.

5.2 A PDQ Protocol with Single Order Comparison Condition

This first protocol allows a user to retrieve values $v_{\boldsymbol{\rho}_i}$ from the database whose j-th coordinate in their key $\boldsymbol{\rho}_i = (\rho_{i,1}, \ldots, \rho_{i,k})$ is less than some constant Q. This corresponds to the following SQL-like statement:

$$\texttt{select } v_{\boldsymbol{\rho}_i} \texttt{ from } \mathcal{D} \texttt{ where } \rho_{i,j} < Q.$$

The protocol proceeds as follows,

1. The user encodes and encrypts the query constant Q to $\overline{Q}$ and sends the pair $(j, \overline{Q})$ to the server.
2. On receipt of the query, $(j, \overline{Q})$, the server
   (a) runs $\overline{X_i} = \mathsf{LT}_{\mathsf{VFE}}(\overline{\rho_{i,j}}, \overline{Q})$ for $\boldsymbol{\rho}_i \in \mathcal{D}$;
   (b) computes $\overline{\Gamma_i} = \overline{v_{\boldsymbol{\rho}_i}} \cdot \overline{X_i}$ for $\boldsymbol{\rho}_i \in \mathcal{D}$;
   (c) returns $\{\overline{\Gamma_i}\}_{\boldsymbol{\rho}_i \in \mathcal{D}}$ to the user.
3. The user obtains the Γ_i's by decrypting their respective ciphertexts $\overline{\Gamma_i}$.

The correctness of this protocol follows directly from that of the algorithm for LT described in Sections 3 and 4. Compared to the LT algorithm, the protocol only requires an additional multiplication and therefore multiplicative depth to account for Step (c). Besides that, for (p, d, n)-VFE encoding, we need more $\log n$ depth-free automorphisms (due to use of $\mathsf{replicate\_first}(\cdot)$ to propagate the result of $\mathsf{LT}_{\mathsf{VFE}}$ being a vector whose first entry stores the comparison result and all others holding intermediate values) in Step (c).

5.3 A PDQ Protocol with Compound Comparison Conditions

Our second protocol allows a user to retrieve values $v_{\boldsymbol{\rho}_i}$ from the database $\mathcal{D}$ whose key, $\boldsymbol{\rho}_i = (\rho_{i,1}, \ldots, \rho_{i,k})$, satisfy certain conditions, roughly corresponding to the following SQL-like statement:

$$\texttt{select } v_{\boldsymbol{\rho}_i} \texttt{ from } \mathcal{D} \texttt{ where } \rho_{i,j_0} < Q_{j_0} \texttt{and } \rho_{i,j_1} = Q_{j_1} \texttt{ and } \ldots \texttt{ and } \rho_{i,j_r} = Q_{j_z},$$

where $z < k$ and the indices $\{j_0, \ldots, j_z\} \subseteq \{1, \ldots, k\}$.

The protocol proceeds as follows,

1. The user encodes and encrypts the query constants Q_{j_u} to $\overline{Q_{j_u}}$ for $0 \leq u \leq z$ and sends the pairs $\{(i_j, \overline{Q_{j_u}})\}_{j=0}^{z}$ to the server.
2. On receipt of the query, $\{(j_u, \overline{Q_{j_u}})\}_{u=0}^{z}$, the server
   (a) runs $\overline{X_{i,0}} = \mathsf{LT}_{\mathsf{VFE}}(\overline{\rho_{i,j_0}}, \overline{Q_{j_0}})$ and $\overline{X_{i,j_u}} = \mathsf{EQ}_{\mathsf{VFE}}(\overline{\rho_{i,j_u}}, \overline{Q_{j_u}})$ for $1 \leq u \leq z$ and $\boldsymbol{\rho}_i \in \mathcal{D}$;
   (b) computes $\overline{\Gamma_i} = \overline{v_{\boldsymbol{\rho}_i}} \prod_{u=0}^{z} \overline{X_{i,j_u}}$ for $\boldsymbol{\rho}_i \in \mathcal{D}$;
   (c) returns $\{\overline{\Gamma_i}\}_{\boldsymbol{\rho}_i \in \mathcal{D}}$ to the user.
3. The user obtains the Γ_i's by decrypting their respective ciphertexts $\overline{\Gamma_i}$.

We note that the connective between the comparison conditions is not restricted to `and` and can be adapted to the protocols for disjunctive or threshold conjunctive conditions proposed by Kim et al. [30,31].

The correctness of this protocol follows from the correctness of its component algorithms. Depending on z, the protocol might not need additional depth compared to the previous protocol. This is because $\mathsf{EQ}_{\mathsf{VFE}}$ does not require as much depth and we can compute $\prod_{u=1}^{z} \overline{X_{i,j_u}}$ first before multiplying it to $\overline{v_{\boldsymbol{\rho}_i}}$ and $\overline{X_{i,j_0}}$. As for computational complexity, we perform z EQ and $z+1$ multiplications to compute $\prod_{u=1}^{z} \overline{X_{i,j_u}}$ and multiply it to the other terms in Step (c).

6 Implementation Results

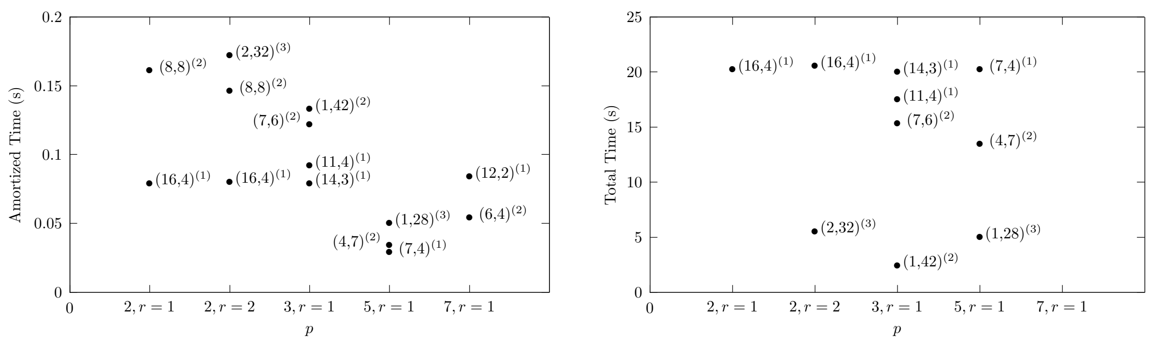
In this section, we present the results of some experiments conducted to evaluate the performance of our proposed algorithms and protocols. First, we detail the experiment platform and parameters used in the experiments. Then, we mention some aspects that were changed from the theoretical descriptions in Sections 3, 4 in our implementation. After that, we introduce and discuss the results of evaluating the performance of the two PDQ protocols proposed in Section 5. Finally, we end the section with some discussion and comparisons with other related work on comparisons with FHE.

6.1 Experiment Platform and Parameter Settings

Experiments were conducted on a server with an Intel® Xeon® Platinum 8170 with a maximum frequency of 3.7 GHz and 192 GB RAM. The following libraries were used to implement the tests: GMP 6.1.2 [24], NTL 11.3.2 [41] (with AVX-accelerated FFT enabled) and HElib (commit 6397b23) [25]. Tests are run using a single thread, with thread boosts in NTL and HElib disabled.

HElib Parameters. In the experiments, we used a variety of FHE instances to test the performance of different (p, d, n)-VFE encodings, focusing on settings which yield estimated $\lambda > 80$ with the LWE estimator (commit 76d05ee)[7] by Albrecht et al. [3] incorporating attacks specific to FHE instances by Albrecht [2]. Note that for fixed m, the smaller L is, the greater the security of the FHE instance. We focused on four small primes, $p = 2, 3, 5, 7$ and used the following instances for each prime:

[7] Available at `https://bitbucket.org/malb/lwe-estimator`.



(a) Amortized Time (Cut-off at 0.2 seconds) (b) Total Time (Cut-off at 25 seconds)

Fig. 1: Figures for Performance of the Protocol in Section 5.2[8]

- $p = 2$: $(m = 21845, L = 13^{(1)}, 17^{(2)}, 19^{(3)})$, $\lambda = 108^{(1)}, 89^{(2)}, 83^{(3)}$;
- $p = 3$: $(m = 14383, L = 13^{(1)}, 15^{(2)}, \lambda = 102^{(1)}, 85^{(2)})$;
- $p = 5$: $(m = 19531, L = 13^{(1)}, 15^{(2)}, 17^{(3)}), \lambda = 117^{(1)}, 111^{(2)}, 100^{(3)}$;
- $p = 7$: $(m = 26419^{(1)}, 30025^{(2)}, L = 17, \lambda = 123^{(1)}, 118^{(2)})$.

In particular, they were chosen because they offered some of the smallest extension fields. The superscripts (i) identify the FHE instances in Figures 1(a) and 1(b) and Tables 1 and 2.

6.2 Implementation Details

The focus is on scalable and real-world applications and so we chose to work with 64-bit integers. When determining the upper bound for other primes $p > 2$, we chose d, n such that $n \cdot d = \lceil 64/\log p \rceil$. This gave us $n \cdot d = 42, 28$ and 24 for $p = 3, 5$ and 7 respectively.

In these experiments, we did not use $\mathbb{F}_{p^d}$ as it is not always possible due to parameter constraints. Instead, we chose parameters where the plaintext space is $\mathbb{F}_{p^{d'}}$ for some $d' \geq d$ and used the subspace $\mathcal{P}_d$. But the same techniques are applicable as we simply exploit the vector space structure on $\mathbb{F}_{p^{d'}}$ for Decomp_r and field operations when evaluating $\mathsf{LT}_{\mathcal{P}}$ and $\mathsf{EQ}_{\mathcal{P}}$. The main impact to the complexity of the algorithms is that the linear maps have $d' > d$ terms and we do not fully utilize the plaintext space.

Also, we did not implement the method to evaluate polynomials with depth-free Frobenius maps for these experiments. It turns out that the range of r that yields efficient comparison is 1 and 2 when $p = 2$ and 1 for $p > 2$. This is because in these algorithms, we are evaluating around p^r polynomials with up to p^r terms each. At $p = 2, r = 3$, we have to evaluate 8 polynomials of degree 8 each per block, as opposed to 1 (resp. 4) polynomial(s) of degree 2 (resp. 4) if $r = 1$ (resp. $r = 2$), but the reduction in the number of blocks is too low to justify. For $p > 2$, setting $r = 2$ already means evaluating p^r polynomials with degree $p^r > 8$ and is not justifiable for similar reasons to the case where $p = 2$.

However, we save some computation when evaluating $\mathsf{LT}_{\mathcal{P}_r}$ and $\mathsf{EQ}_{\mathcal{P}_r}$ by only generating the powers of their inputs once. Furthermore, note that in the cases when $d = r$, $\mathsf{LT}_{\mathcal{P}_r}$ and $\mathsf{EQ}_{\mathcal{P}_r}$ are equivalent to $\mathsf{LT}_{\mathsf{FE}}$ and $\mathsf{EQ}_{\mathsf{FE}}$ and there are fewer steps involved.

6.3 Results for the Protocol in Section 5.2

Our results showed that using $\mathcal{P}_r$ for moderate r is probably the optimal choice for comparisons with VFE encoding, giving a good balance between amortized time and latency (total time). Figure 1(a) shows various (p, d, n)-VFE encodings whose amortized time is under 0.2 seconds per database element, while Figure 1(b) gives the total time for those (p, d, n)-VFE encodings from Figure 1(a) whose total time is under 25 seconds. We also present a comprehensive breakdown of some of the performance results in Table 1.

[8] Labels $(d, n)^{(i)}$ above primes p indicate the point corresponds to the (p, d, n)-VFE encoding with HElib instance (i).

Table 1: Performance of the Protocol in Section 5.2

		$\mathsf{LT_{FE}}$ Algorithm							
(p,d,n,r)	#	Decomp_r	$\mathsf{LT}_{\mathcal{P}_r}$, $\mathsf{EQ}_{\mathcal{P}_r}$	Equation (14)	$\mathsf{EQ_{FE}}$	Equation (12)	Step (c)	Total Time	Amortized Time
(5,7,4,1)$^{(1)}$	697	2.737	13.449	1.857	1.039	1.073	0.062	20.217	0.029
(5,4,7,1)$^{(2)}$	398	2.680	7.638	0.793	1.008	1.260	0.061	13.439	0.034
(5,1,28,1)$^{(3)}$	99	—	*2.240	—	—	2.659	0.075	4.974	0.050
(2,16,4,1)$^{(1)}$	256	10.040	1.789	6.426	0.671	1.234	0.058	20.218	0.079
(2,8,8,1)$^{(2)}$	128	12.925	1.140	3.263	0.900	2.293	0.071	20.592	0.161
(2,1,64,1)$^{(3)}$	16	—	*0.137	—	—	4.617	0.073	4.827	0.302
(2,16,4,2)$^{(1)}$	256	5.701	10.659	2.240	0.659	1.219	0.057	20.535	0.080
(2,8,8,2)$^{(2)}$	128	7.506	6.805	1.111	0.901	2.350	0.074	18.747	0.146
(2,2,32,2)$^{(3)}$	32	—	*1.609	—	—	3.812	0.070	5.491	0.172

# denotes the number of elements packed in a single ciphertext, *: combined $\mathsf{LT_{FE}}$ and $\mathsf{EQ_{FE}}$ times.
Timings are averages of more than 100 runs and given in seconds.

In general, choosing a higher d results in a slower algorithm but is offset by comparing more elements simultaneously. For $p = 2$, choosing $r = 2$ is better in almost every case except at $d = 16$, where the performance is very close (within 2%). For $\mathsf{LT_{FE}}$, the increase in time from evaluating 4 degree-4 polynomials is less than the reduction in the number of linear map evaluations and $\prod_{i=j+1}^{d/r-1} \mathsf{EQ}_{\mathcal{P}_r}$. Especially in the case where there is no decomposition, $(2, 1, 64)$-VFE encoding is only 0.5 seconds faster than $(2, 2, 32)$-VFE encoding but only compares 50% as many elements simultaneously, meaning that the former is more than 75% slower in amortized time.

The results further illustrate the need to consider other bases besides 2 as other bases may have parameters that yield more performant FHE operations even if the algorithms themselves are slightly less efficient. The best results come from using $p = 5$, with the next best amortized performance coming from $p = 7$ but with significantly worse ($> 2\times$) total time taken.

6.4 Results for the Protocol in Section 5.3

In this experiment, we focused on the (p, d, n)-VFE encoding for $p = 5$ since results from the previous experiment indicated that it is by far the best performing base. We set the number of equality conditions $z = 4$ as an estimate of how many equality conditions might be used in practice. Results for this experiments are provided in Table 2.

Table 2: Performance of the Protocol in Section 5.3

				$\mathsf{EQ_{VFE}}$			Total	Result	Amortized	
(p,d,n,r)	z	#	$\mathsf{LT_{VFE}}$	$z\times$	Average	Step (c)	Time	Size	Time	Space
(5,7,4,1)$^{(2)}$	4	697	20.410	4.206	1.052	0.522	25.136	107 KB	0.036	154 B
(5,4,7,1)$^{(3)}$	4	398	21.235	6.421	1.605	0.651	28.307	107 KB	0.071	270 B
(5,1,28,1)$^{(3)}$	4	99	4.293	7.853	1.963	0.414	12.567	107 KB	0.127	1085 B

Timings are averages of more than 100 runs and given in seconds.
Result is a ciphertext which consists of 2 polynomials modulo $\Phi_{19531}(x)$ and $2^{21} < q = 3046837 < 2^{22}$.

First, the performance of $\mathsf{EQ_{VFE}}$ is very good, requiring only 1-2 seconds to compute on average compared to 4-22 seconds for $\mathsf{LT_{VFE}}$. However, one implication is that performing compound conjunctive queries with many equality conditions is better done with (p, d, n)-VFE encodings where d is larger. As seen from the difference in total time, the proportion of time spent computing $\mathsf{EQ}_{\mathsf{VFE}}$ is much less, around 20% for $(5, 7, 4)$-VFE versus almost 200% for $(5, 1, 28)$-VFE. Similarly, the amortized time shows a large jump of more than 200% from results in Table 1 for $(5, 1, 28)$-VFE, going from 0.050 to 0.127 seconds.

Putting everything together, using a moderate degree field extension seems to be the most optimal way; almost $4\times$ improvement in amortized time by paying only $1\times$ increase in total time taken. Results also suggest that adding more equality conditions will only further widen the gap.

As for space complexity, choosing larger d increases the number of integers that can be packed in a single ciphertext. In the case with best performance, with $(5, 7, 4)$-VFE, the protocol only requires an amortized space of 154 bytes to store a single (positive or negative) result. With each result being a 64-bit (8-byte) integer, we see that the space overhead is only $154/8 \approx 19.3\times$. Using simple arithmetic circuits, such as gates in $\mathbb{F}_5$ with $(5, 1, 28)$-VFE in the third row of Table 2, we see an overhead of more than $100\times$ ($1085/8 \approx 135.6\times$) instead.

6.5 Comparisons with Other Works

We implemented the protocol from Section 5.2 with a basic recursive LT algorithm from Cheon et al. [14] with the gate-boostrapped TFHE scheme by Chilotti et al. [16]. They provide a parameter set which is estimated to offer around 128 bits of security.

For $x, y \in [0, 2^{64})$, decomposed into bits, it takes 6.253 seconds to compute LT and 1.584 seconds to compute Step (c). In total, the protocol takes 7.837 seconds to run and process a single database record. Compared to our best result, which takes an amortized time of 0.034 seconds, the TFHE-based system is more than $200\times$ worse. Althrough it takes a long time, it is important to keep in mind that this output can be used in further processing unlike in our experiments.

7 Conclusion

In this work, we introduced a new method of encoding integers for FHE, called the vector of field elements (VFE) encoding. The main motivation for this encoding is to unlock the potential of the native plaintext space of the most widely-used FHE schemes, Brakerski-Gentry-Vaikuntanathan [11], Brakerski [10] and Fan-Vercauteren [20]. The rich $\mathbb{F}_p$-algebra structure of these plaintext spaces has not been considered for computing on encrypted data.

Addressing this, we proposed two algorithms to compute equality ($\mathsf{EQ}_{\mathsf{VFE}}$) and order ($\mathsf{LT}_{\mathsf{VFE}}$) comparisons. The utility of these algorithms are highlighted by designing private database query (PDQ) protocols for simple order comparison and compound comparisons retrieval queries. Experiments conducted demonstrate that these protocols are efficient and the algorithms outperform previous work. We can compute equality (resp. order) comparisons between two 64-bit integers with an amortized time of less than 1.5 (resp. 30) milliseconds. Furthermore, the space overhead is less than $20\times$, which is very close to being practical.

We showed that finite fields can be used for efficient computation on encrypted data. Unlike Jäschke and Armknecht [27], we exploited the vector space structure of finite fields and constructed algorithms that perform better in extension fields. An interesting direction is to extend the techniques of this work to compute addition (and possibly multiplication) of large integers. Also, for greater versatility, it may be useful to incorporate bootstrapping and determine the optimal way to use it with our proposed algorithms.

References

1. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *ACM SIGMOD 2004*, pages 563–574. ACM, 2004.
2. M. R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 103–129. Springer, Heidelberg, 2017.
3. M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3), 2015.
4. Amazon. Amazon relational database service. Available at https://aws.amazon.com/rds/.

5. K. M. M. Aung, H. T. Lee, B. H. M. Tan, and H. Wang. Fully homomorphic encryption over the integers without the sparse subset sum problem. *Theoretical Computer Science*, 2018. Available at `https://doi.org/10.1016/j.tcs.2018.11.014`.
6. A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 224–241. Springer, Heidelberg, Apr. 2009.
7. A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 578–595. Springer, Heidelberg, Aug. 2011.
8. D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu. Private database queries using somewhat homomorphic encryption. In M. J. Jacobson Jr., M. E. Locasto, P. Mohassel, and R. Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 102–118. Springer, Heidelberg, June 2013.
9. D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 563–594. Springer, Heidelberg, Apr. 2015.
10. Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Heidelberg, Aug. 2012.
11. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, pages 309–325. ACM, 2012.
12. M. Bun and M. Zhandry. Order-revealing encryption and the hardness of private learning. In E. Kushilevitz and T. Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 176–206. Springer, Heidelberg, Jan. 2016.
13. N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu. Practical order-revealing encryption with limited leakage. In *FSE 2016*, volume 9783 of *LNCS*, pages 474–493. Springer, Heidelberg, Mar. 2016.
14. J. H. Cheon, M. Kim, and M. Kim. Search-and-compute on encrypted data. In *FC 2015 Workshops*, volume 8976 of *LNCS*, pages 142–159. Springer, Heidelberg, 2015.
15. J. H. Cheon, M. Kim, and M. Kim. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. *IEEE Trans. Information Forensics and Security*, 11(1):188–199, 2016.
16. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Heidelberg, 2016.
17. I. Damgard, M. Geisler, and M. Kroigard. Homomorphic encryption and secure comparison. *Int. J. Appl. Cryptol.*, 1(1), 2008.
18. J. L. Dautrich, Jr. and C. V. Ravishankar. Compromising privacy in precise query protocols. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 155–166. ACM, 2013.
19. F. B. Durak, T. M. DuBuisson, and D. Cash. What else is revealed by order-revealing encryption? In *ACM CCS 16*, pages 1155–1166. ACM Press, Oct. 2016.
20. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.
21. C. Gentry, S. Halevi, C. S. Jutla, and M. Raykova. Private database access with HE-over-ORAM architecture. In T. Malkin, V. Kolesnikov, A. B. Lewko, and M. Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 172–191. Springer, Heidelberg, June 2015.
22. C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 465–482. Springer, Heidelberg, 2012.
23. O. Goldreich. *Foundations of Cryptography: Volume II Basic Applications.* Cambridge University Press, 2004.
24. T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library Version 6.1.2*, 2017.
25. S. Halevi and V. Shoup. HElib: Software library for homomorphic encryption. `http://github.com/shaih/HElib.git`, 2018.
26. Y. Ishimaki and H. Yamana. Non-interactive and fully output expressive private comparison. In *INDOCRYPT 2018*, LNCS, pages 355–374. Springer, Heidelberg, 2018.
27. A. Jäschke and F. Armknecht. (Finite) field work: Choosing the best encoding of numbers for FHE computation. In *CANS 17*, LNCS, pages 482–492. Springer, Heidelberg, 2017.
28. G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. Generic attacks on secure outsourced databases. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 16*, pages 1329–1340. ACM Press, 2016.
29. F. Kerschbaum. Frequency-hiding order-preserving encryption. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 656–667. ACM Press, Oct. 2015.
30. M. Kim, H. T. Lee, S. Ling, S. Q. Ren, B. H. M. Tan, and H. Wang. Better security for queries on encrypted databases. Cryptology ePrint Archive, Report 2016/470, 2016. `http://eprint.iacr.org/2016/470`.

31. 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.
32. M. Lacharite, B. Minaud, and K. G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 1–18.
33. K. Lewi and D. J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In *ACM CCS 16*, pages 1167–1178. ACM Press, Oct. 2016.
34. W. Lu, J.-J. Zhou, and J. Sakuma. Non-interactive and output expressive private comparison from homomorphic encryption. In *ASIACCS 18*, pages 67–74. ACM Press, 2018.
35. M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM CCS 15*, pages 644–655. ACM Press, Oct. 2015.
36. R. Poddar, T. Boelter, and R. A. Popa. Arx: A strongly encrypted database system. Cryptology ePrint Archive, Report 2016/591, 2016. `http://eprint.iacr.org/2016/591`.
37. R. A. Popa, F. H. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *2013 IEEE Symposium on Security and Privacy*, pages 463–477. IEEE Computer Society Press, May 2013.
38. R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In T. Wobber and P. Druschel, editors, *ACM Symposium on Operating Systems Principles (SOSP) 2011*, pages 85–100. ACM, 2011.
39. D. S. Roche, D. Apon, S. G. Choi, and A. Yerukhimovich. POPE: Partial order preserving encoding. In *ACM CCS 16*, pages 1131–1142. ACM Press, Oct. 2016.
40. S. Roman. *Field Theory*. Springer, 2005.
41. V. Shoup. NTL: A library for doing number theory version 11.3.2. Available at http://www.shoup.net/ntl/, 2018.
42. N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *Designs, Codes and Cryptography*, 71(1):57–81, 2014.
43. I. Teranishi, M. Yung, and T. Malkin. Order-preserving encryption secure beyond one-wayness. In *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 42–61. Springer, Heidelberg, Dec. 2014.
44. S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, 6(5):289–300, 2013.

A Depth-Optimized Polynomial Evaluation

Aung et al. [5] gave a method to generate the monomials $x^2, x^3, \ldots, x^n$ with $n-1$ multiplications; for any degree $2 \leq j \leq n-1$, going from lowest to highest, simply multiply x^{2^i} to x^{j-2^i} where $i = \lfloor \log j \rfloor$. Since both monomials are smaller than j, they would already have been generated and it takes only one multiplication to get x^j. This applies to the base $j = 2$ where both monomials are just x and so by induction every monomial from x^2 to x^{n-1} only requires a single multiplication to obtain.

In our case, when working with extensions of $\mathbb{F}_p$, we can use depth-free Frobenius maps to reduce the depth of this procedure when $n-1 \geq p$. For powers of x whose degree is less than p, we stick to Aung et al.'s method and apply depth-free Frobenius maps to get x^{p^i} for all $p^i \leq n-1$. At this point, we have used $p-2$ multiplications and 1 depth-free Frobenius map evaluation. Then, starting from the lowest ungenerated $j = p+1$ to $n-1$, we obtain the remaining x^j by taking the product of two smaller monomials $x^{j'}$ and $x^{j''}$ such that $j' + j'' = j$ and the maximum depth between the two is minimized.

To find the two monomials that yield the minimal depth x^j, we consider a 2-dimensional representation of j, in base p and 2. In base p, we have $j = \sum_{i=0}^{\lfloor \log_p j \rfloor + 1} j_i p^i$. Since the possible digits in base-p are $0 \leq a \leq p-1$, we can obtain x^{ap^i} for $1 \leq a \leq p-1$ and $1 \leq i \leq \lfloor \log_p n \rfloor$ with $a(\lfloor \log_p n \rfloor)$ depth-free Frobenius map evaluations from $x, \ldots, x^{p-1}$. From here, we obtain the remaining monomials by iterating over the Hamming weight of indices. For $2 \leq k \leq (\lfloor \log_p n \rfloor + 1)$, compute all x^i, where the Hamming weight of i is k, by multiplying $x^{i'}, x^{i''}$ where $i' + i'' = i$ and their Hamming weights are around $k/2$. With this, we compute all monomials $x^2, \ldots, x^n$ using $n-1$ multiplications and $(p-1)(\lfloor \log_p n \rfloor)$ depth-free automorphisms. The depth required for this computation is $\log(\lfloor \log_p n \rfloor + 1) + \lfloor \log(p-1) \rfloor$.

To evaluate univariate polynomials $f(x) = \sum_{i=0}^{n-1} f_i x^i$, we simply compute the necessary monomials $1, x, \ldots, x^i$ using the steps above and take appropriate linear combinations of the monomials generated. For bivariate polynomials $f(x, y) = \sum_{i=0}^{n_x - 1} \sum_{j=0}^{n_y - 1} f_{i,j} x^i y^j$, we consider it as a univariate polynomial in $\mathbb{F}_{p^d}[x]$, $f_x(y) = \sum_{j=0}^{n_y - 1} f_j(x) y^j$, where $f_j(x) = \sum_{i=0}^{n_x - 1} f_{i,j} x^i$ and evaluate $n_x + 1$ many univariate polynomials, assuming that $n_x \leq n_y$.

In the bivariate case, we use $n_x + n_y - 4$ multiplications and $(p-1)(\lfloor \log_p n_x \rfloor + \lfloor \log_p n_y \rfloor)$ depth-free automorphisms to obtain the required monomials. Since the polynomials $f_j(x)$ are univariate polynomials where the coefficients are constants, we do not need multiplications or depth. Finally, we use $n_y - 1$ multiplications to evaluate the univariate polynomial $f_x(y)$ for a total of $n_x + 2n_y - 5$ multiplications. The depth of the procedure is $\log(\lfloor \log_p n_y \rfloor + 1) + \lfloor \log(p-1) \rfloor + 1$ since we assumed that $n_x \leq n_y$.